Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Hyper and hypo glycaemia detection based on bag of words

Document: Memòria

Alumne: Eduard Berloso Clarà

Tutor: Beatriz Lopez Ibañez

Departament: Enginyeria Elèctrica, Electrònica i Automàtica

Àrea: ESA

Convocatòria (mes/any): juny / 2017

INDEX

1 INTRODUCCIÓ, MOTIVACIONS, PROPÒSIT I OBJECTIUS

1.1 Introducció

El treball desenvolupat en aquest P/TFC s'emmarca dins del grup de recerca eXiT.

El treball consisteix en la construcció d'un sistema capaç de predir la hiperglucèmia i la hipoglucèmia en pacients diabètics abans que aquesta es produeixi, utilitzant per aquest fi diferents tècniques de Intel·ligència Artificial aplicades a dades històriques dels pacients.

1.1.1 Grup de Recerca eXiT

El grup de recerca eXiT (Enginyeria de Control i Sistemes Intel·ligents) és un grup interdisciplinari de l'Institut d'Informàtica i Aplicacions de la Universitat de Girona.
Un dels principals camps de recerca del grup és el de la Intel·ligència Artificial i les aplicacions que pot tenir aquesta a l'hora de solucionar problemes del mon real. En aquest aspecte un dels àmbits en el que més s'ha treballat és en el de la medicina i en com millorar les condicions de vida de la gent.

El grup de recerca eXiT participa en diferents projectes d'àmbit nacional i internacional sobretot en projectes relacionats amb el camp de la medicina.

1.1.2 Prevenció dels nivells de glucosa

Un dels projectes en que està treballant el grup de recerca eXiT és en la detecció i prevenció dels episodis d'hiperglucèmia i hipoglucèmia en pacients que pateixen diabetis.

La diabetis es una afecció crònica que es desencadena quan l'organisme perd la capacitat de produir suficient insulina o de utilitzar-la amb eficàcia.

Dos dels problemes més comuns que es presenten amb la diabetis son la hipoglucèmia, quan el nivell de sucre en sang és molt baix, i la hiperglucèmia, quan el nivell de glucosa en sang és massa elevat.

Els pacients amb diabetis han d'estar controlant constantment el seu nivell de glucosa en sang per tal de detectar i corregir les hipoglucèmies i les hiperglucèmies. La majoria tenen un sensor capaç de mesurar el nivell de glucosa en temps real. D'aquesta manera poden saber quan s'està produint un dels dos problemes exposats anteriorment.

L'objectiu d'aquest projecte és desenvolupar un sistema capaç de predir els episodis d'hiperglucèmia i hipoglucèmia abans que aquests es produeixin (i siguin detectats per el sensor).
En concret s'estudia l'aplicació de diferents tècniques de *Machine learning* sobre dades històriques obtingudes de pacients (a través dels sensors que aquests porten) per construir un sistema capaç de detectar partons i seqüències en les corbes de glucosa i així predir quan es produirà una hiperglucèmia o hipoglucèmia. A més, es pretén utilitzar una aproximació a la tècnica anomenada «bag of words» per tal de caracteritzar aquestes seqüències i poder fer així les prediccions.

A més de la construcció del sistema predictiu, també es vol oferir una manera fàcil de poder accedir a aquest sistema. Per això també es construirà un portal web on es podran entrar les dades necessàries amb facilitat per tal de obtenir les prediccions sobre l'estat del pacient i fer un seguiment més acurat.

1.2 Motivacions

La principal motivació alhora de escollir aquest P/TFC és que volia participar en un projecte de recerca per aprendre i veure de primera ma com funciona un grup de recerca.

Ja he treballat (i treballo en aquest moment) en empreses de informàtica del mon laboral però mai havia treballat en un grup de recerca i penso que és bo i necessari haver passat per les dues experiències.

En segon lloc, volia augmentar els meus coneixements en l'àmbit de la Intel·ligència Artificial. Durant la carrera de Enginyeria Informàtica s'ensenyen multitud de tècniques utilitzades en la IA, a més d'organitzar-se conferències en que es mostra el gran potencial d'aquestes, però no es pot aprofundir en cap d'elles. Al fer un P/TFC enfocat en l'àmbit de la Intel·ligència Artificial tinc la oportunitat de treballar, estudiar i utilitzar algunes d'aquestes tècniques.

En tercer lloc, com a usuari habitual del llenguatge de programació Python, un projecte d'aquestes característiques em permet aprofitar i aprendre a utilitzar la gran quantitat de eines i llibreries que el python ofereix relacionades tan amb l'àmbit del tractament de dades (pandas, numpy, etc.) com de la Intel·ligència Artificial (p.e. scikit-learn), a més de les relacionades amb la construcció de APIs per entorns client-servidor (flask, django, etc.).

Per últim, el fet que el projecte estigui relacionat amb l'àmbit de la Medicina, es a dir, que la finalitat sigui millorar la qualitat de vida de la gent suposa una motivació extra alhora de escollir un projecte d'aquestes característiques.

1.3 Propòsit

El propòsit d'aquest projecte és estudiar la utilització de tècniques de *Machine learning* sobre dades històriques de pacients per a la detecció i identificació de seqüències en les corbes de glucosa de pacients amb diabetis. Aquestes dades són obtingudes a través d'uns sensors que porten els pacients amb diabetis i que mesuren el nivell de glucosa d'aquests.

Es vol desenvolupar un sistema intel·ligent capaç de predir els episodis d'hiperglucèmia i hipoglucèmia basant-se en la detecció d'aquestes seqüències.

Finalment també es vol construir un portal web per on els metges podran entrar noves dades obtingudes per els sensors. Aquetes dades seran introduïdes en el sistema construït i es mostrarà als metges si el pacient te perill o no de sofrir un episodi d'hiperglucèmia o hipoglucèmia en el futur. A més es mostrarà la seqüència completa que s'ha predit, donant així informació sobre la corba de glucosa que es preveu que tindrà el pacient.

En resum el propòsit del projecte inclou l'estudi, l'anàlisi, el disseny i el desenvolupament del sistema intel·ligent per fer prediccions a més del disseny i la implementació de l'aplicació client-servidor per comunicar-se amb aquest sistema.

1.4 Objectius

Els objectius del P/TFC presentat són:

- Estudiar i trobar la millor manera de modelitzar i tractar amb les dades referents a les lectures de glucosa obtingudes dels sensors dels pacients. Les corbes de glucosa son series temporals, un tipus de dades molt irregulars, amb soroll i difícil de modelitzar. Per tal de poder-les utilitzar en el sistema predictiu s'ha de trobar la millor manera de caracteritzar aquestes dades. En concret s'estudiarà la utilització d'una aproximació de la tècnica de «bag of words».

- Fer l'anàlisi, el disseny i la implementació del sistema intel·ligent que permeti obtenir prediccions sobre els episodis d'hiperglucèmia i hipoglucèmia a partir de dades històriques dels pacients. Rebent com a input les lectures de glucosa del sensor d'un pacient després de haver realitzat un àpat (a més de dades extra com el nivell inicial de glucosa, carbohidrats ingerits, insulina injectada, informació sobre l'exercici realitzat i si s'ha ingerit alcohol, etc.) aquest sistema ha de ser capaç de fer una predicció sobre el risc de hiperglucèmia i hipoglucèmia. Es farà un sistema independent a la interfície de usuari proporcionant un seguit de mètodes per entrar dades en el sistema i extreure'n.

- Fer l'anàlisi, el disseny i la implementació de l'aplicació servidor amb la qual els metges es comunicaran (a través d'un navegador web com per exemple Firefox, Chrome, Opera, etc.) per entrar noves dades al sistema, visualitzar aquestes dades i visualitzar el resultat de la predicció feta per el sistema.

- Fer el disseny i la implementació de les interfícies (pàgines web) i les crides per comunicar-se amb el servidor de l'aplicació client.

La obtenció de dades del sensor i el posterior bolcat en un fitxer de text que serà després utilitzat a través del portal web desenvolupat queden fora de l'abast d'aquest P/TFC.

2 ESTUDI DE VIABILITAT

En aquest apartat s'exposen els elements previstos com a necessaris per al desenvolupament d'aquest projecte. Cal remarcar que el present treball es tracte d'un projecte de recerca, que busca determinar la viabilitat i eficàcia d'una tècnica més que desenvolupar un producte. Per aquest fet, l'estudi de viabilitat realitzat no ha estat exhaustiu en termes d'aspecte econòmic ni de recursos humans, això queda fora de l'abast del present P/TFC.

Per al desenvolupament d'aquest projecte es preveuen com a necessaris els següents elements, classificats en 3 grups::

- Coneixements:
    - Enginyeria del software.
    - Programació orientada a objectes en Python
    - Construcció de pàgines Web (HTML, Javascript, ...)
    - Tècniques d'Inteligència Artificial (tot i que un dels objectius és ampliar els coneixements en aquest camp, es requereix una base prèvia)
- Recursos en Hardware:
    - Una màquina servidor amb un mínim de 4 GB de ram, S.O. Ubuntu 16.04 i Python 3.5 instal·lat.
    - Una màquina client amb navegador web.
- Recursos en Software:
    - Python 3.5 i accés a les seves llibreries.
    - Bootstrap

Respecte els recursos de hardware, la màquina client i la servidor poden ser la mateixa sempre i quan es compleixin els requisits de les dos màquines, reduint-se així els recursos necessaris a una sola màquina.

Pel que fa als recursos en software, cal remarcar que tot el projecte ha estat desenvolupat utilitzant software lliure i gratuït, per tant el cost econòmic és 0.

3 METODOLOGIA

Per desenvolupar aquest P/TFC es va decidir seguir la metodologia àgil combinada amb la metodologia de treball RUP, la qual s'exposarà en les següents seccions.

3.1 RUP

La metodologia RUP (Rational Unified Process) és un procés de enginyeria del software que proporciona un manera de assignar tasques i responsabilitats en el desenvolupament d'una aplicació o producte. Es tracte d'una metodologia de desenvolupament iterativa enfocada als diagrames de casos d'ús.

3.1.1 Principis del RUP

La filosofia del RUP està basada en 6 grans principis:

- Adaptar el procés: el procés s'ha de adaptar a les necessitats del client i no al revés. És molt important interaccionar amb el client per poder determinar be la mida i l'abast del projecte.
- Equilibrar prioritats: durant el procés varis desenvolupadors poden necessitar els mateixos recursos, és important definir la prioritat de cada tasca per determinar la utilització d'aquests recursos.
- Valor de les Iteracions: durant el desenvolupament es realitzen varies iteracions. Al final de cadascuna d'elles el projecte es presenta (encara que sigui internament) per tal de analitzar-ne l'estat actual: saber la opinió dels participants, analitzar-ne l'estabilitat i qualitat, etc. si fa falta també es redefineix la direcció que ha de prendre el projecte.
- Col·laboració entre equips: en el desenvolupament del projecte participen varies persones i equips de persones, es fonamental una bona comunicació entre tots i cadascun dels participants per tal de coordinar requisits, desenvolupament, resultats, etc.

- Enfocar-se en la qualitat: el control de qualitat no s'ha de realitzar només al final de les iteracions, sinó en cada aspecte del desenvolupament.
- Elevar nivell de abstracció: utilitzar representacions visuals de la arquitectura com per exemple UML.

3.1.2 Cicle de Vida

En un projecte desenvolupat en RUP es fa una implementació en espiral.
Es divideix el procés en varies fases, i cadascuna d'aquestes fases es divideix en iteracions. Les iteracions solen ser poques i grans. Cada iteració dona com a resultat un producte executable (o una part del producte final que sigui executable autònomament).

Cada cop que es finalitza el cicle de vida del RUP s'analitza el producte. Si es detecten nous requeriment es realitza un altre cop tot el cicle per introduir els nous requeriments.

3.1.3 Fases del RUP

Es defineixen 4 fases, cada fase s'encarrega d'unes tasques del procés de desenvolupament:

- Fase de Inici: es defineixen els requeriments i l'abast del projecte. S'elabora una visió general de l'arquitectura base del sistema i es planifiquen les següents fases.
    - Diagrames de casos d'ús
    - Especificació de requisits
    - Diagrama de requisits
- Fase d'Elaboració: es seleccionen, analitzen i defineixen els casos de ús necessaris per a poder definir l'arquitectura base del sistema. Es tracta d'una de les fases més importants ja que s'estableix la arquitectura de tot el projecte.
    - Diagrama de classes
    - Model Entitat-Relació
    - Diagrama de Seqüències
    - Diagrama d'Estats

- Fase de Desenvolupament: en aquesta fase es fa la implementació dels casos d'us definits en la fase anterior mitjançant una sèrie de iteracions.
- Fase de Transició: es realitzen els tests necessaris per garantir que el producte està ben acabat i preparat per entregar al usuari.

3.2 Elecció del RUP

A l'hora de escollir la metodologia a seguir les primeres idees que van sorgir son metodologies àgils com Scrum. Es tracte d'una metodologia amb la que estic familiaritzat i molt útil per treballar i desenvolupar en equips. El problema és que scrum es tracta de metodologies molt enfocades en el treball en equips de persones i el P/TFC es tracte d'un projecte individual, per tan perdien el seu gran avantatge. A més l'exigència que demanen en relació al treball organitzatiu i de disseny previs a la implementació del projecte son molt baixos.

Per altra banda, amb la metodologia RUP (tot i estar pensada també per el treball en equips) es defineixen unes fases clares en el desenvolupament del projecte, cosa més propera a les meves necessitats. A més és una metodologia pensada per treballar amb diagrames UML, diagrames amb els que ja estic familiaritzat.

3.3 Aplicació de la metodologia

El present P/TFC es tracte d'un projecte més enfocat a la recerca que en el desenvolupament d'un producte comercial per ser distribuït.

Tot i que si que existeix una part del projecte destinada a fer el producte amigable per a possibles usuaris finals (tot el relacionat amb l'aplicació client-servidor explicada en la introducció) la gran part del treball realitzat està destinat a la construcció d'un bon sistema predictiu. Aquest fet s'ha traslladat a l'hora de aplicar la metodologia de treball.

Es va dividir el P/TFC en dos subprojectes:

- Aplicació Client-Servidor: construcció de l'aplicació servidor i les interfícies del client per a la visualització de dades i la interacció amb el sistema predictiu.
- Aplicació Sistema Predictiu: construcció d'un seguit de classes, mètodes i funcions que defineixen un sistema capaç de realitzar prediccions.

En l'aplicació client-servidor es va treballar seguint la metodologia RUP de forma tradicional: definir els requeriments a partir de les necessitats dels usuaris finals (els metges), dissenyar els diferents diagrames UML que defineixen l'estructura de l'aplicació client servidor, fer la implementació i finalment provar el correcte funcionament del producte. En aquest cas cadascuna de les fases va constar solament d'una sola iteració ja que els requeriments estaven perfectament definits i no en podien sorgir de nous.

En l'aplicació sistema predictiu es va seguir utilitzant la metodologia RUP però amb varies particularitats a tenir en compte:

- En la fase de inici, per tal de poder definir els casos d'us del sistema es va treballar com si el client final fos l'aplicació client-servidor. Es va adoptar aquesta decisió ja que és l'aplicació client-servidor qui interactua amb el sistema predictiu, mai directament els clients. Per tant les necessitats les que s'havien de satisfer (els requisits) venien marcats per l'aplicació client-servidor.
- En la fase de desenvolupament, es va realitzar una iteració per a cadascuna de les diferents formes en que es va implementar el sistema predictiu. Al tractar-se d'un treball de recerca no existia una sola manera de implementar el sistema, sinó que l'objectiu era trobar la millor manera de implementar-lo utilitzant diferents algorismes i modalitzacions de les dades per obtenir els millors resultats. Aquest fet va quedar plasmat en les iteracions de la fase de desenvolupament.
- En la fase transició es va procedir de la mateixa manera que en la fase de desenvolupament: per cada iteració de la fase de desenvolupament es va fer una iteració en la fase de transició en que es testejava i mesurava la qualitat de les prediccions del sistema.

4 PLANIFICACIO

En aquest apartat s'explica la planificació seguida en aquest P/TFC per tal de assolir els objectius establerts en la introducció.

La planificació del projecte es divideix en dos etapes.

La primera etapa es va dedicar al estudi de les dades amb les que es partia al inici del projecte. Es tracte de les dades obtingudes a través dels sensors de glucosa que portaven pacients amb diabetis. A més també es va dedicar a treball bibliogràfic sobre diferents tècniques de Intel·ligència Artificial utilitzades en el P/TFC com «bag of words» (tècnica que, com s'ha explicat en els objectius, s'està intentant adaptar).

La segona etapa i la més important és en la que es desenvolupa la resta del projecte: la part del client-servidor i la part del sistema predictiu. En aquesta etapa és en la que es posa en pràctica la metodologia de desenvolupament RUP explicada en l'apartat de metodologia. Cal destacar que abans de començar a treballar en la part de l'aplicació client-servidor es va treballar en la construcció d'una primera versió del sistema predictiu. Un cop es va obtenir aquesta primera versió es va treballar en el client-servidor i gràcies a aquest treball van sorgir nous requeriments per la part del sistema predictiu.

La planificació presenta les característiques i consideracions següents:

- El P/TFC es va començar el dia 30/01/2017
- La planificació inicial es va pensar per entregar el P/TFC al Juny
- Es dedica una mitjana de 4 hores al dia, ja sigui al laboratori del grup eXiT com a casa.
- Els dies de treball son de dilluns a divendres, reservant els caps de setmana per avançar en la present memòria.
- Hi ha setmanes que per qüestions laborals no és possible treballar en el P/TFC

A continuació es llisten les tasques planificades i les hores de dedicació previstes. Moltes d'aquestes tasques es van realitzar en paral·lel.

Les tasques de cadascuna de les etapes explicades anteriorment son les següents:

- **Etapa 1**
    - **Estudi bibliografia**: lectura de diferents articles relacionats amb la Intel·ligència Artificial. Temps de dedicació previst: 12 hores.
    - **Anàlisis de les dades obtingudes per els sensors de glucosa**: Temps de dedicació previst: 8 hores.
    - **Preposés de dades**: implementació algorisme per preprocessar les dades i transforma-les en un format més adequat per el posterior treball. Temps de dedicació previst: 20 hores.
- **Etapa 2**
    - **Anàlisi disseny del sistema predictiu**: Temps de dedicació previst: 20 hores.
        - Diagrama de classes + Patrons
        - Diagrama Entitat-Relació
    - **Implementació primera versió del sistema predictiu**: Temps de dedicació previst: 60 hores.
        - Generació del vocabulari
        - Construcció de seqüències
        - Construcció del model predictiu
    - **Anàlisi de requeriments aplicació client-servidor:** Temps de dedicació previst: 12 hores.
        - Interfícies de usuaris
        - Diagrama de casos d'us
    - **Anàlisi i disseny aplicació client-servidor:** Temps de dedicació previst: 12 hores.
        - Diagrama de Classes i Base de Dades
        - Patrons
        - Disseny API del servidor

- **Implementació aplicació client-servidor:** Temps de dedicació previst: 56 hores.
  - Construcció de les interfícies (pàgines HTML) del client
  - Implementació del servidor (construcció de la API)
  - Implementació comunicació client-servidor a través de la API
- **Anàlisi dels nous requeriments del sistema predictiu:** durant la implementació de l'aplicació client-servidor està previst que sorgeixin nous requeriments per a l'aplicació del sistema predictiu. Temps de dedicació previst: 12 hores.
  - Diagrama casos d'us
  - Revisió Diagrama de classes
- **Implementació nous requeriments del sistema predictiu**: Temps de dedicació previst: 28 hores.
- **Millorar el sistema predictiu**: es tracte de realitzar noves implementacions del sistema predictiu utilitzant diferents algorismes i modelitzant les dades de formes diferents per tal de obtenir els millors resultats. Temps de dedicació previst: 80 hores.
- **Implementació Tests**: implementació d'una classe utilitzada per testejar i mesurar el nivell de qualitat del sistema predictiu. Temps de dedicació previst: 20 hores.
- **Redacció de la memòria**: redacció d'aquest document incloent els diagrames fets durant tot el projecte. Temps de dedicació previst: 72 hores.

La dedicació total planificada és de 412 hores.

A continuació es mostra un diagrama de Gantt on es mostren les tasques planificades (Fig 4.1).

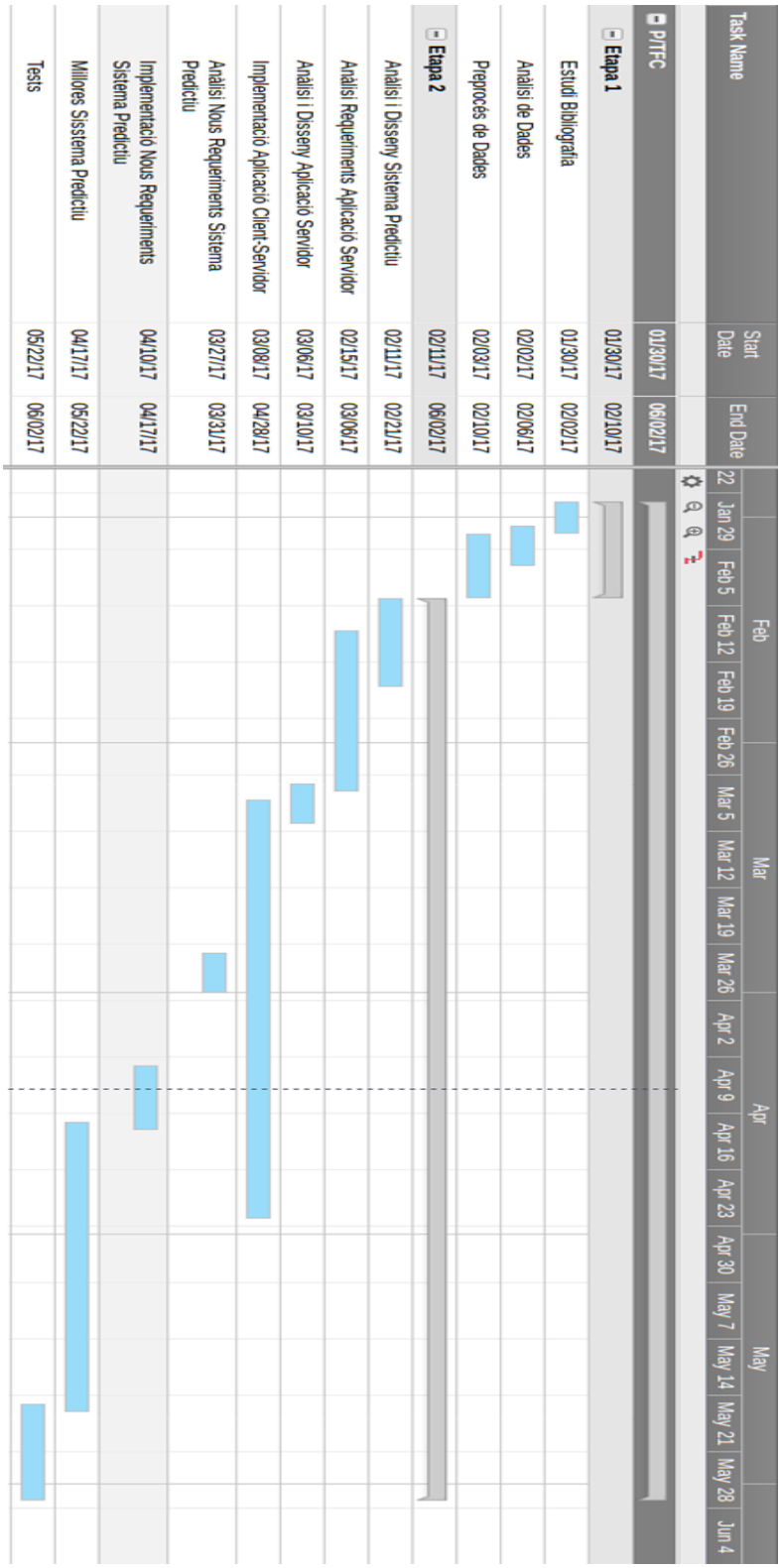


Figura 4.1: Diagrama de Gantt

5 MARC DE TREBALL I CONCEPTES PREVIS

En aquest apartat s'explicarà l'entorn en el que s'ha desenvolupat aquest P/TFC a més de diversos conceptes que permetran entendre millor el present treball.

En primer lloc es parlarà de la diabetis, els pacients i els sensors que aquests porten. En segon lloc es parlarà del equip de treball relacionat amb aquest P/TFC. Per últim s'explicaran un seguit de conceptes relacionats amb l'àmbit de la Intel·ligència Artificial que apareixen al llarg de la present memòria.

5.1 Pacients Diabètics

La diabetis és un conjunt de trastorns metabòlics que resulta generalment en alts nivells de glucosa en sang. Aquest nivell anormal de glucosa es pot produir per varies raons: l'organisme és incapaç de produir prou insulina, l'organisme presenta una gran resistència als efectes de la insulina o l'organisme produeix un nivell anormal de glucosa. En algunes ocasions poden aparèixer varies raons alhora.

Existeixen varis tipus de diabetis, però els més importants son els següents [1]:

- Diabetis tipus 1: és aquella en la que l'organisme no produeix insulina degut a la destrucció de les cèl·lules beta del pàncreas encarregades de la seva producció. Es tracte de la més freqüent en persones joves (fins a 25 anys) i es calcula que afecta a un 5-10% del total de malalts de diabetis.
- Diabetis tipus 2: és aquella en que l'organisme presenta una gran resistència als efectes de la insulina provocant la pujada del nivell de glucosa. A diferencia que la de tipus 1, no es tracte d'una malaltia de caràcter auto-immune, sol anar associada a la obesitat i és freqüent en majors de 40 anys. Es tracte del tipus de diabetis més comú: representa un 90-95% del total de malalts de diabetis.

En aquest P/TFC s'ha treballat amb pacients de diabetis tipus 1.

Al tractar-se d'una malaltia crònica, el tractament de la diabetis (tan la de tipus 1 com la de tipus 2) consisteix en mantenir els nivells de glucosa normals: entre 70 i 150 mg/dl (aquests valors son susceptibles a canvis degut a factors com l'edat).

Un tractament complet de la diabetis inclou:

- Dieta sana
- Evitar ingesta de sucres ràpids
- Exercici físic moderat

Combinant aquests factors amb les injeccions adequades de insulina, es mantenen els nivells de glucosa normals i s'eviten les complicacions com [1][2]:

- Hipoglucèmia: produïda quan el nivell de sucre en sang és molt baix. Pot aparèixer degut a un esforç físic no habitual, sobredosis de insulina, ingesta insuficient de hidrats de carboni, etc. Es tracte d'una complicació molt perillosa i és molt important la ràpida ingesta de sucres per contrarestar-la.
- Hiperglucèmia: produïda quan el nivell de glucosa en sang és massa elevat. Pot aparèixer degut a un error en la dosificació de la insulina, un excés de hidrats de carboni o per la ingesta de medicaments que produeixen un augment en els nivells de glucosa.

Per tan, és essencial mantenir els nivells de glucosa en valors normals. Amb aquesta finalitat molts pacients de diabetis porten uns sensors que mesuren en temps real el nivell actual de glucosa en sang. Gràcies a això poden detectar quan aquests nivells son anormals, és a dir, quan s'està produint una hiperglucèmia o hipoglucèmia.

Tal i com s'ha explicat en l'apartat de la introducció, l'objectiu del present treball es detectar els nivells anormals de glucosa abans que aquests es produeixin i siguin detectats per el sensor per evitar tan la hipoglucèmia com la hiperglucèmia.

Les dades amb les que s'ha treballat son les obtenies per sensors que portaven pacients de diabetis tipus 1. Aquests sensors envien la següent informació en intervals de 5 minuts:

- Insulina injectada
- Carbohidrats ingerits
- Nivell de glucosa

A més d'aquestes dades, també es consta de informació extra referent als pacients i a la seva situació abans i després d'un àpat:

- Referent al pacient:
    - Edat
    - Sexe
- Referent al àpat:
    - Ingesta de alcohol: valor binari Si/No
    - Exercici abans del àpat: valor binari Si/No
    - Exercici després del àpat: valor binari Si/No

5.2 Equip de Treball

El present P/TFC s'ha desenvolupat en l'entorn del grup de recerca eXiT, un grup interdisciplinari de l'Institut d'Informàtica i Aplicacions de la Universitat de Girona que participa en diversos projectes relacionats amb la Intel·ligència Artificial molts cops aplicada al camp de la Medicina i a la millora de la qualitat de vida de la gent.

Dins aquest grup de recerca he col·laborat i comptat amb el suport de:

- Beatriz Lopez com a tutora del present P/TFC
- Natalia Mordanyuk com a suport al laboratori

5.3 Conceptes i Tècniques en Intel·ligència Artificial

Com ja s'ha dit, l'objectiu del present P/TFC és construir un sistema predictiu basat en la detecció de seqüències i les dades històriques del pacients. En la construcció d'aquest sistema s'utilitzen i combinen varies tècniques de Intel·ligència Artificial:

- Cada conjunt de lectures de glucosa llegides després d'un àpat, al que anomenarem sessió a partir d'ara, es assignat a un clúster. Aquests clústers es formen utilitzant tot el conjunt de sessions disponibles utilitzant la tècnica anomenada k-means.
- Per obtenir i buscar seqüències de sessions similars s'utilitzen les tècniques del *k-nearest-neighbour* i el *suport vector machine*.
- Cada seqüència de sessions és representada com un conjunt de etiquetes o "paraules" fent una aproximació de la tècnica "bag of words".

A continuació s'expliquen amb més detall en que consisteixen cadascuna de aquestes tècniques.

5.3.1 Clustering

Clustering és el procés de agrupar dades en classes o clústers de manera que els objectes dins d'un clúster siguin molt similars entre ells i al mateix temps es diferenciïn molt dels objectes de la resta de clústers.

Un dels algorismes de clustering més coneguts és el K-means [3]. La seva gran avantatge és la simplicitat de aplicació i la gran eficàcia que demostra. Segueix un procediment simple de classificació d'un conjunt de objectes en un determinat número K de clústers. El valor de K es fixa a priori.

En el k-means cada clúster es representa pel seu centroide. Si es representa gràficament els clústers obtinguts per aquest algorisme, s'observaria que el centroide es troba al centre dels elements que formen el clúster (Fig. 5.3.1).

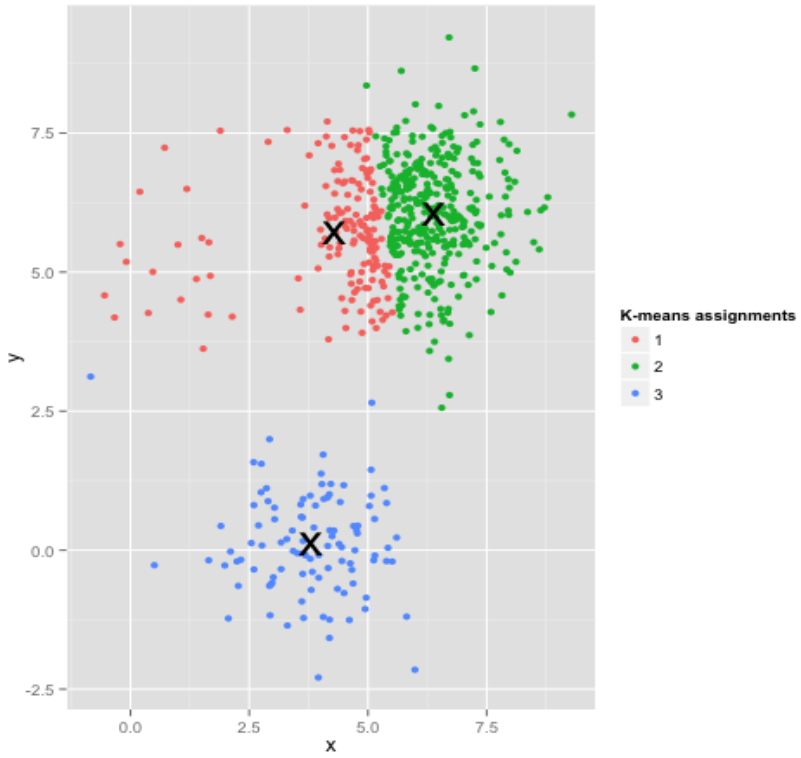


Figura 5.3.1: Representació K-Means amb 3 clusters

L'algorisme del k-means es desenvolupa en 4 etapes:

- Escollir aleatòriament K objectes per formar els K clústers inicials. En aquesta etapa els valors dels clústers son el valor dels objectes que els formen respectivament.
- Assignar cadascun dels objectes existents a un dels k clústers formats. Per assignar un objecte x a un clúster es calcula la distancia a cadascun dels clústers i s'assigna al que tingui la distancia més petita. Per calcular la distancia d'un objecte a un altre la mesura utilitzada sol ser la distancia euclidiana.
- Quan ja s'ha assignat cada objecte a un clúster, es tornen a calcular els centres dels k clústers.
- S'ha de repetir les etapes 2 i 3 fins al moment en que després d'haver calculat els centres dels clústers (etapa 3) ja no es fa cap reassignació d'un objecte a un altre clúster en el que no estava (etapa 2).

L'algorisme K-means presenta una complexitat NP-hard. Si sabem el nombre *k* de clústers i el nombre *d* de dimensions, es pot resoldre en un temps $O(n^{dk+1})$, on *n* es el número de objectes a agrupar [4].

Tot i que l'algorisme k-means sempre acaba, no es garanteix obtenir la solució més optima possible. És molt sensible a la elecció aleatòria dels centroides inicials. Per minimitzar aquest problema es sol aplicar l'algorisme varis cops de forma successiva sobre el mateix conjunt de dades, perjudicant així la seva eficàcia. Un altre inconvenient és el d'haver de fixar a priori el número K de clústers ja que es tracte d'una informació que a la pràctica no es coneix.

5.3.2 K-Nearest-Neighbour

L'algorisme *k-nearest-neighbour* o knn és un mètode de classificació supervisada que es basa en la suposició que els objectes més propers (veïns) seran de la mateixa classe que un mateix [3][5]. Utilitzant tots els objectes que ja es tenen classificats dins del sistema, es classifiquen els nous objectes mesurant la distancia amb la resta de objectes (Fig. 5.3.2).

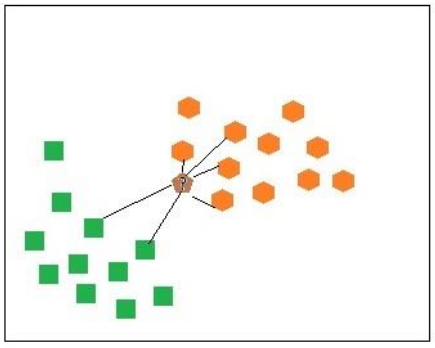


Figura 5.3.2: Representació K-NN amb K = 6

La idea bàsica del algorisme és:

- Agafar la informació del objecte que s'ha de classificar
- Calcular les distancies amb tots els altres objectes existents
- Escollir el K objectes més propers. El valor de K s'ha fixat anteriorment
- A partir dels K objectes més propers escollits es classificarà el nou objecte

A l'hora de implementar l'algorisme knn s'ha de prendre dos decisions que afectaran molt a la seva eficàcia: escollir el mètode per calcular la distància i fixar el valor de K.

El tipus de distància més adient per knn depèn de les dades amb les que s'està treballant. Per exemple, si tenim atributs com ingressos mensuals, deutes, altres ingressos, etc, llavors una distancia euclidiana seria una bona mesura ja que tots els atributs es troben en la mateixa escala i rang de valors. Per altra banda si combinéssim atributs que utilitzen escales i rangs diferents com pes, edat, número de fills, ingressos mensuals, deutes, etc. una millor manera de calcular la distancia seria utilitzant la distancia de mahalanobis. Una altre opció seria escalar tots els atributs en el mateix rang de valors.

En la elecció del valor de K ens tornem a trobar en la mateixa situació que en el cas del k-means: depèn de les dades amb les que estiguem treballant i la seva distribució. En general es solen realitzar varis tests per determinar el valor de k. Tot i això, en general una elecció d'una K major sol augmentar la correctesa de les prediccions. També és bo la elecció d'una k imparell per empats.

Un altre punt important en la implementació de l'algorisme és el factor de cerca. Si tenim un sistema amb milers d'objectes seria inviable calcular la distancia amb tots ells. Per solucionar-ho hi ha dos maneres: escollir subconjunt d'objectes amb els més "icònics" de cada classe utilitzats per fer la classificació o definir un rang de cerca que limiti els objectes amb els que comparar.

5.3.3 Suport Vector Machine

El 'Suport Vector Machine' (SVM) és un algorisme de aprenentatge supervisat utilitzat en problemes de classificació i regressió [6][7][8].
En aquest algorisme, es representa cada objecte com un punt en un espai de n dimensions (on n és el número de característiques que defineixen els objectes) agafant com a valor per les coordenades els valors de cada característica de l'objecte. Llavors es troba un hiperplà capaç de

dividir correctament els objectes de cada classe (Fig. 5.3.3). D'aquesta manera, quan arriba un nou objecte al sistema, per classificar-lo només és necessari determinar a quin costat del pla es troba.

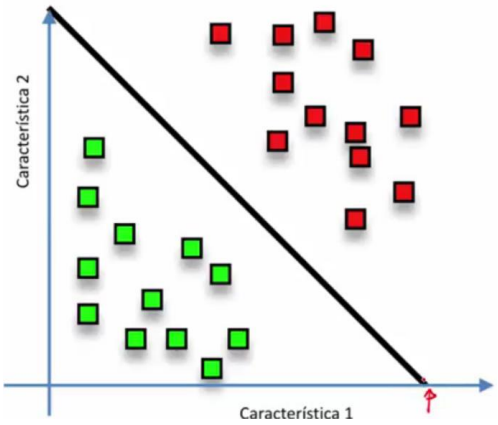


Figura 5.3.3: Representació SVM amb 2 classes

Per tant, quan s'implementa el SVM el problema està en trobar l'hiperplà frontera entre els objectes de cada classe. Aquest hiperplà frontera es troba utilitzant un conjunt de entrenament format per objectes suficientment representatius. Per mostrar com es pot trobar aquest hiperplà es farà un exemple continuant amb la situació mostrada en la Figura 5.3.3 on tenim 2 classes i es vol obtenir una frontera lineal entre elles:

- Si tenim 2 classes i volem una frontera lineal no existirà una manera única de definir aquesta frontera (Fig. 5.3.4):

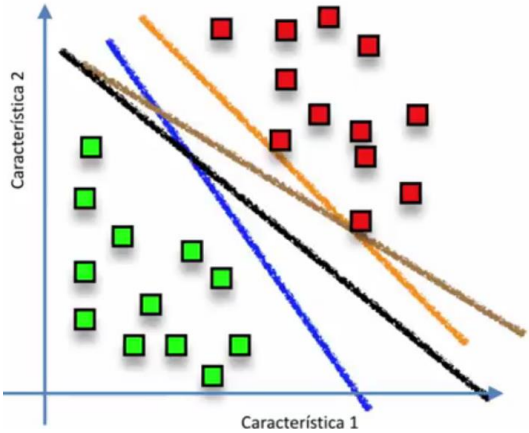


Figura 5.3.4: Representació SVM amb 2 classes i múltiples possibles fronteres

- Per trobar la solució òptima el primer pas és obtenir per cada classe del conjunt de entrenament un número limitat de objectes que presentin unes determinades propietats i als que anomenarem vectors de suport (Fig. 5.3.5):

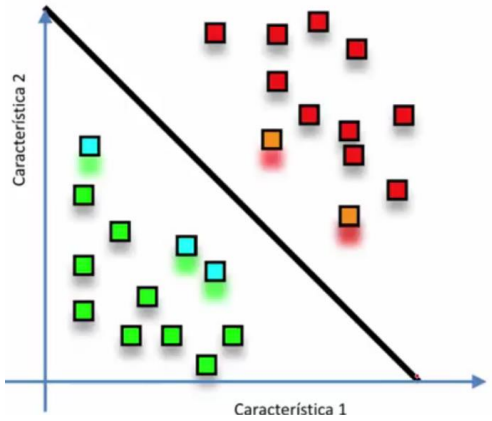


Figura 5.3.5: Representació SVM amb 2 classes i objectes dels vectors de suport destacats

- Amb els vectors de suport es defineixen dos hiperplans. La zona entre els dos hiperplans formats estarà buida de mostres (Fig. 5.3.6):

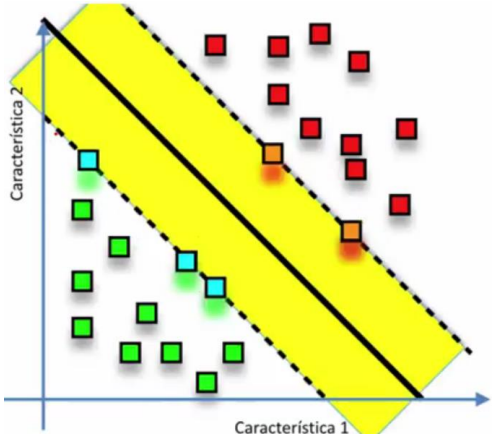


Figura 5.3.6: Representació SVM amb 2 classes i el marge entre els vectors de suport destacat

- Quan la solució és l'òptima la distancia entre els dos hiperplans que formen cadascun dels vectors de suport de cada classe serà màxima. Per tant, si calculéssim la distancia entre els vectors de suport de cadascuna de les fronteres mostrades en la figura 5.3.4, la solució òptima seria la que obtingués una major distancia entre els vectors de suport.

L'exemple explicat correspon al model més bàsic de SVM en que la frontera és una funció lineal i no hi ha solapament entre els objectes de diferents classes. Una situació així no es sol donar en els casos pràctics. Per això el SVM té mètodes per tractar amb altres tipus de funcions frontera. Existeixen diferents tipus de kernels que defineixen diferents tipus de fronteres:

- Polinomial
- Perceptron
- Gaussià

5.3.4 Bag of Words

El model "bag of words" és un mètode utilitzat en el processament de llenguatge per representar documents ignorant l'ordre de les paraules. La idea bàsica és representar un document com un histograma amb el número de cops que apareix cada paraula.

Aquesta idea es porta sovint al camp del reconeixement de objectes per representar imatges. Una imatge pot ser tractada com un document i les característiques extretes de certs punts de la imatge son considerades com paraules.

El seu funcionament és el següent:

- Identificar les paraules rellevants
- Definir un vocabulari amb les paraules més freqüents
- Es compta les ocurrències de cada paraula del vocabulari dins el document
- Cada document es representa com un histograma del vocabulari

En el present P/TFC s'ha intentat adaptar aquesta tècnica per tal de representar les sessions (conjunt de lectures de glucosa just després de d'àpat en un pacient). Així dons un document correspondria a una seqüència de sessió i les paraules serien les característiques extretes de la corba de glucosa corresponent a la sessió.
El problema en l'adaptació d'aquesta tècnica ha estat la extracció i representació de les característiques de les corbes de glucosa degut a la irregularitat d'aquest tipus de corbes.

Una altre aproximació que s'ha realitzat ha estat la de considerar que cada corba de glucosa és una paraula i els documents són les seqüències de sessions. En aquest cas per tal de reduir-la mida del vocabulari i no tenir una paraula diferent per cada corba de glucosa que hi ha en el sistema s'ha aplicat clustering (k-means) per generar un vocabulari de K paraules. Finalment s'ha etiquetat cada sessió amb una paraula del vocabulari.

6 REQUISITS DEL SISTEMA

En aquest apartat s'exposaran els requeriment que ha de complir el sistema resultant, tan funcionals com no funcionals. Seguint amb la metodologia RUP aquests requisits s'han anat complint en fases i de forma iterativa.

Com s'ha explicat en anteriors apartats, s'han desenvolupat dos sistemes independents que interactuen entre ells: l'aplicació client-servidor i el sistema predictiu. Tenint en compte aquest fet es mostraran els requisits de cada component per separat.

Tan els requeriments funcionals com els no funcionals s'han obtingut a través de reunions amb la tutora del projecte Beatriz Lopez.

6.1 Requeriments funcionals

Els requeriments funcionals són aquells que indiquen el que l'aplicació ha de fer. Estan relacionats directament amb les necessitats del client ja que defineixen el que l'usuari vol realitzar a través de l'aplicació.

6.1.1 Aplicació Client-Servidor

Els requeriments funcionals que s'han determinat per l'aplicació client-servidor son els següents:

- Carregar les dades de les lectures de glucosa obtingudes pels sensors a través d'un fitxer de text.
- Mostrar les dades llegides per el sensor en un plot continu, mostrant altre informació com dades descartades per soroll o els nivells que es consideren hiperglucèmia i hipoglucèmia.

- Mostrar la seqüència generada amb les dades de la nova sessió formada ja per les paraules del vocabulari generat. Ha de indicar si hi ha una hiperglucèmia o hipoglucèmia o si per el contrari ens trobem en un estat normal.
- Mostrar totes les paraules del vocabulari generat.
- Mostrar el resultat de la predicció junt amb el nivell de confiança.
- Mostrar les seqüències més semblants amb les quals es basa la predicció.

6.1.2 Sistema Predictiu

Els requeriments funcionals que s'han determinat per l'aplicació del sistema predictiu son els següents:

- Classificar una sessió en hiperglucèmia/hipoglucèmia/normal. Es tracte d'una classificació multietiqueta.
- Predir la paraula assignada a una sessió.
- Introduir noves dades al sistema a través d'un fitxer de text amb la informació necessària.
- Donada una sessió obtenir la informació de la seqüència en la qual aquesta sessió és la última de les sessions que la seqüència.
- Donada una seqüència obtenir les seqüències més similars a aquesta.
- Ha de generar un vocabulari de n paraules a partir de la informació de les sessions.

6.2 Requeriments No Funcionals

Els requeriments no funcionals són aquells que descriuen com ha de ser el programa resultant i no pas el que aquest ha de fer.

Estan més relacionats amb el disseny del sistema i no pas amb les seves funcionalitats.

6.2.1 Aplicació Client-Servidor

Els requeriments no funcionals que s'han determinat per l'aplicació client-servidor son els següents:

- Ha de funcionar en els diferents navegadors més usuals: mozilla firefox, google chrome, opera, etc.
- El temps de resposta entre el client i el servidor ha de trobar-se dins d'uns nivells acceptables per al usuari. En el servidor es realitzen una gran quantitat de operacions per tal de complir els requeriments de l'aplicació, tot i això s'ha de vigilar que el temps de resposta es mantingui sempre en uns nivells assumibles per l'usuari.
- Tot el tractament de dades dels pacients i tot el relacionat amb la realització de les prediccions es realitza en el sistema predictiu i no en l'aplicació servidor. El servidor només interacciona amb el sistema predictiu mitjançant mètodes i funcions definides en el sistema predictiu. En resum, tota la lògica de les prediccions es realitza fora de l'aplicació del servidor, el servidor només s'encarrega d'obtenir dades i construir les pàgines web retornades al client.
- El client es comunica amb el servidor fent peticions a una API construïda per el servidor.

6.2.2 Sistema Predictiu

Els requeriments no funcionals que s'han determinat per l'aplicació del sistema predictiu son els següents:

- Dissenyar l'arquitectura de manera que el tractament de dades i el mètode de predicció (KNN, SVM, etc.) siguin independents. Per fer això el sistema divideix les tasques de tractament de dades i de predicció en 2 classes diferents: CGMLoader i CGMPredicter.
  En la classe CGMLoader s'implementen tots els mètodes relacionats amb la introducció de dades noves, construcció de sessions, construcció del vocabulari, construcció de seqüències, etc.
  En la classe CGMPredicter s'implementen els mètodes necessaris per fer prediccions utilitzant les dades proporcionades per la classe CGMLoader. Gràcies aquesta separació

es poden desenvolupar i implementar diferents mètodes de predicció sense preocupar-se per la obtenció de les dades en el format adequat ja que sempre serà el mateix i estarà implementat en el CGMLoader.

- Utilitzar un sistema de gestió de bases de dades. Degut a la gran quantitat de dades amb les que es tracte no era viable treballar llegint i processant les dades sempre des d'un fitxer de text. Per això és necessari la utilització i construcció d'una base de dades amb la que el sistema predictiu tractés per obtenir les dades.

7 ESTUDIS I DECISIONS

En aquest apartat es descriuran el maquinari, les llibreries i el programari que s'han utilitzat durant el desenvolupament del present P/TFC així com les raons d'haver escollit cadascuna de les llibreries utilitzades.

7.1 Programari

En aquesta secció s'exposaran les llibreries utilitzades, es farà un breu resum del seu funcionament i s'explicarà la seva utilitat dins del projecte.

7.1.1 Python

Tot el projecte, exceptuant algunes parts de l'aplicació client-servidor que s'han desenvolupat en java-script, ha estat desenvolupat en Python 3.5 (la ultima versió estable publicada en el moment de iniciar el projecte).

El python és un llenguatge de programació d'alt nivell i propòsit general [9]. Una de les seves principals característiques és que està enfocat en la llegibilitat del codi de manera que es puguin expressar conceptes complexos en més poques línies que en altres llenguatges com C o Java.
Una altre característica de python és que tot i ser molt utilitzat com a llenguatge de script, suporta una gran quantitat de paradigmes de programació com: programació orientada a objectes, imperativa i funcional.

Els motius de l'elecció del llenguatge de programació python per al desenvolupament d'aquest projecte son les següents:

- Es tracte d'un llenguatge amb el que ja estava familiaritzat i amb el que ja he treballat. Per tant és un llenguatge que ja es dominava la qual cosa beneficia el desenvolupament del projecte evitant gastar temps en l'aprenentatge d'un llenguatge nou.

- El python compta amb una llibreria estàndard molt gran (és una de les seves principals fortaleses). Aporta una gran quantitat de mètodes, classes i funcions ja programades que poden realitzar una gran quantitat de tasques.
- En els últims anys el python ha anat agafant importància en el camp de la Intel·ligència Artificial fins a convertir-se en un dels més utilitzats en aquest camp. Python ofereix una gran quantitat de llibreries molt potents relacionades amb multitud de camps de la Intel·ligència Artificial. Moltes d'aquestes llibreries s'han utilitzat en el present P/TFC i s'explicaran més endavant.

7.1.2 Scikit-learn

Scikit-learn és una llibreria que ofereix una gran quantitat d'algorismes de aprenentatge supervisat i no supervisat [10]. Ofereix una interfície en python per utilitzar aquests algorismes de forma senzilla i personalitzable a través dels paràmetres.

Va ser inicialment desenvolupada per David Cournapeau, actualment consta amb més de 30 col·laboradors actius.

Durant el desenvolupament del present P/TFC s'han utilitzat els següents mòduls de la llibreria scikit-learn:

- Sklearn.preprocessing: ofereix mètodes per preparar i prerocessar les dades per a algorismes de Machine learning. En el projecte s'han utilitzat les classes MinMaxScaler i StandardScalert per a preprocessar les dades sobre les lectures llegides pels sensors.

- Sklearn.cluster: proporciona algorismes de clustering supervisats i no supervisats. En concret s'ha utilitzat classe KMeans. Es tracte d'una classe que implementa l'algorisme amb el seu mateix nom. En el projecte s'utilitza aquesta classe per generar el vocabulari amb que s'etiqueta les sessions a partir del total de sessions existents en el sistema.

Ofereix molts paràmetres per personalitzar el comportament de l'algorisme, alguns dels més importants i que s'han utilitzat en el projecte són:

- N_clusters: número de clústers a generar
- Init: tipus de inicialització dels clústers
- Tol: tolerància d'error
- N_init: número de temps (minuts) que l'algorisme s'executarà amb cada llavor

- Sklearn.neighbors: en aquest mòdul s'implementa l'algorisme k-nearest-neighbours. Aquest algorisme ja s'ha explicat en l'apartat de 5.3.2. S'utilitza per comparar seqüències de sessions i fer així prediccions sobre si es produirà hiperglucèmia o hipoglucèmia. Entre d'altres paràmetres, deixa personalitzar el número de veïns amb els que comparar i els pesos de cada variable.

- Sklearn.svm: en aquest mòdul s'implementa l'algorisme suport vector machine. El funcionament de l'algorisme s'ha explicat en l'apartat 5.3.3. Igual que el k-nearest-neighbour, s'utilitza per comparar les seqüències de sessions i predir si es produirà hiperglucèmia o hipoglucèmia. Ofereix la possibilitat de triar el tipus de kernel utilitzat en l'algorisme: lineal, polinòmic, sigmod, etc.

- Sklearn.model_selection: aquest mòdul ofereix la classe StratifiedKFold. Aquesta classe és utilitzada per testejar models mitjançant cross validation. La tècnica de cross validation s'explica en posteriors apartats.

- Sklearn.metrics: ofereix diferents eines per mesurar el nivell de qualitat d'un sistema de classificació. En concret s'ha utilitzat els mètodes per generar matrius de confusió que llavors s'utilitzen per extreure'n diferents mètriques sobre la qualitat del sistema.

7.1.3 Sqlitle3

Sqlitle es un paquet de software de domini públic que ofereix un sistema de gestió de bases de dades relacional [11]. Un sistema de bases de dades relacional s'utilitza per emmagatzemar registres definits per l'usuari. A més pot processar consultes complexes que combinen dades de varies taules per generar informes i resums de dades.

La principal característica de sqlitle és que a diferència que amb altres programes de gestió de bases de dades com postgres i mysql, no te una estructura client-servidor. Sqlitle és un sol programa independent que funciona a través de crides simples a subrutines i funcions.
El conjunt de la base de dades (definicions, taules, índexs, etc.) es guarden en un sol fitxer de text en la pròpia màquina que s'està executant.

Sqlitle ofereix un mòdul de python per tractar i gestionar bases de dades [12]. Per aquesta raó i també per la senzillesa que ofereix el fet de no treballar en una estructura client-servidor és perquè s'ha escollit aquest sistema per a gestionar la base de dades del projecte.

7.1.4 Pandas

Pandas és una llibreria de python utilitzada per a la manipulació i anàlisis de dades [13]. Ofereix unes estructures de dades flexibles i que permeten treballar de forma molt eficients:

- Series: arrays unidimensionals amb indexació. Són similars als diccionaris.
- DataFrame: són estructures de dades similars a les taules de bases de dades relacionals sql.
- Panel: estructures que permeten treballar amb més de dos dimensions.

Durant el projecte s'han utilitzat Series i DataFrames alhora de tractar amb les dades. Pandas ofereix un mètode gràcies al qual es poden generar DataFrames a partir de consultes SQL. Això ha resultat molt útil ja que ofereix una forma de tractar amb les dades que s'extreuen e la base de dades.

7.1.5 LmFit

LmFit és un paquet de python que proporciona eines senzilles per a la construcció de models de funcions no lineals a partir de dades reals [14].

El seu funcionament és el següent: es defineix una classe Model que representarà una funció no lineal. Aquesta classe conté tan la formula que defineix aquesta funció com els paràmetres que intervenen en aquesta formula.

Donats un seguit de dades lmfit és capaç de trobar els valors dels paràmetres que més s'ajustin a aquestes dades.

Consta amb un seguit de models ja predefinits: GaussianModel, LorentzianModel, VoigtModel, etc. De tots ells el més interessant per aquest projecte és el primer: GaussianModel.
Una de les formes en que s'ha representat una sessió és com una distribució gaussiana de les seves lectures de glucosa. Per trobar aquesta funció s'ha utilitzat LmFit.

7.1.6 Plotly

Plotly és una llibreria de python per a la visualització gràfica de dades [15]. Permet la creació de gràfiques de qualsevol tipus. Les gràfiques generades son interactives, permetent entre d'altres coses ocultar i mostrar línies, realitzar zoom a les zones desitjades, etc.

Les gràfiques es generen en forma de codi html perfecte per ser integrat en altres pàgines web, fet que s'ha utilitzat en el present P/TFC. Totes les gràfiques d'aquest projecte s'han desenvolupat utilitzant aquesta llibreria i s'han integrat en la pàgina web del client per a la seva visualització.

7.1.7 Flask

Flask és un framework escrit en python que permet la creació d'aplicacions web de manera ràpida i senzilla [16][17][18]. Es tracte d'un framewok minimalista ja que no incorpora eines extres per realitzar tasques comuns en el desenvolupament web, sinó que consta del mínim i necessari per fer funcionar una aplicació.

Incorpora un servidor web de desenvolupament de manera que no es necessitin altres programes com Nginx o Apache. També ofereix un sistema de plantilles (Jinja2) molt complet per crear pàgines web dinàmicament.

Flask és el framework utilitzat en el projecte per a la creació de l'aplicació client-servidor. Flask no te ORMs, wrappers o configuracions complexes, això el converteix en un gran candidat per crear aplicacions àgils com és el cas del present projecte. A més el sistema de plantilles Jinja2 és perfecte per generar les pàgines web que s'envien al client ja que comunicant-se amb l'aplicació del sistema predictiu es poden obtenir dades de la base de dades i generar contingut dinàmicament.

7.1.8 Bootstrap

Bootstrap es un framework de codi obert per dissenyar aplicacions web [19]. Proporciona plantilles de disseny de tipografia, formularis, botons, etc. basats en html, css i javascript.
Es tracte d'un dels frameworks més utilitzats en el desenvolupament web.

En l'aplicació client-servidor, les pàgines web que es creen al servidor i s'envien al client es dissenyen utilitzant bootstrap. Gràcies a això, s'ofereix un disseny uniforme i adaptable dinàmicament a la mida del dispositiu en que s'executa.

7.2 Maquinari

S'ha utilitzat un servidor amb les següents prestacions:

- Processador: Intel Core i7-6700HQ CPU 2.60 GHz
- Memòria Ram: 8 GB
- Espai Lliure al Disc: 500 GB
- Sistema Operatiu: Ubuntu 16.04 LTS

Tot i les prestacions descrites anteriorment, les aplicacions desenvolupades en aquest projecte estan preparades per ser executades en qualsevol Sistema Operatiu en que hi hagi instal.lat python 3.5.

8 ANÀLISI I DISSENY DEL SISTEMA

En capítol s'exposa l'anàlisi del sistema basat en els diagrames de casos d'ús, les fitxes de casos d'us i els diagrames de activitats referents a les diferents tasques dels sistemes.

També s'exposa el disseny basat en els models entitat relació, els diagrames de classes, els patrons utilitzats i el disseny de les interfícies d'usuaris.

Tan l'anàlisi com el disseny es divideixen en dos apartats més, un per cada sistema desenvolupat: sistema predictiu i aplicació client-servidor.

8.1 Anàlisi

8.1.1 Aplicació Client-Servidor

Diagrama de Cas d'Ús

En aquest apartat es mostrarà un diagrama de casos d'ús general de l'aplicació client-servidor (Figura 8.1). En aquest diagrama es mostra el comportament de l'aplicació i les situacions en que es pot trobar. Alguns aspectes a tenir en compte sobre el diagrama:

- L'usuari és una persona que es comunicarà amb el servidor a través d'un navegador web.
- El diagrama fa referencia al servidor, ja que en el client no hi ha cap tipus de lògica. Només és un navegador web que rep pàgines HTML amb la informació ja processada per el servidor.
- Els casos d'ús marcats en verd indiquen que es tracte d'una tasca que el servidor demana a l'aplicació del sistema predictiu.

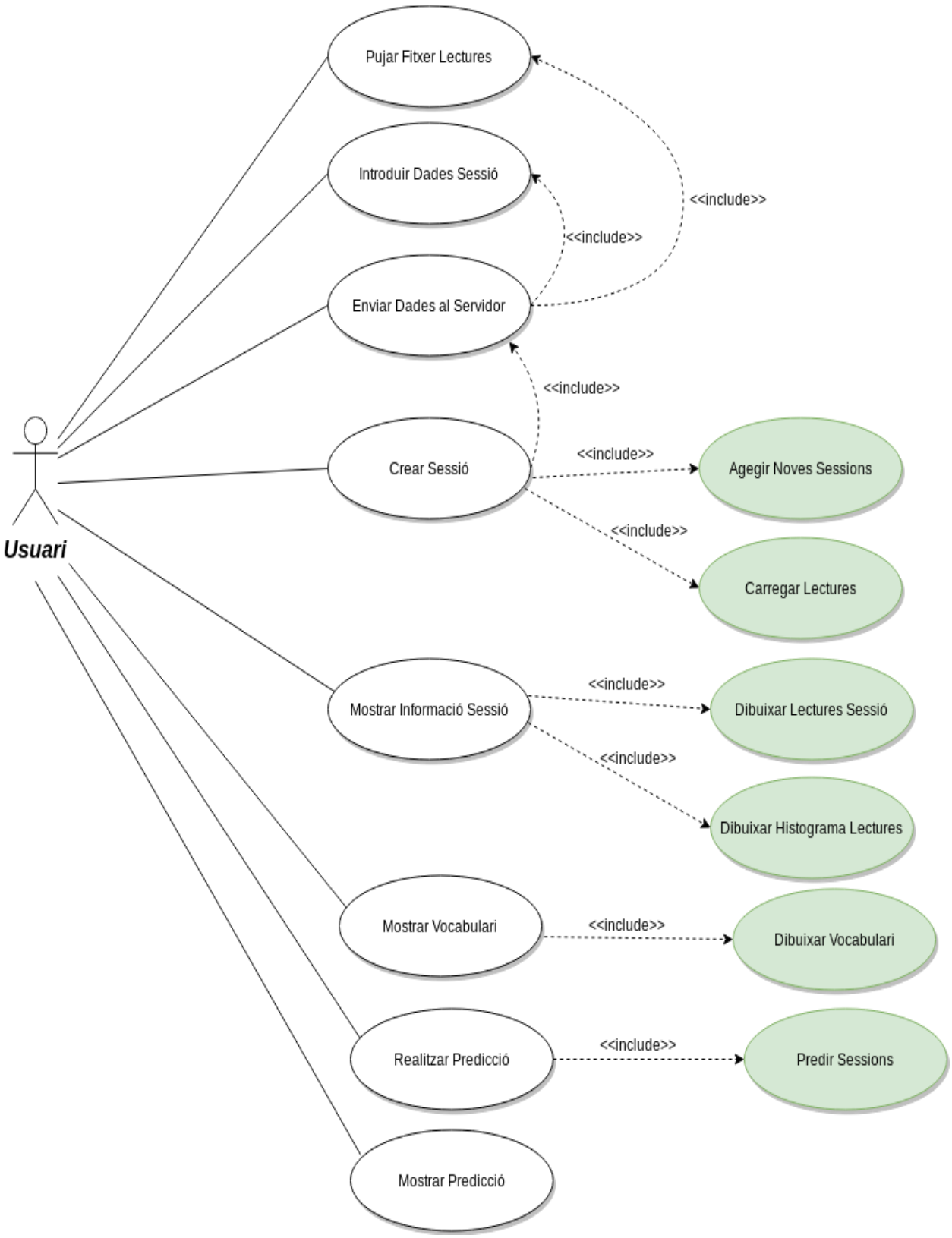


Figura 8.1: Diagrama de cas d'ús del Servidor

Fitxes de Cas d'Ús

En aquest apartat es desenvolupen les fitxes de cas d'us que descriuen el flux principal del funcionament de l'aplicació per a realitzar prediccions, des de la introducció de dades fins a la visualització dels resultats. Aquest flux principal inclou les següents fitxes de cas d'ús:

1. Enviar Dades al Servidor
2. Crear Sessió
3. Mostrar Informació Sessió
4. Realitzar Predicció
5. Mostrar Predicció

Fitxa de cas d'ús – Enviar Dades al Servidor

Descripció	Enviar dades al servidor
Actor	Usuari amb navegador web
Precondició	-
Flux Principal	1. Usuari obre navegador i introdueix ruta de l'aplicació. 2. El servidor envia pàgina HTML amb la pàgina de inici on es mostra un formulari per entrar les dades de la sessió. 3. Entrar dades: • File: seleccionar fitxer amb les dades de les lectures anteriors a l'àpat actual del pacient. • Patient: Pacient al que pertanyen les dades (mitjançant desplegable). • Date: data amb hora i minuts en que s'ha realitzat l'apat. • Insulin: quantitat de insulina injectada. • Carbohydrates: quantitat de carbohidrats ingerits. • Exercice Before: Si/No, si el pacient ha fet esport abans de menjar.

	• Exercice After: Si/No, si el pacient ha fet esport després de menjar. • Alcohol: Si/No, si el pacient ha ingerit alcohol durant el menjar. • Glucose: nivell de glucosa inicial. 4. Usuari prem el botó de "Continuar". S'envien les dades al servidor parsejades utilitzant Json. 5. Validar les dades entrades des del servidor. a. En cas que alguna dada no sigui valida (format de la data incorrecte, no s'ha seleccionat fitxer, insulina negativa, etc.) s'envia un missatge d'error al client i no es canvia de pàgina. b. Si es validen totes les dades correctament el servidor continua el flux creant la sessió i el client es manté a l'espera d'un nou missatge.
Postcondició	El client no ha rebut cap missatge que indiqui que les dades no son correctes i es manté a l'espera d'un nou missatge del servidor.

Fitxa de cas d'ús – Crear Sessió

Descripció	Crear Sessió
Actor	Usuari amb navegador web
Precondició	El servidor te totes les dades necessàries per a crear noves sessions. Aquestes dades han estat validades.
Flux Principal	1. Es crea un nou identificador (ID) per a la sessió del nou àpat combinant el nom del pacient i la data de la sessió. 2. Es construeix una llista L amb les dades sobre la sessió (exceptuant el fitxer amb les lectures) .

	3. Es crida a un mètode del Sistema Predictiu per crear una nova sessió passant com a paràmetre la llista L amb les dades de la sessió i l'identificador ID. 4. Es crida a un mètode del Sistema Predictiu per afegir noves sessions a partir de lectures passant com a paràmetres la ruta del fitxer amb les lectures. 5. S'envia missatge de confirmació al client en el qual es proporciona l'identificador de la nova sessió creada.
Postcondició	El client ha rebut un missatge amb l'identificador de la nova sessió. La nova sessió existeix en el sistema predictiu.

Fitxa de cas d'ús – Mostrar Informació Sessió

Descripció	Mostrar Informació Sessió
Actor	Usuari amb navegador web
Precondició	El client ha rebut un missatge amb l'identificador d'una sessió
Flux Principal	1. El client envia automàticament una petició al servidor sobre la informació de la sessió identificada per l'ID rebut en el missatge inicial. 2. El servidor rep la petició de informació amb l'identificador de la sessió. 3. S'obtenen les sessions anteriors sAnt a la sessió ID. 4. Es crida a un mètode del Sistema Predictiu per dibuixar les sessions sAnt i ID. 5. Es construeix una pàgina HTML utilitzant els dibuixos obtinguts anteriorment i s'envia al client. També s'envia l'identificador de la sessió 6. El client mostra la pàgina web rebuda des del servidor.

Postcondició	El client ha rebut i mostrat una pàgina web amb la informació de la sessió identificada per el ID obtingut del missatge rebut inicialment.

Fitxa de cas d'ús – Realitzar Predicció

Descripció	Realitzar Predicció
Actor	Usuari amb navegador web
Precondició	El client es troba en la pàgina HTML en que es mostra la informació de la sessió.
Flux Principal	1. L'usuari prem el botó Continuar i el client envia una petició al servidor per realitzar la predicció sobre la sessió S identificada amb ID. El client espera resposta. 2. El servidor rep la petició de predicció sobre la sessió S amb identificador ID. 3. El servidor crida a un mètode del Sistema Predictiu per realitzar la predicció passant com a paràmetre l'identificador ID. 4. El servidor crida a un mètode del Sistema Predictiu per dibuixar una la seqüència a la que pertany la sessió S. 5. El servidor crida a un mètode del Sistema Predictiu per dibuixar el vocabulari generat durant la predicció. 6. El servidor construeix i envia al client una pàgina HTML amb els dibuixos de la seqüència i el vocabulari. 7. El client mostra la pàgina HTML. 8. L'usuari prem el botó "Continuar" i s'envia al servidor la petició de mostrar el resultat de la predicció. El client espera resposta. 9. El servidor construeix i envia la pàgina HTML en que es mostra el resultat de la predicció.

	10. El client rep la pàgina HTML i la mostra.
Postcondició	El client ha rebut una i mostrat una pàgina web amb el resultat de la predicció.

Diagrama de Activitats

A continuació es desenvolupa un diagrama de activitats que mostra el comportament general de l'aplicació client-servidor per realitzar una predicció (Figura 8.2).

El diagrama mostrat engloba les 4 fitxes de casos d'ús explicades en l'apartat anterior i mostra el flux i la relació que hi ha entre elles.

També serveix per observar la interacció entre la part client i la part servidor de l'aplicació així com la interacció del servidor amb el Sistema Predictiu.

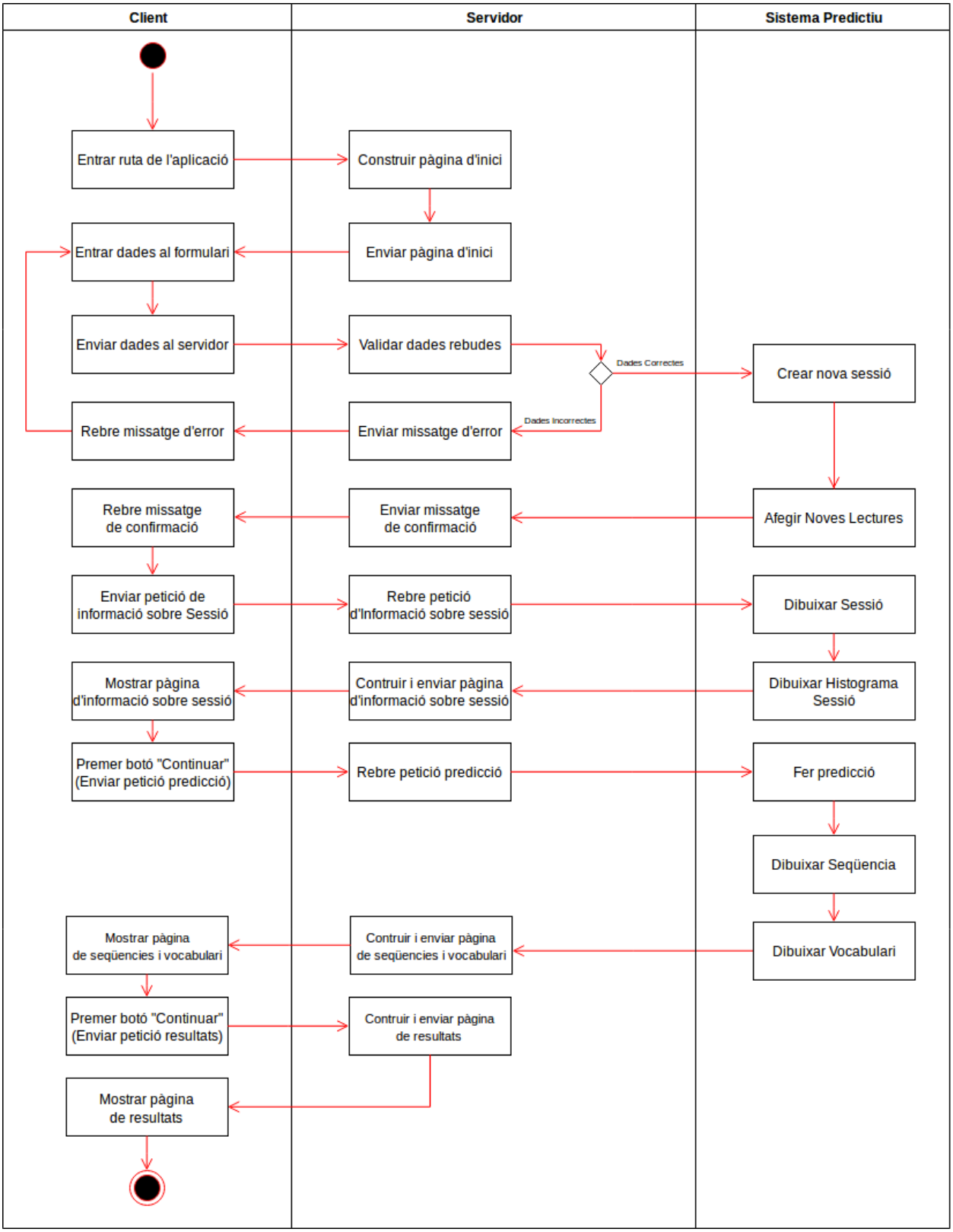


Figura 8.2: Diagrama d'activitats del Servidor

8.1.2 Sistema Predictiu

Diagrama de Cas d'Ús

En aquest apartat es mostrarà un diagrama de casos d'ús general del sistema predictiu (Figura 8.3). En aquest diagrama es mostra el comportament del sistema i les situacions en que es pot trobar. Alguns aspectes a tenir en compte sobre el diagrama:

- En el sistema predictiu l'usuari que interactua amb l'aplicació no és una persona directament, sinó que és l'aplicació client-servidor que realitza peticions al sistema a través de crides a mètodes i funcions.
- A part de la funcionalitat principal de realitzar prediccions, la resta de funcionalitats s'han determinat segons els requisits que s'han trobat al realitzar la aplicació servidor i la majoria tenen a veure amb la obtenció i visualització de dades.

Figura 8.3: Diagrama de cas d'ús del sistema predictiu

Fitxes de Cas d'Ús

En aquest apartat es desenvolupa la fitxa de cas d'ús en que es descriu el flux principal que es segueix en el sistema predictiu per a realitzar una predicció. Al tractar-se d'una funcionalitat molt complexa i que implica molts passos s'ha decidit dividir en 3 fitxes més petites:

- Crear Sessió: introducció de dades sobre una nova sessió. Es vol predir el que es produirà després de aquesta nova sessió.
- Generació dels models: es genera el vocabulari a partir de les sessions del sistema i es construeixen les seqüències.
- Realització de les prediccions: a partir de les seqüències construïdes es prediu el que es produirà després de la nova sessió introduïda.

Fitxa de cas d'ús – Crear Sessió

Descripció	Crear Sessió
Actor	Aplicació servidor
Precondició	El paràmetre d'entrada és una llista de tuples. Cada tupla conté totes les dades necessàries per a crear una sessió (una de les dades és el fitxer de lectures). Aquestes dades han estat validades.
Flux Principal	1. Per cada tupla T de la llista fer: a. Crear nou registre a la taula Sessions de la base de dades amb la informació de la tupla T. Aquest registre conté la següent informació: • ID • ID Pacient • Insulina • Carbohidrats • Exercici Before • Exercici After

	• Alcohol • Glucose • Tipus Àpat (esmorzar/dinar/sopar) • Label (indicant segons les lectures de la sessió si s'ha produït hiperglucèmia, hipoglucèmia o res) b. Per cada línia del fitxer de lectures de la tupla T, crear nou registre a la taula de Lectures. Aquest nou registre conté la següent informació: • Valor de la lectura • Timestamp • ID Sessió 2. Aplicar mètodes de "neteja" de dades al les lectures. Aquests mètodes creen registres a la taula CleanLectures. 3. Per cada registre nou a la taula Sessions fer: a. Calcular histograma de lectures utilitzant la taula CleanLectures. b. Normalitzar histograma (donar valors relatius entre 0 i 1). c. Crear registre a la taula SessionsFDP amb les dades del histograma
Postcondició	S'han actualitzat les taules Sessions, Lectures, CleanLectures i SessionsFDP a partir de les dades entrades.

Fitxa de cas d'ús – Generació dels models

Descripció	Generació dels models
Actor	Aplicació servidor

Precondició	El paràmetre d'entrada és un diccionari que conté el número de paraules del vocabulari (K) i la llargada de les seqüències (NS).
Flux Principal	1. Obtenir tots els registres de la taula SessionsFDP 2. Generar vocabulari a partir de les SessionsFDP 3. Crear taula Words amb la informació del clusters (o paraules) generats 4. Assignar una paraula a cada registre de la taula Sessions 5. Per cada sessió S de la taula Sessions: a. Obtenir NS-1 sessions anteriors a S del mateix pacient b. Obtenir label nL de la sessió immediatament següent a S del mateix pacient c. Crear registre a la taula Sequences amb els ids de les sessions anteriors, el id de la sessió S i el label nL
Postcondició	S'han creat les taules Words i Sequences. S'ha actualitzat la taula Sessions.

Fitxa de cas d'ús – Predir Sessions

Descripció	Predir Sessions
Actor	Aplicació servidor
Precondició	El paràmetre d'entrada és una llista amb els ids de les sessions a predir (sp_ids). Ja s'han generat el vocabulari i les sessions.
Flux Principal	1. Obtenir ids de totes les sessions (train_ids) que no estiguin a la llista de sessions a predir sp_ids. 2. Obtenir models de les sessions train_ids. Un model d'una sessió està format per la informació de les sessions que

	formen part de la seqüència a la que pertany. La informació d'una sessió està composta per les dades de la sessió (carbohidrats, insulina, etc.) i la paraula amb que s'ha etiquetat. 3. Obtenir labels de les seqüències de les sessions train_ids. 4. Construir objectes per predir (KNN o SVM) amb els models i els labels de train_ids. 5. Fer prediccions de les sessions sp_ids. 6. Processar la llista de prediccions retornada i assignar 1, 0 o -1 segons es tracti de hiperglucèmia, normal o hipoglucèmia respectivament.
Postcondició	Es retorna una llista amb la predicció de cadascuna de les sessions entrades.

Diagrama de Activitats

A continuació es desenvolupa un diagrama de activitats que mostra el comportament general del sistema predictiu per realitzar una predicció (Figura 8.4).

El diagrama mostrat engloba les 3 fitxes de casos d'ús explicades en l'apartat anterior i mostra el flux i la relació que hi ha entre elles.

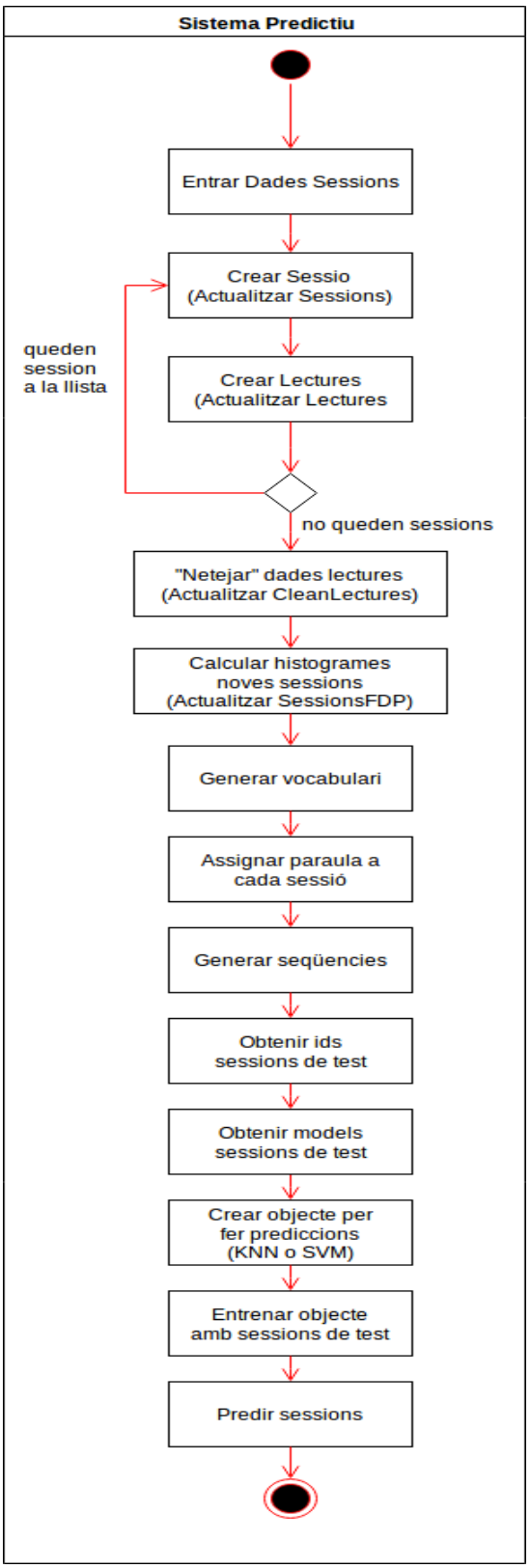


Figura 8.4: Diagrama d'activitats sistema predictiu

8.2 Disseny

8.2.1 Aplicació Client-Servidor

Per l'aplicació client-servidor no s'han realitzat diagrama de classes ja que te una estructura molt senzilla: una classe Servidor relacionada 1 a 1 amb una classe CGMPredicter (classe "pare" que implementa el sistema predictiu).

Aquesta senzillesa s'aconsegueix gràcies a la utilització del mòdul de python Flask que emmascara tot el necessari per a la creació i execució d'una aplicació servidor deixant al programador l'única tasca de dissenyar i implementar una API. Per la mateixa raó tampoc s'han desenvolupat diagrames de seqüències.

En resum, el treball de disseny de l'aplicació client-servidor es troba en el disseny de la API que utilitzaran client i servidor per comunicar-se i finalment en el disseny de les interfícies.

API REST

Una API REST és una interfície entre sistemes que utilitza HTTP per obtenir dades o generar operacions sobre aquestes dades. Una API REST te les següents característiques:

- Protocol client/servidor sense estat: en cada missatge HTTP hi ha tota la informació necessària per comprendre l'estat. Per tant ni el client ni el servidor han de recordar cap estat de les comunicacions anteriors.
- Conjunt de operacions ben definides: es defineixen 4 operacions: POST (crear), GET (obtenir), PUT (actualitzar) i DELETE (eliminar) que es podran aplicar a tots els recursos de l'aplicació.
- Sintaxi universal: cada recurs és accessible per una única URI (identificador de recursos universal).
- Ús de hipervincles: la representació d'un estat en un sistema REST és típicament un HTML. Es possible i recomanable poder navegar d'un recurs a molts altres a través senzillament de enllaços sense necessitar operacions addicionals (com registres).

Al desenvolupar una API REST per aquest projecte s'han reflexat aquestes característiques en el disseny.

Hi ha dos recursos als que es poden aplicar les operacions GET, POST, PUT i DELETE: les sessions i els objectes de predicció (predicter). Addicionalment també s'han definit operacions per a pujar fitxers al servidor.
A continuació es mostra i descriu cadascuna de les operacions que es poden realitzar fent peticions a la API REST:

- GET sessió
    - URI: /session/<id>
    - Dades petició: id de la sessió de la que es vol obtenir informació.
    - Descripció: retorna una pàgina HTML amb una gràfica mostrant informació sobre les lectures de la sessió i una gràfica mostrant l'histograma de les lectures de la sessió utilitzades. En aquesta pàgina hi ha un enllaç per crear un predicter.
- POST sessió
    - URI: /session/
    - Dades petició: totes aquelles necessàries per crear una nova sessió: nom del fitxer amb les lectures de la sessió, pacient, data, insulina, carbohidrats, exercici abans, etc.
    - Descripció: crea una nova sessió en el sistema predictiu. Si tot funciona correctament s'envia un missatge de confirmació en forma de resposta HTTP codi 200. Altrament s'envia un missatge d'error amb codi 500.
- GET predicter
    - URI: /predicter/<id>
    - Dades petició: id del objecte predicter utilitzat per predir la sessió amb el mateix id.
    - Descripció: es realitza la predicció de la sessió amb identificador <id> i es retorna una pàgina HTML mostrant aquesta informació.

- POST predicter
    - URI: /predictor/
    - Dades petició: id de la sessió que es vol predir.
    - Descripció: es crea i guarda un objecte per fer prediccions. Es retorna una pàgina HTML que inclou unes gràfiques amb les seqüències i el vocabulari generats amb l'objecte predicter creat. En aquesta pàgina HTML hi ha un enllaç per fer una petició GET predicter.
- POST file
    - URI: /file/
    - Dades petició: fitxer a pujar al servidor
    - Descripció: es puja i guarda un fitxer de lectures al servidor. Si tot funciona correctament s'envia un missatge de confirmació en forma de resposta HTTP codi 200. Altrament s'envia un missatge d'error amb codi 500.

Mitjançant la API REST que s'acaba de definir, el sistema és capaç de realitzar totes les operacions que s'han descrit en l'apartat d'anàlisi.

Interfícies d'Usuari

En aquest apartat es mostrarà el disseny de les diferents interfícies que l'usuari utilitzarà per interaccionar amb el servidor. Junt amb cada imatge s'explica els components de la interfície i les opcions que té l'usuari. Algunes consideracions sobre les imatges de les interfícies que es mostren a continuació:

- Les imatges mostrades han estat fetes en un ordinador amb una resolució de 1920x1080. Tot i això la mida de la pantalla no és important ja que gràcies a la utilització de bootstrap durant el desenvolupament, el contingut s'adapta a qualsevol mida de pantalla, fins i tot a pantalles de mòbil.
- El navegador utilitzat ha estat Mozilla Firefox, però també s'ha provat amb Opera i Chrome.

Home

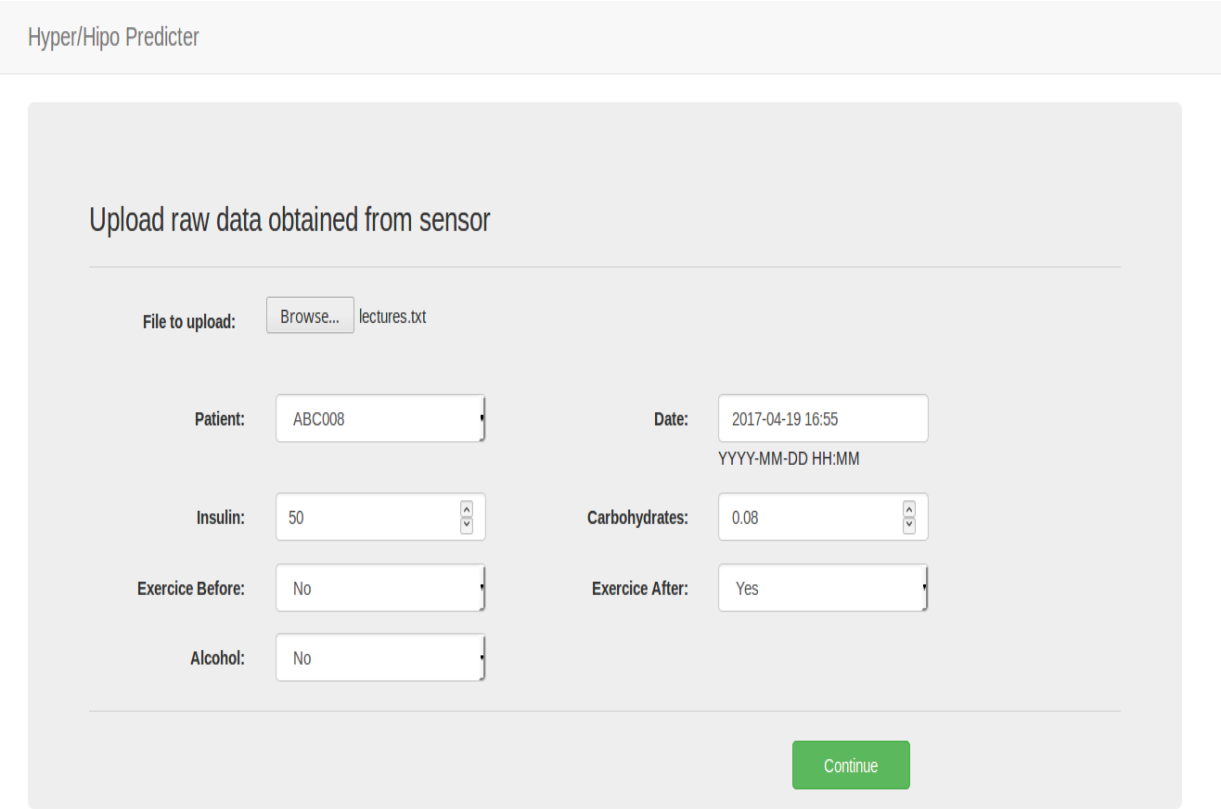


Figura 8.5: Pantalla Home

Es tracte de la pantalla de inici, la pantalla que veu l'usuari al connectar-se al servidor introduint la URL del servidor en el seu navegador.

En aquesta pantalla s'introdueixen totes les dades de la nova sessió així com el fitxer que conté les lectures de glucosa fetes pel sensor. Els diferents camps del formulari estan inicialitzats amb valors per defecte, per exemple: la data correspon a la data i hora en que s'ha entrat a la pàgina.

Un cop introduïdes les dades l'usuari ha de prémer el botó "Continuar". Al prémer el boto s'envien totes les dades al servidor i aquest realitza les tasques per introduir les dades de la nova sessió en el sistema (Fitxa de cas d'ús – Enviar Dades al Servidor).

La primera d'aquestes tasques és la validació de les dades: si les dades no son correctes es mostra un missatge indicant l'error al usuari:

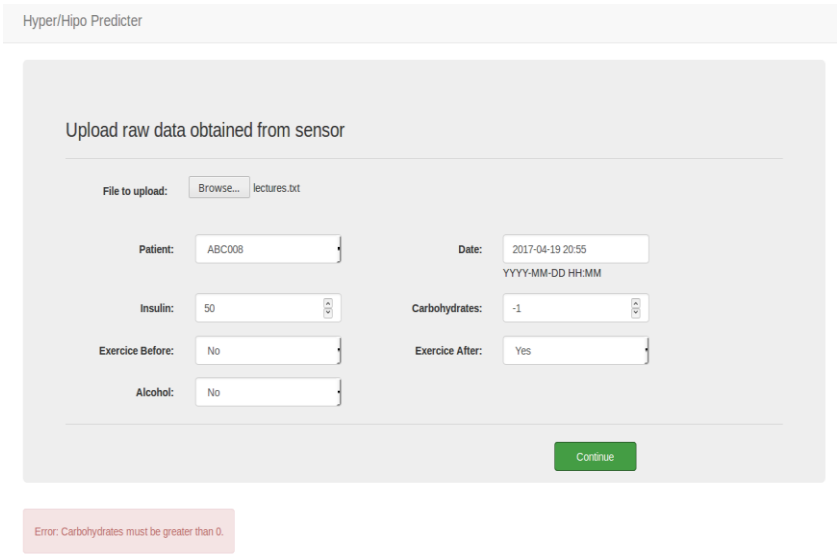


Figura 8.6: Pantalla Home amb missatge de "Error"

Si la validació de dades és correcte el servidor continua amb la resta de tasques per crear la nova sessió (Fitxa de cas d'ús – Crear Sessió). Es mostra un missatge al usuari indicant que el servidor està treballant:

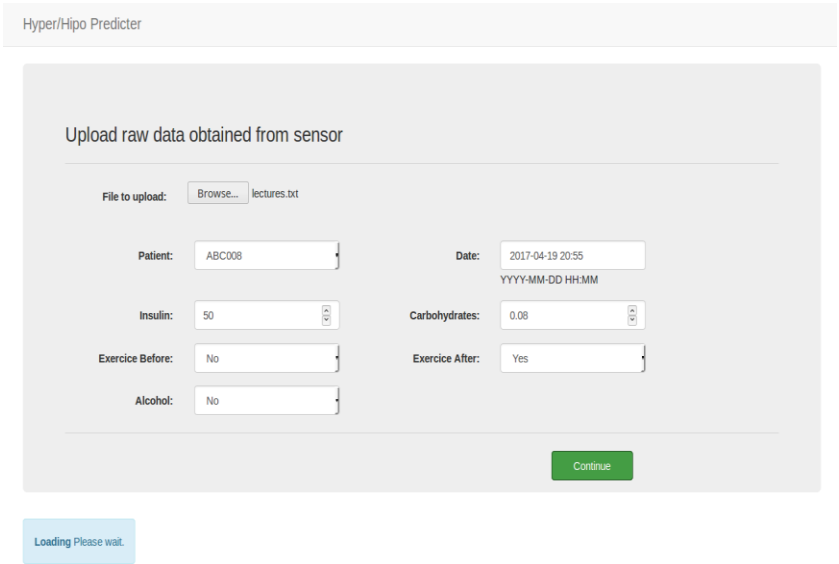


Figura 8.7: Pantalla Home amb missatge "Loading"

Data View

Figura 8.8: Pantalla Data View

Un cop el servidor ha carregat les dades de la nova sessió al sistema predictiu, s'envia al client la pantalla de "Data View" (Fitxa de cas d'ús – Mostrar Informació Sessió). En aquesta pantalla es mostra al usuari les dades que ha carregat amb el fitxer de lectures.

El gràfic correspon a la corba de glucosa del pacient. Es mostra en color blau la totalitat de les dades entrades i en color groc/taronja les dades que s'utilitzen (es treballa només amb les dades dins una determinada franja de temps ja que son les més rellevants).

També es mostren els límits inferior i superior perquè es consideri que s'ha produït en el pacient una hipoglucèmia o una hiperglucèmia respectivament. Els punts de color verd indiquen el moment en que s'ha ingerit carbohidrats i insulina, es a dir, el moment del àpat.

Finalment, prement el botó "Continuar" es demana al servidor que realitzi la predicció (Fitxa de cas d'ús – Realitzar Predicció) i es passa a la següent pàgina (Fitxa de cas d'ús – Mostrar Predicció).

Seqüence View

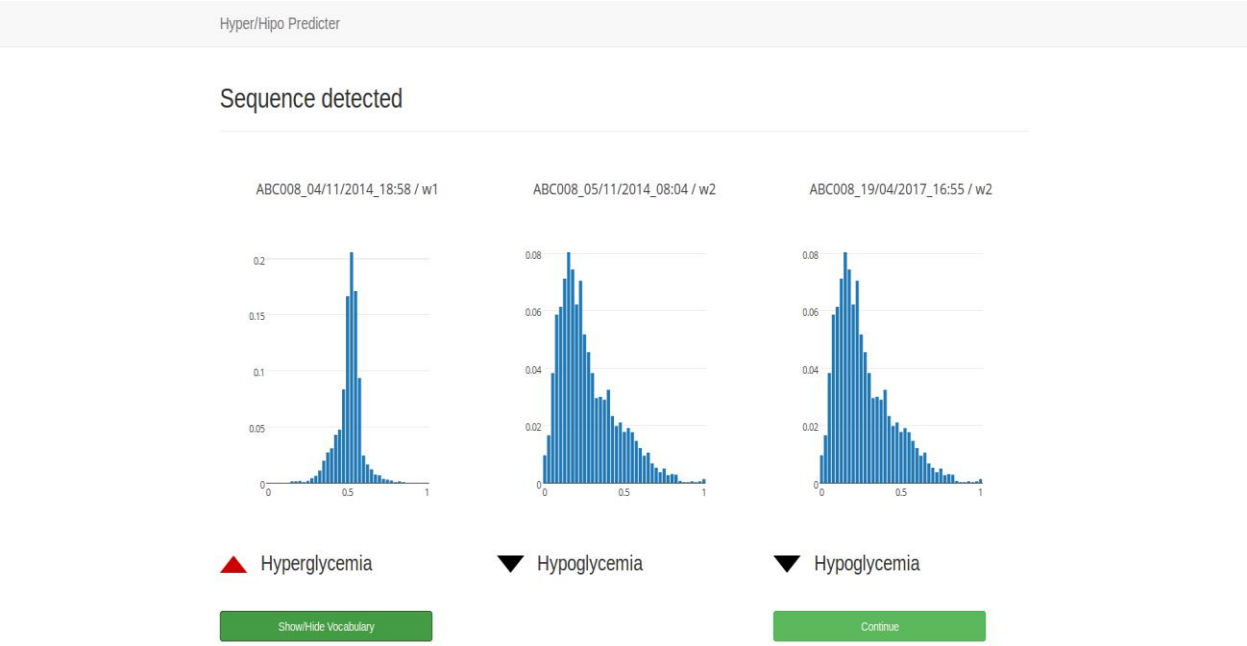


Figura 8.9: Pantalla Sequence View

En aquesta pantalla es mostra informació sobre la seqüència a la que pertany la nova sessió que l'usuari ha introduït. En aquest punt el servidor ja ha generat tot el vocabulari i les seqüències de sessions.

Es mostra un gràfic per cada sessió que forma part de la seqüència a més d'una fletxa sota de cada gràfic indicant si en la sessió en qüestió es va produir hiperglucèmia, hipoglucèmia o cap

de les dos. En aquest cas els gràfics no son histogrames de les lectures de les sessions, sinó que corresponen a la informació sobre la paraula amb que es va etiqueta cada sessió.

Es pot veure com a sobre de cada gràfic hi ha escrit el nom de la sessió a la que correspon i tot seguit el nom de la paraula amb que es va etiquetar aquesta sessió. Els noms de les paraules del vocabulari tenen la següent estructura: w<tipus>_<número>, on "tipus" pot ser 1/0/-1 segons es tracti de una paraula que representi una hiperglucèmia, normoglucèmia o hipoglucèmia respectivament; i "número" és un comptador de 1 a N paraules (N és la mida del vocabulari generat).

A sota de la informació de la seqüència hi ha dos botons que fan el següent:

- Show/Hide Vocabulari: es mostra o amaga el vocabulari que s'ha generat. Aquest vocabulari es mostra al final de la pàgina (Fig. 8.10).
- Continuar: es passa a la pàgina final en que es mostra el resultat de la predicció

Figura 8.10: Pantalla Sequence View amb Vocabulari

Results View

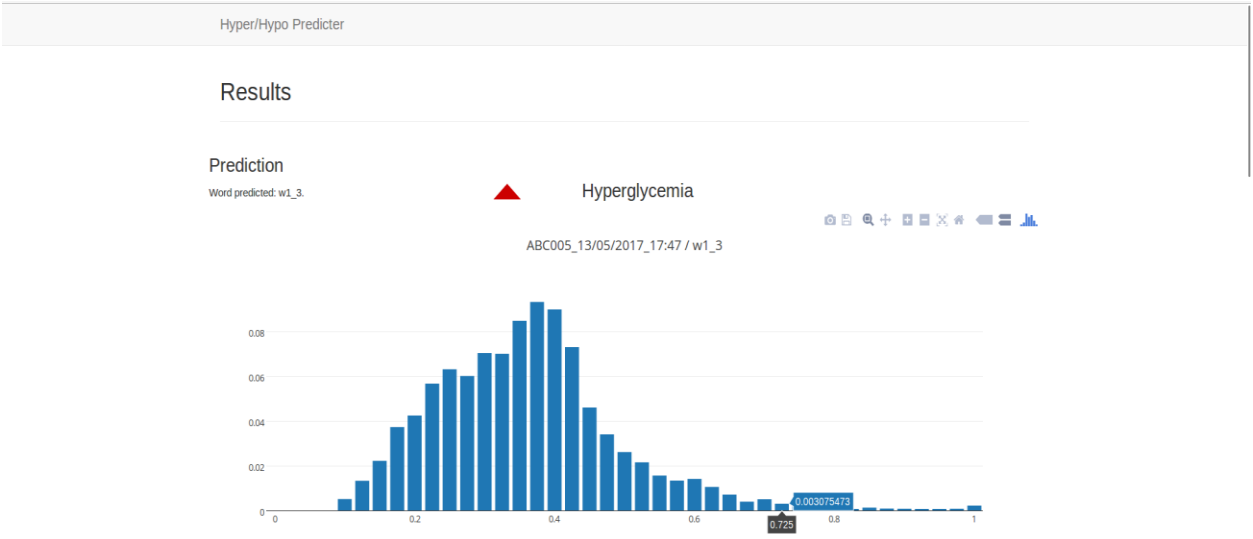


Figura 8.11: Pantalla Final: sessió predita

En aquesta pantalla es mostra tota la informació sobre els resultats obtinguts. El primer que es mostra és la paraula que s'ha predit per a la sessió i el seu tipus: hiperglucèmia, normal o hipoglucèmia.

A continuació es mostra la seqüència completa de paraules utilitzada en les prediccions.

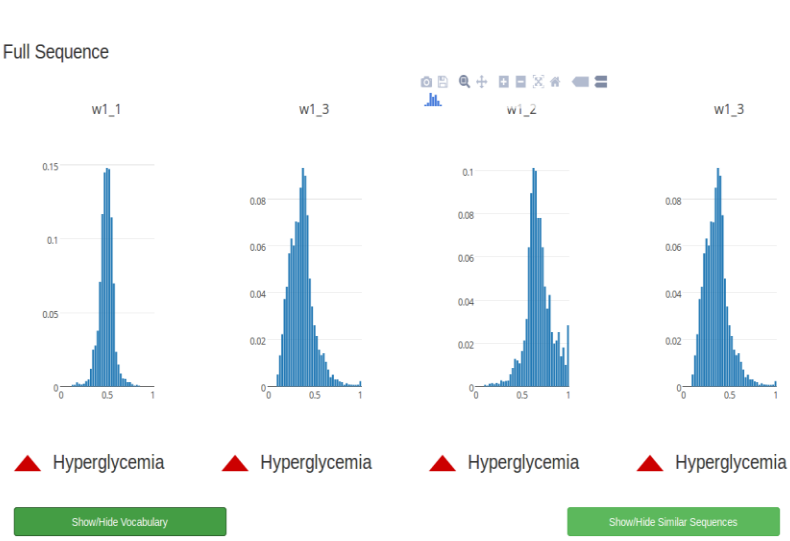


Figura 8.12: Pantalla Final: seqüencia completa

Després de la seqüència de paraules hi ha dos botons: el primer (show/hide vocabulary) fa el mateix que en la pantalla anterior, mostra i oculta el vocabulari generat amb el que s'ha treballat. El segon (show/hide similar sequences) mostra i oculta les seqüències de sessions més similars a la seqüència predita.

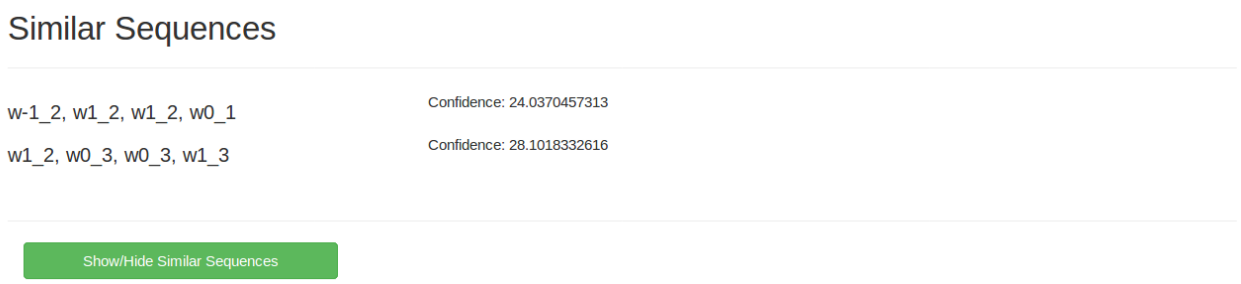


Figura 8.13: Pantalla Final: seqüencies similars

8.2.2 Sistema Predictiu

El treball en el disseny del sistema predictiu consta de la implementació del diagrama de classes i l'estructura de la base de dades. També s'explica i descriu els patrons de disseny utilitzats.

En aquest cas no hi ha disseny d'interfícies ja que es tracte d'una aplicació que no està pensada per ser utilitzada per usuaris directament, sinó que està dissenyada per ser integrada i utilitzada per altres aplicacions (com en aquest projecte fa l'aplicació client-servidor).

Diagrama de Classes i Base de Dades

En aquest apartat es mostra el diagrama de classes del sistema predictiu (Fig 8.14).

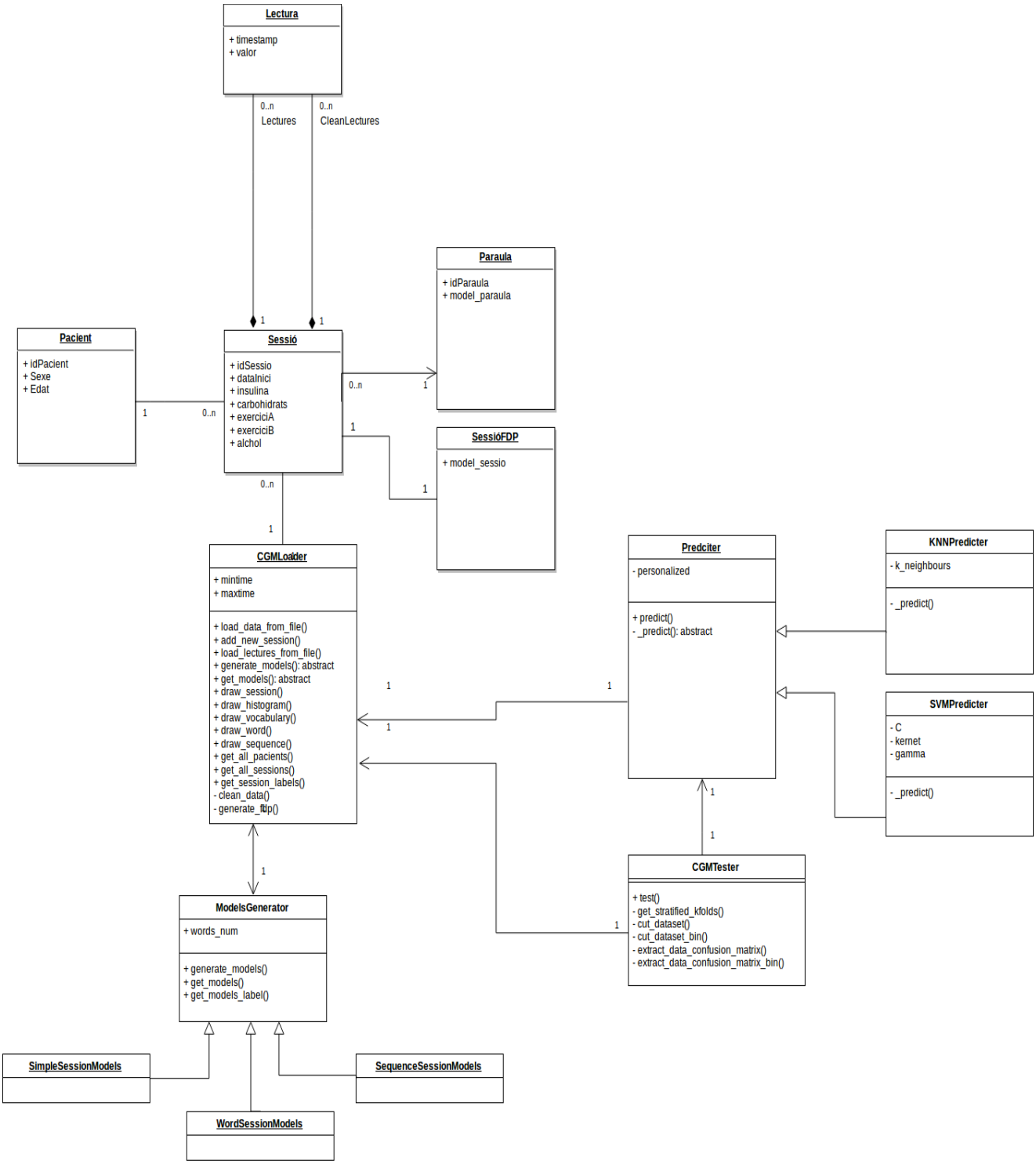


Figura 8.14: Diagrama Classes del sistema predictiu

Analitzant el diagrama de classes s'observa el següent:

- Hi ha tres classes principals:
    - CGMLoader: és l'única que interacciona amb la base de dades. S'encarrega de crear, tractar i proporcionar totes les dades.
    - ModelGenerator: s'encarrega de generar i proporcionar els models (dades en un format determinat) que s'utilitzaran per fer prediccions. La classe CGMLoader utilitza aquesta classe per proporcionar dades dels models al CGMPredicter.
    - CGMPredicter: encarregada de realitzar les prediccions. Utilitza la classe CGMLoader per obtenir les dades necessàries.
- La resta de classes només s'utilitzen per encapsular la informació, no realitzen cap tipus de lògica. Per aquesta raó son les persistents en la base de dades.
- Per claredat, només s'ha inclòs 3 "ModelsGenerator" de tots els implementats.

En l'apartat 9 d'Implementació i Proves es farà una descripció més acurada de les funcions i mètodes de cadascuna de les classes més rellevants.

No totes les classes mostrades en el diagrama es guarden a la base de dades. Les classes que tenen persistència i són guardades en una base de dades son les referents a les dades dels pacient. En la Figura 8.15 es mostra l'estructura de la base de dades.

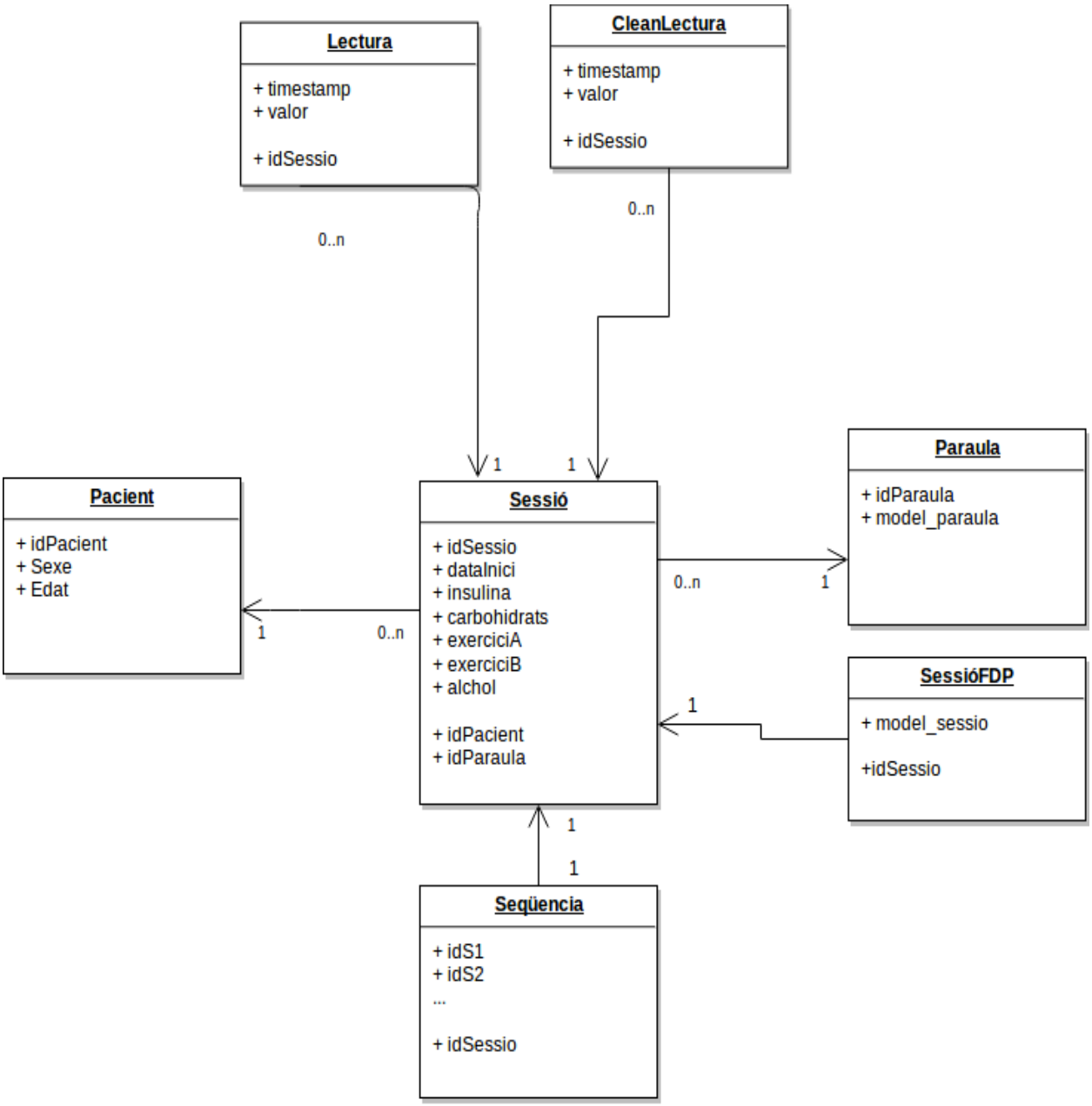


Figura 8.15: Estructura de la Base de Dades

Patrons

El principal problema que s'ha trobat durant el disseny del sistema predictiu ha estat trobar una manera flexible de poder generar i obtenir els models utilitzats per realitzar les prediccions.

Tal i com s'ha explicat en la fitxa de cas d'us "Predir Sessions", per realitzar una predicció es necessita obtenir els models de les sessions. Un model s'entén com una vector de atributs que representen una sessió. Aquests models s'utilitzen per entrenar el sistema i per realitzar les posteriors prediccions.

La raó per la qual es necessitava una manera flexible de poder generar i obtenir aquests models és que trobar la millor manera de representar les sessions és un procés de recerca llarg que implica realitzar moltes representacions diferents, algunes d'elles combinant varies representacions, fins a trobar la que produeix més bons resultats.

Per exemple:
Una representació base d'un model d'una sessió seria l'histograma de lectures.
Una segona representació seria els atributs d'una sessió (carbohidrats, insulina, etc.). Finalment una tercera seria un histograma junt amb els atributs, és a dir, una combinació de les dos anteriors.

Existeix un patró de disseny que compleix perfectament els requisits de la situació descrita: el patró *decorator*. A continuació s'explicaran les característiques d'aquest patró i tot seguit s'exposarà la seva aplicació en el disseny del sistema predictiu.

Patró Decorator

Aquest patró proporciona una forma dinàmica de afegir comportaments a una funcionalitat d'un objecte sense haver de utilitzar l'herència d'objectes.

El seu funcionament és el següent:

- Es defineix un objecte Component que proporciona una interfície per al mètode A.
- Es defineix objecte Decorator que conté un objecte Component i proporciona una interfície igual a la del objecte Component per al mètode A. Quan es crida al mètode Decorator.A, aquest el que fa és invocar el mètode Component.A del objecte Component que el forma.
- Es defineixen varis objectes Decorator que criden el mètode A i després afegeixen la seva pròpia implementació.
- Es defineix un objecte ConcretComponent que implementa el mètode A.

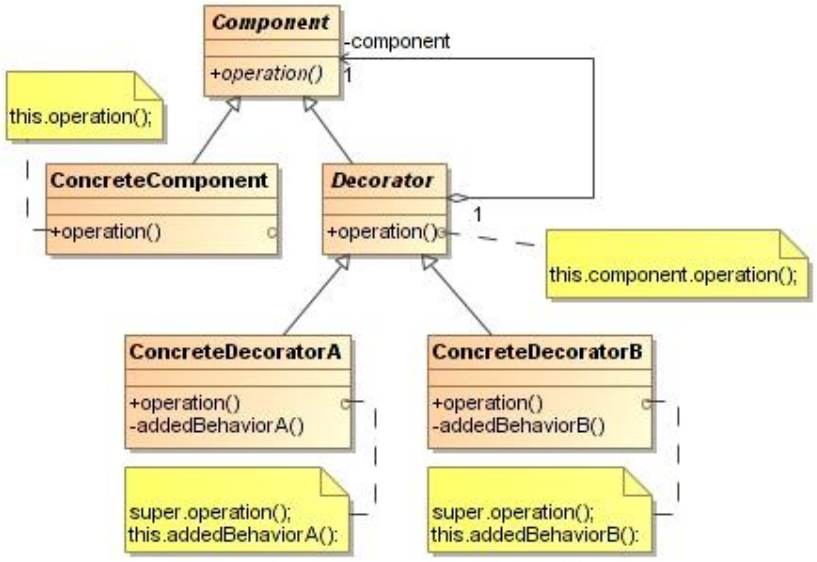


Figura 8.16: Estructura Patró Decorator

Amb aquesta estructura (mostrada en la figura 8.16) es crea un objecte ConcretDecorator i llavors es van creant objectes Decorator al que se'ls passa com a paràmetre l'objecte al que se l'hi ha de estendre la funcionalitat. Al primer Decorator d1 se l'hi passarà el ConcretComponent, però al següent Decorator d2 se l'hi passarà el Decorator d1. D'aquesta manera es poden anar afegint implementacions successivament.

Aplicació Patró Decorator

Tal i com ja s'ha explicat al inici d'aquest apartat, el patró decorator s'ha utilitzat per poder canviar dinàmicament el comportament dels mètodes generar i obtenir models de forma dinàmica.

Així, es consta d'una classe base: la classe Loader, que defineix els mètodes:

- get_models()
- generate_models()
- get_models_labels()

En la classe Loader aquests mètodes no realitzen cap tasca, només defineixen la interfície dels mètodes. Es tracte de la classe Component del patró.

A la classe Loader se l'hi assigna un objecte de la classe ModelGenerator. Sempre que es crida un dels 3 mètodes anteriors, el Loader es limitarà a cridar el mètode amb nom homònim de la classe ModelGenerator. La classe ModelGenerator és la classe Decorator.

Un objecte de la classe ModelGenerator té com a atribut un objecte ModelGenerator, que serà el "decorator fill". Quan es crida un mètode del ModelGenerator el primer que es fa és cridar el mateix mètode per l'objecte "fill" i es llavors es treballa a partir dels resultats que aquest retorni.

Un exemple concret de classes "Decorator" que s'han implementat son les següents:

- SimpleModels:
    - generate_models(): no realitza cap tasca addicional.
    - get_models(): retorna la FDP de les sessions.
- WordsModels:
    - generate_models(): genera el vocabulari a partir dels models obtinguts amb el mètode get_models() del seu objecte fill.

    - get_models(): afegeix a cada model la informació corresponent a la paraula amb que ha estat etiquetada la sessió d'aquell model.
- SequenceModels:
    - generate_models(): genera les seqüències a partir dels models obtinguts amb el mètode get_models() del seu objecte fill.
    - get_models(): afegeix a cada model la informació corresponent la resta de sessions de la seqüència.

En l'apartat 9.2.2 s'explica amb detall tots els generadors desenvolupats i el seu funcionament.

9 IMPLEMENTACIÓ I PROVES

En aquest apartat es descriu amb detall la implementació de cadascun dels dos sistemes desenvolupats en el present P/TFC així com de les classes (o components) que els formen.
També s'exposen els problemes trobats durant la seva implementació així com les solucions proposades.

Finalment, en l'apartat de proves, al trobar-nos en un projecte més enfocat a la recerca que al desenvolupament d'un producte, es descriuen els mètodes utilitzat per a mesurar la qualitat del sistema predictiu en comptes de dedicar-lo al testeig de mètodes i components del sistema. És a dir, que s'exposen les mètriques, tècniques i mètodes utilitzats per a mesurar la qualitat de les prediccions que realitza el sistema predictiu així com les classes implementades per a la realització d'aquestes proves.

9.1 Implementació Aplicació Client-Servidor

L'aplicació client-servidor ha estat implementada utilitzant python3.5 i la llibreria del framework Flask. Les raons de la utilització de Flask així com les seves principals característiques ja han estat exposades en l'apartat 7.1.7.

En aquesta secció s'explicarà la implementació concreta d'aquesta aplicació: classes, estructura del projecte, mètodes, etc.

9.1.1 Estructura del Projecte

Una aplicació desenvolupada en Flask, a diferencia que amb altres frameworks com per exemple Django, no te una estructura predefinida ja que, degut a la senzillesa en la implementació que aporta es podria fer un servidor en un sol fitxer amb poques línies de codi. Per tant l'estructura la defineix el propi desenvolupador segons les seves necessitats.

En aquest cas les necessitats van ser:

- Pujar fitxers al servidor
- Tenir plantilles per a les respostes

Seguint aquestes necessitats l'estructura del projecte final es va definir de la següent manera:

- /WebCGM.py: fitxer de python amb el codi del servidor.
- /Uploads/ : directori on es guarden els fitxers pujats al servidor.
- /Templates/: directori on es guarden les plantilles (desenvolupades utilitzant Jinja2) que retorna el servidor com a resposta. En aquestes plantilles s'implementa la poca lògica que es deixa per al client.
- /static/: directori amb els que componen el framework Bootsrap utilitzat en el disseny de les plantilles.
    - /css/: fitxers de estil.
    - /img/: imatges i icones.
    - /js/: fitxers amb funcions de javascript i jQuery.

9.1.2 WebCGM: el Servidor

En aquest fitxer hi ha tot el codi referent al servidor de l'aplicació. Consta de un mètode "main" al que se l'hi passen tots els paràmetres de configuració del sistema (a través de línia de comandes) i de una classe: CGMServer.

La classe CGMServer és l'encarregada de crear el servidor utilitzant el framework Flask.
Cada crida a l'api s'implementa amb un mètode de la classe amb una sintaxi especial que defineix Flask:

```
@app.route(path, methods=[tipus_peticions])
def metode():
    # code
    return resposta
```

En la primera línia (*@app.route*) es on es defineix l'api. Te dos paràmetres que defineixen el següent:

- Ruta: defineix una nova petició a l'api amb la ruta especificada, per exemple "/user/<id>".
- Method: descriu el tipus de petició al que s'ha de respondre. Es tracta d'una llista que pot contenir: "GET", "PUT" i "POST". Si es fa una petició a la ruta definida però el tipus de petició no es troba en la llista, no es respon.

Amb aquesta sintaxi es construeix l'api amb l'estructura explicada en l'apartat 8.2.1.

Cadascuna de les rutes va acompanyada del seu mètode en python en que es realitza la lògica necessària segons la petició. El codi complet de cada mètode es pot consultar en l'annex 14.3.

A més de la definició de l'api a través dels mètodes tal i com s'ha descrit, en el codi hi ha els següents elements que serveixen per configurar i finalment iniciar el servidor:

```
# Configuration options
UPLOAD_FOLDER = './uploads'
ALLOWED_EXTENSIONS = set(['csv'])

# Prepare the server
app = Flask(__name__, static_folder="static", template_folder="templates")
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Inizialite Loader, connect to existing database
loader = Loader(new_db=False, resources_path=app.config['UPLOAD_FOLDER']+"/")
predicter_dict = {}

# Start the server
app.run(host="127.0.0.0", port="18069")
```

En la primera línia estem definint el directori en que es guarden els fitxers pujats al servidor, en la segona indiquem que només poden ser fitxers de tipus .csv.

En el següent bloc es en el que es crea l'objecte flask que serà el nostre servidor. Se l'hi dona un nom i se l'hi indica els directoris dels templates i dels fitxers css, js, etc. com es pot observar es correspon amb l'estructura explicada anteriorment.

En el tercer bloc s'inicia el Loader, la classe del sistema predictiu que ofereix la interfície (a través de mètodes) per poder realitzar prediccions a més de estar connectada amb la base de dades i proporcionar dades. També és l'encarregada, per exemple, de construir els gràfics que es mostren en els templates enviats al client.
La següent línia inicialitza un diccionari, en aquesta estructura de dades es guarden i encuen les peticions de prediccions rebudes per els clients per poder contestar-les totes.

Finalment en la última línia s'inicia el servidor en la direcció i port indicats.

Amb aquestes línies de codi es posa el servidor a funcionar esperant peticions a l'api que es defineixi. Com es pot observar, gràcies a Flask es tracta d'una tasca relativament ràpida, deixant tota la feina en el disseny i implementació de l'api.

Aquest fet va solucionar un dels problemes sorgits durant la implementació de l'aplicació: els canvis en els requeriments per part del client.

Durant el desenvolupament van sorgir noves necessitats per part del client: per exemple, inicialment no s'havia plantejat la necessitat de mostrar la pantalla sobre la informació de les lectures entrades (Figura 8.8) . Això no va suposar un problema gràcies a ka flexibilitat que aporta flask, només es va haver de definir un nou mètode amb una nova crida a l'api que retornes el template indicat.

9.1.2 Templates: el Client

Al dissenyar la aplicació client-servidor, un dels requeriments era que el client fos un simple navegador web, de manera que l'aplicació fos més accessible per a qualsevol persona.

Durant el disseny del servidor però, es va observar la necessitat que el client executes alguna lògica, per exemple, per crear rutes de l'api dinàmicament o per reaccionar de formes diferent davant les respostes del servidor.

Aquest fet va plantejar el problema següent: com es introduir lògica al client, si aquest és un simple navegador i no es pot instal·lar cap tipus de aplicació tercera o extensió?

La solució al problema van ser les plantilles Jinja2 que el propi flask proporciona. Una plantilla Jijinja2 no és més que una pàgina HTML que el servidor renderitza i envia al client com a resposta. La seva principal característica és que conté parts de codi python que al renderitzar-se creen codi html de forma dinàmica. Aquest fet va ser útil per poder crear contingut de forma dinàmica en les respostes enviades al client: per exemple, introduir tants gràfics referents al vocabulari generat com paraules hi haguessin.

Tot i això, aquesta no és la característica interessant per poder resoldre el problema plantejat. El fet més important és que es tracta d'unes plantilles que es generen en el servidor. Al generar-les se'ls van introduir funcions i mètodes en javascript que implementessin la lògica necessària per part del client. Per tant, la lògica del client es va concentrar en un seguit de mètodes dins les plantilles generades per el servidor de manera que el navegador no hagués de fer res excepte executar-les, fet que pot realitzar sense problemes ja que tots els navegadors en l'actualitat són capaços de interpretar el javascript.

Tots els templates així com el codi de les funcions de javascript utilitzats en l'aplicació client-servidor es poden trobar en l'annex 14.3.

9.2 Implementació Sistema Predictiu

La totalitat del sistema predictiu ha estat implementada utilitzant python3.5 i un seguit de llibreries ja explicades en l'apartat 7 (skit-learn, sqlitle, pandes, etc.).

Es tracte d'un projecte en python que s'ha dividit en 3 fitxers:

- CGMLoader.py: conté la classe Loader i les classes ModelsGenerator
- CGMPredict.py: conté la classe Predicter i totes les classes que hereten d'ella.
- CGMTest.py: conté la classe CGMTester utilitzada per provar el sistema.

El primer que es pot observar és que no hi ha com a classes elements com "Sessió", "Pacient", "Lectura", etc. que es veien en el diagrama de classes de la figura 8.14. Inicialment si que existien i així es va reflexar en el diagrama de classes, però durant el desenvolupament es va acabar optant per eliminar-les.
Es tractava d'unes classes que tenien com a única finalitat encapsular informació obtinguda de la base de dades i no comptaven amb cap mètode ni funció que executes cap tipus de lògica. El fet de haver de construir objectes d'aquestes classes a partir de la informació extreta de la base de dades alentia molt tots els processos en que hi participaven una gran quantitat de dades (mot comuns en aquest projecte). Per tant, finalment es va optar per treballar directament amb les dades extretes de la base de dades sense intermediaris.

A continuació s'explicarà el funcionament i implementació dels tres principals components del sistema: Loader, ModelsGenerator i Predicter. Al mateix moment s'aniran exposant els problemes trobats i les solucions implementades per a ells.

La classe CGMTester conté tot el necessari per a la realització de proves del sistema i mesurar la seva qualitat. No es tracte d'un component del sistema, sino d'una eina construïda i utilitzada per testejar-lo, per tant s'explicarà en el l'apartat 9.3 de proves.

9.2.1 Loader

Aquesta classe és la encarregada d'interaccionar amb la base de dades i proporcionar dades tan a la classe Predicter (per poder realitzar prediccions) com a l'aplicació client-servidor (per poder mostrar informació com els gràfics sobre les lectures, les paraules, etc.).

A continuació s'explicarà els aspectes més importants sobre el seu funcionament, principalment el relacionat amb la carrega i processament de dades. Per veure el codi complet es pot consultar l'apartat 14.3 del annex.

Cal destacar que aquesta classe s'ha desenvolupat no només pensant en les necessitats de l'aplicació client-servidor, sinó també en les necessitats del CGMTester. Això implica que hi ha mètodes i comportaments implementats estrictament per a la realització de proves i que no són de utilitat per al sistema final.

Creació de la Base de Dades

Al crear-se una instància d'aquesta classe hi ha dos opcions: crear una nova base de dades o utilitzar-ne una d'existent. Això es defineix utilitzant els paràmetres d'entrada del constructor.

Si s'escull crear una nova base de dades s'esborra la antiga (si existia) i se'n crea una nova i tota la seva estructura de taules. La base de dades estarà buida.

Si per altra banda s'escull utilitzar-ne una de existent s'ha de passar la ruta del fitxer corresponent (en sqlite una base de dades es guarda en un sol fitxer de text).

Dades Inicials

Un cop creada la base de dades, el següent pas es carregar-hi les dades. Cal recordar que aquest projecte consisteix en realitzar prediccions basant-se en dades històriques de pacient.

Aquestes dades històriques son les que s'han de carregar a la base de dades i amb les que s'ha treballat durant tot el desenvolupament del projecte.

Es tracte d'unes dades proporcionades per el *Imperial College of London* (subjectes a un conveni de confidencialitat) al grup de recerca eXiT. Contenen la següent informació:

- ID Pacient
- Gènere
- Lowertarget (nivell de glucosa considerat hipoglucèmia)
- Normaltarget (nivell de glucosa considerat normal)
- Uppertarget (nivell de glucosa considerat hiperglucèmia)
- Edat
- Data de inici del àpat
- Hora de inici del àpat
- Glucosa inicial
- Carbohidrats
- Insulina
- Exercici Before
- Exercici After
- Alcohol
- Lectures de glucosa cada 5 minuts

Són unes dades amb que ja s'hi ha treballat en el grup de recerca eXiT, gràcies a això se'm va poder proporcionar aquestes dades amb un preposés de format inicial ja realitzat en el que s'ajuntaven totes en un sol fitxer .csv on cada línia corresponia a una sessió (un àpat) amb tota la informació que s'acaba de descriure i en que ja s'havien eliminat algunes sessions en que, per exemple, no hi havia cap lectura.

Aquesta carrega de dades només s'hauria de realitzar el primer cop que s'inicia el sistema. Des de llavors sempre s'inicia el Loader utilitzant la base de dades ja existent amb les dades introduïdes.

Durant la carrega de dades s'etiqueta cada sessió amb '1', '0' i '-1', que corresponen a hiperglucèmia, normal i hipoglucèmia. Aquest etiquetatge es realitza seguint els següents criteris establerts al inicia del projecte:

- Híperglucemia: si s'observen lectures per sobre del uppertaget del pacient durant 60 minuts.
- Hipoglucemia: si s'observa alguna lectura per sota del uppertaget.
- Normal: si no s'observa cap dels dos casos anteriors.

Com es pot observar, la definició de hiperglucèmia és molt més estricte que la de la hipoglucèmia. Es va definir així perquè les hipoglucèmies son molt més greus que les hiperglucèmies i es volia que en les situacions en que ocorreguessin les dos, es classifiques sempre com a hipoglucèmia.

Preprocés de Dades

En el procés de carrega de dades no només es limita a crear sessions, pacients i lectures a les taules corresponents. Un cop fet això es realitza una sèrie de tasques de "neteja" a les lectures per fer-les més aptes per ser utilitzades per realitzar les prediccions. Les noves lectures resultants d'aquest preposés són les que es guarden en la taula CleanLectures mostrada en el diagrama de la base de dades de la figura 8.15.

Les diferents tasques de "neteja" que es realitzen son les següents:

- Filtratge: consultant amb metges es va determinar que s'havia de treballar amb les lectures de glucosa que es troben en la franja de 1.5 a 6 hores després de l'àpat ja que és en aquestes hores quan es pot observar l'efecte de la insulina.

- “Smooth”: es tracte d’un procés en que es ‘suavitzen’ els valors de les lectures de cada sessió de manera que una lectura comparada amb l’anterior i la següent els canvis siguin més petits.

- Normalitzar: implica canviar l’escala de la distribució de valors de manera que la mitjana observada és 0 i la desviació estàndard és 1. Un valor *i* és normalitzat de la següent manera:

  i = (x – mitjana) / desviació_estàndard

  mitjana = sum(x) / compte(x)

  desviació_estàndard = sqrt(sum((x - mitjana) ^ 2) / recompte(x))

- Reescalar: canviar l’escala de les dades de la gamma original de manera que tots els valors observats estan dins l’interval 0 i 1. Un valor *i* es reescala de la següent manera:

  i = (i – min) / (max – min)

Per poder veure l’efecte d’aquest processos a continuació es mostra pas a pas l’estat d’una corba de glucosa (formada per les lectures de glucosa) des de l’inici fins al final del preprocés:

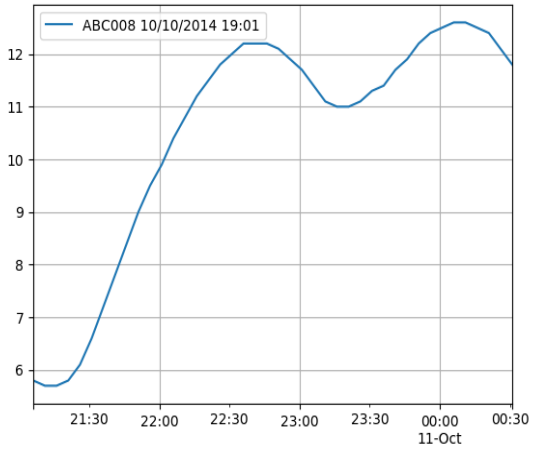


Figura 9.1: Estat Inicial

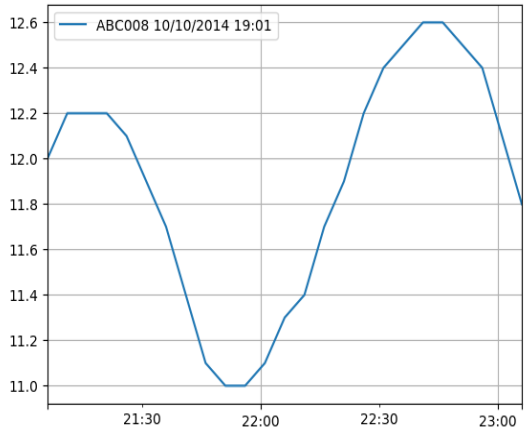


Figura 9.2: Filtratge aplicat

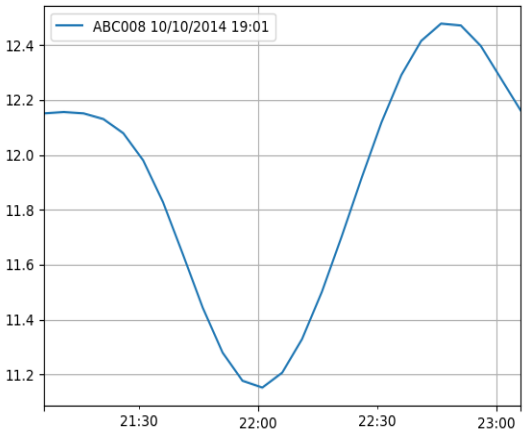


Figura 9.3: “Smooth” aplicat

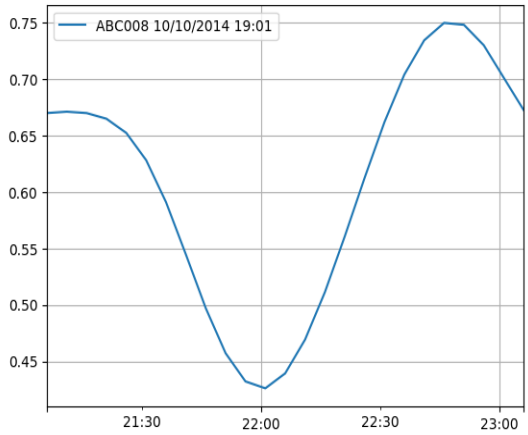


Figura 9.4: Normalització aplicada

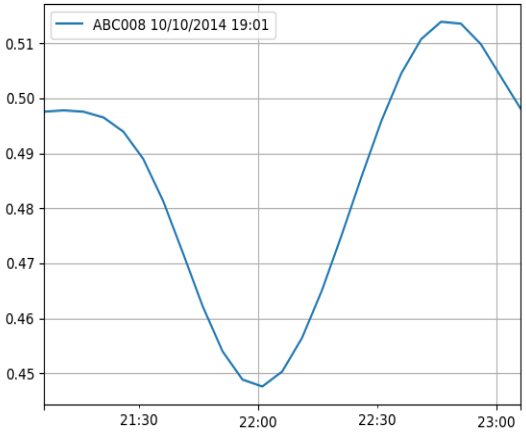


Figura 9.5: Reescalament aplicat

En les fase inicials del projecte, es treballava directament carregant les dades del fitxer en estructures de dades de python (en general pandes.DataFrames). A mesura que es va anar avançant es va detectar que, tot i funcionar correctament, el fet de carregar les dades des d’un fitxer cada vegada que s’iniciava el sistema incrementava molt el temps de resposta cap al usuari final de l’aplicació client-servidor.

En aquest moment es va decidir utilitzar un gestor de base de dades de manera que la carrega de dades només s'hagués de fer un cop (el primer cop que s'iniciï el sistema).

D'aquesta manera es va millorar molt el rendiment general del sistema i s'evitava realitzar tots els passos de preprocés cada vegada que s'iniciava el sistema, ja que es guarden a la base de dades les lectures ja processades.

Representació de Sessions: Generació de Histogrames

Les corbes resultants del preprocés proporcionen unes dades amb uns valors que resulten més bons alhora de treballar-hi que els inicials (per exemple, és molt important en algorismes com el KNN o el SVM que les dades estiguin escalades entre 0 i 1).

Tot i això, va sorgir un problema: es tracte de series temporals i aquest tipus de estructures presenten varis problemes alhora de treballar-hi, per exemple, que no tenen el mateix número de valors. Una sèrie pot estar representada per 25 lectures mentre que una altre pot constar només de 15. Aquest fet impossibilita comparar directament les corbes de glucosa en l'estat actual i per tant, no poden ser utilitzades en algorismes de Machine Learning.

Per tant, va sorgir la necessitat de trobar una primera representació de les corbes amb que el generadors de models poguessin treballar. La solució trobada va ser la de treballar amb els histogrames de valors de les corbes en comptes de amb les corbes directament.

Un histograma d'una corba conté tota la informació rellevant sobre la sessió (valors de lectures ocorreguts) i l'únic que es perd és la informació temporal de les lectures (saber que la lectura de valor 11.5 es va produir el dia X a la hora Y). Tot i això, al treballar amb seqüències es pot mantenir certa informació sobre la temporalitat.

Abans de poder generar aquest histogrames és necessari un últim preprocés. Per tant, un cop finalitzada la "neteja" de lectures explicada en l'apartat anterior, el que es fa és arrodonir tots els valors de les lectures a intervals d'un valor determinat (per exemple, cada 0.025). Gràcies a això aconseguim que els possibles valors de les lectures de totes les sessions siguin els mateixos: de 0 a 1 en un període X.

En aquest punt ja es poden generar els histogrames, que es guarden a la taula SessionsFDP mostrada en el diagrama de la base de dades de la figura 8.15, i comença la feina dels ModelsGenerator.

9.2.2 ModelsGenerator

Aquesta classe és l'encarregada de la generació dels models de les sessions amb la que treballa el Predicter.

A continuació s'explicarà el funcionament de la classe "pare", el ModelsGenerator, i en els següents apartats s'explicarà cadascuna de les classes ModelsGenerators creades que s'han acabat utilitzant així com l'estructura detallada dels models que generen.

Per veure el codi complet de tots els ModelGenerators es pot consultar l'apartat 14.3 de l'annex.

ModelsGenerator

Tal i com s'ha explicat en l'apartat 8.2.2 de patrons, es tracta de la classe que implementa el decorador en el patró "Decorator".

Un ModelsGenerator te com a atributs un Loader i una altre objecte ModelsGenerator que serà el generador "fill". Quan s'instancien s'han de passar com a paràmetre un dels dos de manera que si es proporciona un Loader es tractarà del primer generador i no tindrà cap generador

"fill"; i si es proporciona un generador es tractarà d'un generador "pare" i s'obtindrà el Loader del generador "fill" que s'ha passat per paràmetre (veure figura 9.6).

```
def __init__(self, model_generator=None, loader=None):
    if model_generator:
        self.loader = model_generator.loader
        self.child_mgen = model_generator
    elif loader:
        self.loader = loader
        self.child_mgen = None
    else:
        raise Exception("Loader required")
    self.cursor = self.loader.cursor
```

Figura 9.6: Constructor del ModelsGenerator

A la classe ModelsGenerator es defineix una interfície dels mètodes que totes les classes "filles" han d'implementar. Aquest mètodes son:

- generate_models()
- get_models()
- get_models_labels()

El funcionament en aquests tres mètodes és sempre el mateix, primer es crida el mateix mètode per el generador fill (si se'n té) i després es realitzen les tasques pròpies del generador actual (veure figura 9.7):

```
def generate_models(self, args_dict):
    """ Generate models of sessions """
    if self.child_mgen:
        self.child_mgen.generate_models(args_dict)
    self._generate_models(args_dict)

def get_models(self, session_ids, args_dict=None):
    """ Returns models of given sessions"""
    if args_dict is None:
        args_dict = {}
    return self._get_models(session_ids, args_dict)

def get_models_label(self, session_ids):
    """ Returns labels of models of given sessions """
    return self._get_models_label(session_ids)
```

Figura 9.7: Mètodes del ModelsGenerator

Com que aquest comportament és igual en tots els generadors i l'únic que es canvia és la implementació concreta realitzada després de cridar el mètode del generador fill es va encapsular aquest comportament en uns altres mètodes que són els que realment han d'implementar les classes que heretin del ModelsGenerator:

```
def _generate_models(self, args_dict):
    return None

def _get_models(self, session_ids, args_dict):
    return None

def _get_models_label(self, session_ids):
    return None
```

Figura 9.8: Mètodes per ser implementats per els fills

Aquests mètodes, tal i com es pot observar en la figura 9.8, en la classe ModelsGenerator no realitzen cap tasca ja que només estan definint la interfície per les classes filles, que implementaran tota la seva lògica en aquests mètodes.

SimpleModels

Es tracta del generador base, el més senzill de tots. Qualsevol altre generador tindrà sempre com a fill base (l'últim fill) el SimpleModels., per tant és aquest a qui se l'hi proporciona el Loader en el constructor.

No realitza cap tasca addicional en la generació de models. Els models que proporciona son els histogrames de sessions guardats en la taula SessionsFDP que s'han explicat en anteriorment.
El model d'una sessió té la següent estructura:

- [ID; valor_histograma_1, ... , valor_histograma_N, Label]

LmFitModels

Aquest generador treballa un nivell per sobre del SimpleModels, és a dir, que utilitza els models proporcionats per el SimpleModels per construir els seus models.

Com es pot deduir per el seu nom, aquest generador utilitza les capacitats de la llibreria lmfit (introduïda en l'apartat 7.1.5) per construir models de funcions no lineals. Concretament s'utilitza per la seva capacitat de construir models de funcions gaussianes.

En la generació de models, es construeix la funció gaussiana corresponent a cadascun dels histogrames que proporcionen els models del SimpleModels.

En la obtenció de models, es proporcionen els paràmetres característics de les funcions gaussianes construïdes a partir dels histogrames. Es tracte del paràmetres que defineixen una distribució gaussiana: amplitud, centre i sigma.
El model d'una sessió te la següent estructura:

- [ID; amplitud, centre, sigma, Label]

Utilitzant distribucions gaussianes per representar les sessions en comptes de treballar directament amb els histogrames s'esperava millorar els resultats eliminant soroll i treballant amb models que estiguessin formats per menys atributs (el model gaussià només necessita 3 atributs mentres que en l'histograma cada valor que apareix correspon a un atribut). Però va introduir de nou el problema del temps de resposta: la generació de models és un procés molt lent i costos. Com a solució es va construir el generador LmFitWordsModels que s'explica més endavant.

NormalizedModels

Aquest generador no proporciona models en si, només modifica les dades amb les que treballarà el generador que estigui per sobre seu. Al generar models, realitza una tasca de normalització de tots els atributs de les sessions (carbohidrats, insulina, etc.), de manera que tots els valors es trobin sempre entre 0 i 1.

En la obtenció de models, proporciona els mateixos que el seu generador fill.

WordsModels

En aquest generador és on es crea el vocabulari de sessions. Sempre va per sobre d'un SimpleModels o d'un LmFitModels.

En la generació de models, es crea un vocabulari a partir dels models proporcionats per el generador fill i s'etiqueta cada sessió amb la paraula que se l'hi assigna. La tècnica utilitzada per generar el vocabulari és la tècnica del K-Means ja explicada en l'apartat 5.3.1. Els clústers que es creen amb aquesta tècnica és el que s'anomenen paraules i el seu conjunt és el vocabulari.

No es crea un sol vocabulari directament a partir de totes les sessions, sinó que es creen tres vocabularis: un de hiperglucèmies, un de hipoglucèmies i finalment un de sessions normals. Per fer-ho s'aplica la tècnica del K-means per separat a les sessions etiquetades amb cada classe.

La raó per la qual es generen tres vocabularis per separat en comptes d'un de sol és que durant el desenvolupament es va detectar el problema de que les dades amb les que es treballava estaven molt desbalancejades, és a dir, hi havia moltes sessions d'un tipus (hiperglucèmia) i molt poques d'un altre (normals). Aquest fet implicava que al generar un sol vocabulari hi hagués moltes paraules que representessin hiperglucèmies i molt poques que en representessin de normals. La solució de generar tres vocabularis per separat assegurava que hi hagués el mateix nombre de paraules de cada tipus i per tant cada classe tingues varies representacions.

En la obtenció de models, es proporcionen els valors de les paraules amb que s'ha etiquetat cada sessió. A més també es proporcionen la resta de atributs que es te sobre les sessions. El model d'una sessió te la següent estructura:

- [ID; edat, sexe, tipus_apat, glucosa_inicial, carbohidrats, insulina, exerciciBf, exerciciAf, alcohol, valors_de_la_paraula_assignada, Label]

LmFitWordsModels

Es tracta d'una variant del generador WordsModels explicat anteriorment que soluciona el problema plantejat en l'explicació del generador LmFitModels. Recordem que aquest problema consistia en que la generació de models gaussians a partir dels histogrames augmentava massa el temps de resposta del sistema.

Igual que en el WordsModels es genera el vocabulari a partir dels models obtinguts del generador fill. La diferencia està en que, un cop generat el vocabulari, es realitza un altre procés sobre les paraules generades: per cada paraula generada es construeix el seu model gaussià corresponent i es guarda en la base de dades.

D'aquesta manera obtenim els beneficis de treballar amb models gaussians en comptes de directament amb histogrames i evitem el problema del temps de resposta ja que només s'han de obtenir els models de les paraules (per exemple, 30) en comptes de fer-ho per totes les sessions (per exemple, 1000).

Del fet de necessitar els histogrames per la generació dels models gaussians es pot deduir que aquest generador sempre tindrà com a fill el SimpleModels, a diferència del WordsModels que podia tenir qualsevol generador com a fill.

En la obtenció de models es proporcionen els paràmetres característics de les funcions gaussianes construïdes a partir dels histogrames de les paraules.

El model d'una sessió te la següent estructura:

- [ID; edat, sexe, carbohidrats, insulina, exerciciBf, exerciciAf, alcohol, amplitud_paraula, centre_paraula, sigma_paraula, Label]

SequenceModels

Es tracte del generador en que es construeixen les seqüències. Pot tenir com a generador fill qualsevol altre generador gràcies a la implementació genèrica amb que s'ha desenvolupat però en la construcció del sistema predictiu sempre es posa per sobre d'un generador de paraules.

En la generació de models, es creen les seqüències de 'n' sessions. Per fer-ho, per cada seqüència s'obtenen les 'n' seqüències anteriors del mateix pacient. Llavors es guarden a la taula de la base de dades Sequences amb el següent format:

- sessio_id (x), sessió anterior (x-n), ... , sessió anterior (x-1)

Llavors, en la obtenció de models l'únic que s'ha de fer és obtenir els models de cadascuna de les sessions de la seqüència a partir del generador fill. El model d'una sessió te la següent estructura:

- [ID; edat, sexe, tipus_apat, glucosa_inicial, carbohidrats, insulina, exerciciBf, exerciciAf, alcohol, info_sessio_anterior_1, info_sessio_anterior_2, ... , info_sessio_anterior_n, Session_Label]

Fins aquest punt, en la resta de generadors l'atribut 'Label' sempre feia referencia a una etiqueta que diferenciés entre hiperglucèmia, normal i hipoglucèmia (normalment representats amb 1, 0 i -1 respectivament). Però en fases avançades del desenvolupament del projecte es va plantejar la possibilitat de predir la paraula que s'assignaria a la sessió en comptes de només la label. Predient la paraula que s'assignaria en un futur es proporcionaria molta més informació al usuari final ja que no només s'informaria de si es produeix hiperglucèmia, hipoglucèmia o normal, sino que també es proporcionaria una aproximació de com seria la corba de glucosa (recordem que una paraula es tracta d'un histograma de lectures de glucosa que representa un prototipus de sessió). Per això en aquest generador l'últim atribut que fins ara era simplement 'Label', s'ha anotat com a 'Session_Label'.

Haver de predir la paraula, una etiqueta amb més de 20 possibilitats (una per paraula existent), en comtes d'una simple etiqueta de 3 possibilitats va tenir efectes negatius en els resultats i va comportar una sèrie de canvis en el Predicter per tal de alleujar aquests efectes. Aquests canvis s'explicaran en l'apartat 9.2.3. En referencia al generador actual, l'únic que va comportar va ser haver de canviar l'últim atribut dels models per la paraula corresponent a cada sessió (de 'Label' a 'Session_Label').

Estructura del Generador

Tal i com s'ha explicat en l'apartat de patrons de la secció 8.2.2, gràcies a l'aplicació del patró decorator en la implementació dels generadors podem formar qualsevol estructura de generadors. Tot i això, hi ha certs generadors que no te sentit combinar entre ells.

L'estructura general de decoradors que s'ha seguit durant el desenvolupament ha estat sempre el següent:

GeneradorDeSeqüències(GeneradorDeParaules(... SimpleModels()))

És a dir, sempre s'ha treballat amb els decoradors de manera que es formin seqüències de paraules. Llavors aquestes paraules es generen a partir de, com a mínim, un SimpleModels. Gràcies al patró decorator, podem definir la quantitat de classes intermitjes entre el SimpleModels i el generador de paraules que es vulguin sense haver d'alterar la manera de funcionar del sistema.

9.2.3 Predicter

Classe encarregada de realitzar les prediccions sobre la classe a la que pertany unes determinades sessions.

En termes de implementació, segueix una estructura similar al ModelsGenerator explicat en l'apartat anterior: hi ha una classe "pare" anomenada Predicter i un seguit de classes filles que

implementen diferents sistemes de predicció. En aquest cas però, no s'està implementant el patró decorator. Senzillament s'ha encapsulat el comportament general en la classe pare i 'han desenvolupat varies classes amb una herència simple que defineixen comportaments més específics.

Per veure el codi complet de tots els Predicters desenvolupats es pot consultar l'apartat 14.3 del annex.

Predicter

Classe encarregada de realitzar les prediccions. Consta d'un objecte Loader a través del qual es generen i s'obtenen els models de les sessions que s'utilitzen per fer les prediccions.

Aquest objecte Loader es proporciona a través del constructor junt amb els paràmetres que defineixen característiques dels models generats. És en el constructor mateix on es generen els models de les sessions amb les que es treballarà en el futur.

```
def __init__(self, loader, words_num=20, seq_len=3):
    """
    :param
    loader: object of the type Loader. It contains all the data about CGM and
            is able to generate the models with which the classifier will work.
    """
    self.loader = loader
    self.loader.generate_models({'k': words_num, 'ns': seq_len})
```

Figura 9.9: Constructor objecte Predicter

Tal i com es pot observar en la figura 9.9, els paràmetres que es proporcionen defineixen dos aspectes:

- El número de paraules del vocabulari que es generarà a través del WordsModels
- El número de sessions que formaran part de cada seqüència generada a través del SequenceModels

A més de generar els models, la classe predicter defineix el mètode “predict”, que com el seu nom indica és l’utilitzat per a realitzar les prediccions (figura 9.9):

```
def predict(self, session_ids, train_sessions=None):
    """
    :param test_index:  indexs of the session to predict
    :param train_index: indexs of the data used as train to built the predict_obj.
                        If no train_index is given, it will be used all the data exept the ones of the 'model_indexs'.
    :return: labels predicted
    """
    if not train_sessions:
        train_sessions = self.loader.get_all_sessions(for_train=True)
        train_sessions = [x for x in train_sessions if x not in session_ids]
    res = self._predict(train_sessions, session_ids)
    return res
```

Figura 9.9: Mètode “predict” del objecte Predicter

En *Machine Learning* la capacitat de classificar o realitzar prediccions s’aconsegueix, bàsicament, aplicant una sèrie de algorismes sobre unes dades ja classificades a les que s’anomena dades de entrenament.

En aquest mètode s’implementa la part referent a obtenció de les sessions que seran utilitzades per a entrenar el sistema, que és la part comuna per a tots els objectes Predict.

L’obtenció dels models d’aquestes sessions, l’entrenament del sistema i la realització de la predicció s’encapsula en el mètode “_predict”. Aquest és el mètode que cada fill implementarà i que es troba buit en la classe tal i com es mostra en la figura 9.10:

```
def _predict(self, train_index, model_indexs):
    """To be implemented by childs"""
    return None
```

Figura 9.10: Mètode “_predict” del objecte Predicter

KNNPredicter

Aquesta classe implementa l'algorisme del K-nearest-neighbours (explicat en l'apartat 5.3.2) per a realitzar prediccions.

Per fer-ho utilitza l'objecte *KNeighboursClassifier* proporcionat per la llibreria skit-learn (explicada en l'apartat 7.1.2). Es tracte d'una implementació molt completa de l'algorisme KNN que fins i tot permet definir els paràmetres de l'algorisme com el número de veïns a tenir en compte (la 'K').

Tots els objectes de la llibreria skit-learn utilitzats per fer les prediccions ofereixen la mateixa interfície per treballar-hi:

- Es construeix l'objecte per fer les prediccions, en aquest cas un *KNeighboursClassifier*. En el constructor es poden proporcionar paràmetres utilitzats en l'algorisme com la 'K' (número de veïns).
- S'entrena l'objecte a través del mètode "fit()" passant com a paràmetre un DataFrame (veure apartat 7.1.4) amb els models d'entrenament i un vector amb les etiquetes de cadascun dels models d'entrenament: KNN.fit(models, labels).
- Es realitzen les prediccions utilitzant el mètode "predict()" passant com a paràmetre un dataframe amb els models dels elements que es volen classificar. Aquest DataFrame tindrà els mateixos atributs que el dels models utilitzats per entrenar l'objecte.
- El mètode "predict()" retornarà una llista amb les etiquetes amb que s'ha classificat cada model.

Utilitzant l'objecte loader s'obtenen els models de les sessions d'entrenament i de les sessions que s'han de predir. En aquest segon cas, l'últim atribut dels models que fa referència a l'etiqueta estarà buit ja que és el que es vol predir.

En l'apartat 9.2.2, concretament en la subsecció en que s'explica el SequenceModels, es va exposar un canvi en els requeriments que va sorgir en fases avançades del desenvolupament

del projecte. Aquest canvi consistia en el fet de haver de predir la paraula ('Session_Label') amb que s'etiquetaria la sessió en comptes de predir només si es tractaria d'una hiperglucèmia, hipoglucèmia o normal ('Label').

Aquest fet implicava passar de predir 1 possibilitat entre 3 a haver de predir 1 possibilitat entre més de 12 com a mínim (concretament 1 entre el número de paraules del vocabulari, que sempre és major de 12). Òbviament aquest fet va tenir un impacte en els resultats del sistema predictiu ja que la dificultat de realitzar prediccions correctes és directament proporcional al número de etiquetes possibles

Per intentar minimitzar els efectes d'aquest canvi es van implementar una sèrie de passos addicionals en el mètode "_predict" de manera que es realitza una doble predicció: primer per a 'Label' (hiperglucèmia, hipoglucèmia o normal) i després per a 'Session_Label' (paraula del vocabulari que se li assignarà a la sessió). Donats 2 conjunts de sessions, un d'entrenament E i un a predir S:

- Obtenció de les sessions d'entrenament E a través del *loader.get_models().*
- Substitució de les 'Session_Label' (que representen la paraula assignada a la sessió) per l'etiqueta que representi si la paraula corresponia a hiperglucèmia, hipoglucèmia o normal ('Label').
- Construcció del objecte 'P' per fer prediccions (KNN) sobre 'Label'.
- Entrenament del objecte 'P' utilitzant els models de les sessions d'entrenament E a través del mètode *fit().*
- Obtenció de les sessions a predir S a través del *loader.get_models().*
- Realització de les prediccions utilitzant el mètode *predict()* del objecte *P*. Es tracte de la primera predicció, que ens diran si les sessions a predir són hiperglucèmia, hipoglucèmia o normal ('Label').
- Substitució de les 'Label' per les 'Session_Label' originals.
- Divisió dels conjunts E i S en tres subconjunts cadascun agrupant per el resultat de la primera predicció de manera que s'obtenen els subconjunts:

    - E1, E2 i E3: subconjunts de E formats per les sessions predites com a hiperglucèmia, hipoglucèmia i normal respectivament.
    - S1, S2 i S3: subconjunts de S formats per les sessions predites com a hiperglucèmia, hipoglucèmia i normal respectivament.
- Construcció d'un nou objecte per fer prediccions 'P2' (KNN) sobre 'Session_Label'.
- Realització de la segona predicció utilitzant l'objecte 'P2'. Es fa per cada parella (Ei, Si).

En resum, el que es fa és fer primer la predicció sobre si la sessió serà hiperglucèmia, hipoglucèmia o normal ('Label') i llavors fer la predicció de la paraula de la sessió ('Session_Label'), però en aquesta segona predicció només s'utilitzen les sessions que siguin de la mateixa classe que ella.

D'aquesta manera es redueix a 1/3 les possibilitats de la etiqueta referent a la paraula a predir ja que només es treballa amb un dels 3 vocabularis (hiperglucèmia, hipoglucèmia o normal) segons el resultat de la primera predicció.

SVMPredicter

Aquesta classe implementa l'algorisme *Support Vector Machine* (explicat en l'apartat 5.3.3) per a realitzar prediccions. Igual que amb el KNNPredicter, utilitza un objecte de la llibreria skit-learn que ofereix la mateixa interfície per a construir i entrenar el sistema i finalment realitzar les prediccions.

Els passos a seguir en el mètode "_predict" per realitzar les prediccions són el mateixos que els explicats en el KNNPredicter amb la diferència que s'utilitza l'algorisme del SVM en comptes del KNN.

En aquest cas els paràmetres disponibles per personalitzar el sistema son els següents:

- Kernel: tipus de kernel utilitzat en l'algorisme. Pot ser lineal, polinòmic, rbf, ...
- Gamma: coeficient del kernel en cas que sigui polinómic o rbf.
- C: penalització del error

SVM_KNN_Predicter

Es tracte de una classe que combina els algorismes de SVM i KNN a l'hora de realitzar les prediccions. Com ja s'ha explicat en l'apartat del KNNPredicter, es realitzen 2 prediccions: primer per predir la etiqueta hiperglucèmia/hipoglucèmia/normal utilitzant l'objecte P1 i després per obtenir la paraula utilitzant l'objecte P2.

En els Predicters anteriors tan P1 com P2 utilitzaven el mateix algorisme (KNN o SVM). En aquesta classe els objectes P1 i P2 utilitzen algorismes diferents. Per predir hiperglucèmia/hipoglucèmia/normal (objecte P1) s'utilitza SVM, en canvi per predir la paraula (objecte P2) s'utilitza KNN.

La raó de utilitzar primer SVM i després KNN no és cap altre que millorar el temps de resposta. En el SVMPredicter, es gasta molt de temps en fer les prediccions de les paraules ja que en situacions amb moltes classes en l'algorisme del SVM és molt costos trobar els hiperplans que separin aquestes classes. Al utilitzar KNN en comptes de SVM en aquestes situacions es millora el temps de resposta sense penalitzar els resultats respecte les prediccions de hiperglucèmia/hipoglucèmia/normal ja que l'objecte P1 continua sent un SVM.

9.3 Proves

Com ja s'ha explicat en la introducció d'aquest apartat, al ser aquest un projecte de recerca les proves que s'han realitzat han estat enfocades a mesurar la eficàcia del sistema desenvolupat en comptes de realitzar proves dels components del sistema.

Evidentment això no implica que no s'hagi comprovat el correcte funcionament de cadascun dels mètodes i classes implementades. Cada mètode ha estat testat amb el debugger proporcionat per l'IDE (en aquest cas PyCharm) per observar que s'obtenia l'esperat, però no s'ha seguit la tècnica de testeig de les proves unitàries.

Per tal de mesurar la eficàcia del sistema predictiu que s'ha desenvolupat s'han aplicat un seguit de mètriques utilitzant la tècnica de validació creuada o cross-validation. Tot això s'ha implementat en la classe CGMTester.

A continuació s'explicarà en que consisteix la tècnica de la validació creuada, les mètriques utilitzades i finalment com s'ha implementat a la classe CGMTester.

9.3.1 Cross-Validation

La tècnica de la validació creuada és una tècnica que serveix per avaluar els resultats d'una sèrie de anàlisis estadístics sobre unes dades i garantir que aquests resultats son independents a les característiques d'aquestes dades [20][21]. Per fer-ho s'aplica la mètrica repetidament sobre diferents subconjunts de les dades d'entrenament.

Es tracta d'una tècnica molt utilitzada en sistemes com el desenvolupat en el present treball en que es realitza una predicció i es vol mesurar la precisió d'aquesta.

Existeixen varis tipus de validacions creuades, la implementada en aquest treball ha estat la anomenada "k-folds cross-validation" o valdidació creuada de K iteracions.
Aquesta tècnica consisteix en el següent:

- Es divideixen les dades en K subconjunts.
- Es repeteix K vegades el següent:
    - Escollir un dels subconjunts com a dades de prova i la resta com a dades d'entrenament.
    - Entrenar el sistema amb les dades d'entrenament.
    - Realitzar les prediccions amb les dades de prova.
    - Aplicar les mètriques que corresponguin i obtenir el resultat.
- Realitzar la mitjana aritmètica de tots els resultats obtinguts.

Els subconjunts que es construeixen es generen de forma que es respecti la proporció de dades de cada classe (stratified k-folds). És a dir, si en les dades tenim que un 40% de instàncies corresponen a la classe 1 i un 60% a la classe 2; tots els subconjunts es generaran intentant mantenir aquesta proporció.

Gràcies a la tècnica de la validació creuada, es poden realitzar proves en que les dades de test i entrenament van canviant. D'aquesta manera la configuració no estarà esbiaixada cap a unes dades concretes.

S'ha aplicat cross-validation en diferents escenaris per trobar els millors paràmetres del sistema. Els paràmetres que s'han de ajustar son:

- Trobar la millor mida del vocabulari que es genera, és a dir, el nombre de paraules amb que es treballarà. Els tests s'han realitzant utilitzant el generador de models de l'escenari EsM1 i el SVMPredicter amb un llargada de seqüències fixada en 3.

- Trobar la llargada de les seqüències amb les que es treballarà, és a dir, el número de sessions a tenir en compte en les seqüències. Els tests s'han realitzant utilitzant el generador de models de l'escenari EsM1 i el SVMPredicter amb un vocabulari de mida 15.

- Trobar la millor configuració del objecte Predicter utilitzat. Per el KnnPredicter només s'ha de escollir la 'k' (nombre de veïns). Per la resta s'han de escollir els paràmetres de l'algorisme del SVM: gamma, kernel, C, etc. Els tests s'han realitzant utilitzant el generador de models de l'escenari EsM1 i el SVMPredicter, un llargada de seqüències fixada en 3 i un vocabulari de mida 15.

- Trobar el millor objecte Predicter. Escenaris testejats:
    - **EsP1**: KNNPredicter
    - **EsP2**: SVMPredicter

- Trobar la millor combinació de ModelsGenerator. Les possibilitats son totes les combinacions de ModelsGenerators que incloguin 1 generador de seqüències, 1 generador de paraules i 1 o varis generadors per sota del generador de paraules. Els escenaris testejats són:
    - **EsM1**: SequenceModels + WordsModels + SimpleModels
    - **EsM2**: SequenceModels + LmFitWordsModels + SimpleModels
    - **EsM3**: SequenceModels + WordsModels + LmFitModels + SimpleModels
    - **EsM4**: SequenceModels + WordsModels + NormalizedModels + SimpleModels
    - **EsM5**: SequenceModels + LmFitWordsModels + NormalizedModels + SimpleModels
    - **EsM6**: SequenceModels + WordsModels + LmFitModels + NormalizedModels + SimpleModels

    Els 6 escenaris s'han testejat utilitzant la mateixa mida de vocabulari (15 i 45) i llargada de seqüències (3 i 10). Els predicters utilitzants han estat el SVMPredicter i el KNNPredicter. Tan en el SVMPredicter com en el KNNPredicter s'ha utilitzat la mateixa configuració de paràmetres per els 6 escenaris:
    - SVMPredicter: una c de 10 i 100 ; una gamma de 0.1 i 1 i un kernel 'rbf'.
    - KNNPredicter: una k (numero de veïns) de 5, 10 i 20.

Totes les proves realitzades per tal de trobar la millor configuració del sistema així com els resultats concrets que s'han obtingut i el resultat final s'explicaran l'apartat de 10 de resultats.

Per totes les proves s'han aplicat les mateixes mètriques les quals s'explicaran a continuació.

9.3.2 Mètriques

Mitjançant la tècnica de la validació creuada explicada anteriorment s'apliquen una sèrie de mètriques que serveixen per determinar la eficàcia del sistema.

Aquestes mètriques s'apliquen sobre les matrius de confusió construïdes a partir dels resultats obtinguts i els resultats esperats. Les matrius de confusió son unes eines molt utilitzades en el camp de la Intel·ligència Artificial per avaluar l'eficàcia de algorismes d'aprenentatge supervisat (com els utilitzats en aquest treball: SVM i KNN).

Una matriu de confusió es construeix de la següent manera:

- Matriu de n * n, on 'n' és el nombre de classes
    - Hi ha una fila per cada classe
    - Hi ha una columna per cada classe
- A la casella Cij, de la fila 'i', columna 'j', es posa el nombre d'elements que l'algorisme ha predit amb una classe 'j' i que en realitat pertany a la classe 'i'.
- La suma de tots els elements d'una columna 'j' és el total d'elements predits com a classe 'j'.
- La suma de tots els elements d'una fila 'i' és el total d'elements de la classe 'i'.

Per exemple, si tinguéssim un sistema en que s'ha de classificar una instància entre tres classes diferents:

		Valors Predits		
		Classe 1	**Classe 2**	**Classe 3**
Valors Reals	**Classe 1**	10	3	4
	Classe 2	1	7	2
	Classe 3	9	6	5

D'aquesta matriu es pot interpretar, per exemple, que 10 instàncies de la classe 1 han estat predites correctament com a classe 1. En canvi 3 instàncies de la classe 1 han estat classificades incorrectament com a classe 2.

Vist des de la perspectiva d'una classe, una predicció pot pertànyer a 4 grups diferents:

- True Positive (TP): instàncies de la classe X classificades correctament com a classe X (cel.les C_{ii}).

- False Positive (FP): instàncies d'una classe Y (Y!=X) classificades incorrectament com a classe X.

- True Negative (TN) : instàncies d'una classe Y (Y!=X) classificades correctament com a no X.

- False Negative (FN): instàncies de la classe X classificades incorrectament com a no X.

A partir d'una matriu de confusió, computant els seus TP, FP, TN i FN es poden extreure una gran quantitat de mètriques.

Algunes de les més importants i les utilitzades en el present treball són:

- Exactitud (Accuracy): mesura la proporció de prediccions que son correctes entre totes les prediccions fetes.
    - (TP + TN) / (TP+TN+FP+FN)

- Sensibilitat (Recall): capacitat de classificar com a classe X les instàncies que realment son classe X. És a dir, proporció d'instàncies classificades com a classe X entre totes les que realment son classe X.
    - TP / (TP + FN)

- Precisió (Precision): proporció de positius correctes entre tots els positius. És a dir, proporció d'instàncies que són realment classe X entre totes les classificades com a classe X.
    - TP / (TP + FP)

- F1: mitjana harmònica que combina la precisió i la sensibilitat.
    - 2*TP / (2*TP + FP + FN)

- AUC: es tracte de l'àrea sota la corba ROC. La corba ROC és una representació gràfica de la sensibilitat (TP / (TP + FN)) respecte l'especificitat (TN / (TN + FP)).

La AUC és una mesura que s'aplica en sistemes binaris (amb 2 classes). Per poder-la aplicar en el sistema predictiu desenvolupat, en el qual es treballa amb 3 classes (hiperglucèmia, hipoglucèmia i normal), s'ha de binaritzar les matrius de confusió que es generen de manera que de la matriu original en resultin 3 de noves (una per classe) les columnes i files de les quals representen si una instància és/s'ha predit com "Classe X" o "No Classe X", on "X" serà hiperglucèmia, hipoglucèmia o normal.

9.3.3 CGMTester

Per realitzar les proves del sistema predictiu desenvolupat es va dissenyar la classe CGMTester, que implementa la tècnica de la validació creuada i les mètriques explicades.

Quan s'instància un CGMTester, se l'hi passa el nom d'un fitxer amb totes les dades dels pacients. En el constructor es construeix un Loader utilitzant aquest fitxer i llavors es realitza el procés de "neteja" de lectures i creació dels histogrames guardats en la taula SessionFDP. Les proves que es realitzaran utilitzaran les dades carregades en aquest procés.

Per la realització de les proves es va implementar el mètode “test”. Aquest mètode rep com a paràmetres:

- params: llista de llistes amb els paràmetres corresponents al objecte Predicter, el número de paraules del vocabulari i la llargada de les seqüències. Per exemple, les següents llistes donen valor als paràmetres kernel, c i gamma del SVM; a la mida del vocabulari i a la llargada de les seqüències:

  [[“poly”, 1, 0.0001, 30, 5],
  [“linear”, 1, 0.0001, 30, 5],
  [“rbf”, 1, 0.0001, 30, 5],
  [“sigmod”, 1, 0.0001, 30, 5],]

- predict_obj_names: llista d’objectes Predict amb que es realitzaran proves. Per exemple:

  [“KNNPredicter”, “SVMPredicter”, “SVM_KNN_Predicter”]

- model_generator_names: llista de tuples que contenen estructures de ModelGenerators a utilitzar. Per exemple:

  [(“SequenceModels”, “WordsModels”, “SimpleModels”),
  (“SequenceModels”, “LmFitWordsModels”, “SimpleModels”),
  (“SequenceModels”, “WordsModels”, “LmFitModels”, “SimpleModels”),
  (“SequenceModels”, “LmFitWordsModels”, “NormalizedModels”, “SimpleModels”),
  (“SequenceModels”, “WordsModels”, “NormalizedModels”, “SimpleModels”)]

- nfolds: número de folds utilitzats en el mètode de la validació creuada. Per defecte sempre es treballa amb 10 folds.

- prefix: text que s’afegeix al inici del nom del fitxer que es crea amb els resultats.

El funcionament del mètode “test” és el següent:

1. Per cada objecte Predicter P a predict_obj_names:

1.1 Per cada estructura de ModelsGenerators:

    1. Crear els ModelsGenerators seguint l’estructura proporcionada.
    2. Assignar al objecte Loader el ModelGenerator creat.
    3. Per cada llista de paràmetres a params:
        I. Crear l’objecte Predicter P passant com a paràmetre l’objecte Loader de que disposa el CGMTester i els paràmetres de la llista actual. Es generen els models de les sessions.
        II. Obtenir tots els models de les sessions.
        III. Crear els ‘n’ folds a partir dels models. Per cada fold:
            1. Obtenir sessions de de entrenament (la resta de folds).
            2. Realitzar prediccions utilitzant l’objecte Predciter P. Les sessions a predir son les del fold actual i les sessions d’entrenament son les obtingudes en el pas anterior. Es retorna una llista amb les etiquetes assignades a cada sessió.
            3. Obtenir llista amb les etiquetes reals de cada sessió.
            4. Construir matriu de confusió i guardar-la a una llista LM.
        IV. Construir una matriu de confusió a partir de totes les matrius de la llista LM (aquesta llista es reinicia per cada llista de paràmetres amb la que es treballa).
        V. Extreure mètriques de cada matriu de confusió de LM i de la matriu general construïda en el pas anterior. S’escriuen en un fitxer que es genera utilitzant com a nom la combinació de objectes Predicter, ModelsGenerator i paràmetres que s’han testejat.

Quan es realitzen les prediccions, es fixa un timeout ja que és possible que en l’algorisme de SVM depenent de la configuració es trigui una gran quantitat de temps a finalitzar.

El codi complet de la classe CGMTester es pot veure en l’apartat 14.3 de l’annex.

10 Resultats

10.1 Vocabularis

S'ha treballat amb mides de vocabularis que van des de 12 a 150 paraules. En general els resultats obtinguts amb les diferents mides no varien excessivament en el rang de 15 a 60. Amb vocabularis més grans s'obtenen pitjors resultats. Amb vocabularis més petits no empitjoraven els resultats però un nombre massa petit de paraules redueix molt la informació que es mostra sobre les corbes de glucosa (recordar que cada paraula del vocabulari es tracte de una possible representació dels histogrames de les corbes de glucosa dels pacients, per tant, si hi ha menys paraules hi ha menys representacions possibles).

En l'apartat 14.1 de l'annex es mostren les paraules dels vocabularis de mida 15 i 45.

10.2 Escenaris

S'han testat els escenaris mostrats en l'apartat 9.3.1 corresponents a les diferents opcions de ModelGenerators i als Predicters. Cada escenari de Predicters (de EP1 i EP2) s'ha testat amb tots els escenaris de ModelsGenerator (del EM1 al EM6). Per cada escenari resultant, s'ha realitzat proves amb els següents valors de cada paràmetre del sistema:

- Mida vocabulari: 15.
- Llargada seqüències: 3
- C (només per SVM): 10
- Gamma (només per SVM): 0.1
- K (només per KNN): 5

Els valors corresponents a C, Gamma i K son els obtinguts com a millors possibles al realitzar una "Grid Search". Els que fan referencia al vocabulari i les seqüències son resultat de tests anteriors.

Les proves s'han realitzat utilitzant *stratified* k-folds amb k=5. Al ser 'stratified' es manté la proporció original de sessions classificades com hiperglucèmia, normoglucèmia i hipoglucèmia en tots els folds. En les dades amb les que es treballa, aquesta proporció està desbalançejada: hi ha moltes més hiperglucèmies que normoglucèmies i hipoglucèmies. Aquest fet comporta que mètriques com per exemple la accuracy no siguin les millors per determinar la qualitat del sistema: si en un sistema amb un 80% de instancies de la classe A i un 20% de la classe B es prediu sempre que una instància serà de la classe A s'obtindrà una accuracy del 80%, un resultat a priori molt bo tot i que en realitat no s'estan realitzant prediccions correctament. Per aquesta raó es va escollir utilitzar la ROC AUC (veure apartat 9.3.2).

A l'hora de determinar la qualitat de un sistema basant-se amb la AUC es pren de referència la AUC que s'obtindria amb una classificador aleatori: un 0,5. La AUC pot prendre un valor entre 0 i 1 sent aquests els millors possibles. En les taules amb els resultats es mostra la AUC obtinguda per cada fold i per cada classe i la AUC combinant els resultats de tots els folds.

Com a informació addicional, també es mostren la accuracy i la F1 del total dels folds. Finalment també es mostra el percentatge de sessions que no s'han pogut classificar. Una sessió no es pot classificar quan l'algorisme de classificació (KNN o SVM) puntua amb el mateix valor 2 o més classes, per exemple puntuant amb 0.4 hiperglucèmia, 0.4 normoglucèmia i 0.2 hiperglucèmia.

EP1: KNNPredicter

		Folds					Total			
Escenari	Predicció	F1 AUC	F2 AUC	F3 AUC	F4 AUC	F5 AUC	AUC	Accuracy	F1	No Classificats
EM1	Hipo.	0,58	0,52	0,47	0,56	0,53	**0,53**	0,69	0,23	23%
	Norm.	0,53	0,56	0,55	0,52	0,52	**0,53**	0,73	0,22	
	Hiper.	0,59	0,58	0,51	0,61	0,52	**0,56**	0,55	0,64	
EM2	Hipo.	0,55	0,54	0,57	0,55	0,56	**0,55**	0,7	0,27	24%
	Norm.	0,61	0,58	0,58	0,63	0,52	**0,58**	0,73	0,29	
	Hiper.	0,57	0,55	0,6	0,58	0,54	**0,57**	0,56	0,63	
EM3	Hipo.	0,55	0,52	0,51	0,6	0,52	**0,54**	0,7	0,26	27%
	Norm.	0,58	0,55	0,6	0,53	0,52	**0,56**	0,73	0,25	
	Hiper.	0,6	0,57	0,58	0,65	0,56	**0,59**	0,57	0,66	
EM4	Hipo.	0,64	0,7	0,58	0,65	0,57	**0,62**	0,72	0,31	23%
	Norm.	0,53	0,66	0,63	0,58	0,52	**0,58**	0,74	0,28	
	Hiper.	0,65	0,66	0,69	0,62	0,61	**0,65**	0,61	0,68	
EM5	Hipo.	0,52	0,49	0,56	0,56	0,6	**0,55**	0,69	0,24	23%
	Norm.	0,52	0,56	0,55	0,54	0,49	**0,53**	0,72	0,29	
	Hiper.	0,53	0,53	0,68	0,59	0,62	**0,59**	0,6	0,67	
EM6	Hipo.	0,48	0,55	0,6	0,51	0,58	**0,54**	0,7	0,24	22%
	Norm.	0,52	0,63	0,67	0,61	0,54	**0,59**	0,73	0,36	
	Hiper.	0,59	0,59	0,68	0,6	0,66	**0,62**	0,6	0,66	

EP2: SVMPredicter

		Folds						Total			
Escenari	Predicció	F1 AUC	F2 AUC	F3 AUC	F4 AUC	F5 AUC		AUC	Accuracy	F1	No Classificats
EM1	Hipo.	0,47	0,48	0,44	0,47	0,55		**0,48**	0,75	0,02	2%
	Norm.	0,66	0,62	0,55	0,51	0,54		**0,53**	0,76	0,01	
	Hiper.	0,54	0,55	0,51	0,53	0,58		**0,5**	0,51	0,67	
EM2	Hipo.	0	0	0	0	0		**0**	0	0	99%
	Norm.	0	0	0	0	0		**0**	0	0	
	Hiper.	0	0	0	0	0		**0**	0	0	
EM3	Hipo.	0,51	0,47	0,47	0,46	0,56		**0,5**	0,75	0,0	0.5%
	Norm.	0,61	0,57	0,55	0,52	0,5		**0,51**	0,76	0,0	
	Hiper.	0,54	0,48	0,56	0,55	0,55		**0,51**	0,51	0,67	
EM4	Hipo.	0,6	0,73	0,55	0,57	0,63		**0,61**	0,78	0,14	32%
	Norm.	0,61	0,64	0,68	0,56	0,54		**0,6**	0,8	0,3	
	Hiper.	0,63	0,62	0,7	0,67	0,65		**0,65**	0,65	0,76	
EM5	Hipo.	0,44	0,46	0,53	0,55	0,52		**0,5**	0,51	0,29	11%
	Norm.	0,49	0,59	0,46	0,54	0,5		**0,52**	0,65	0,27	
	Hiper.	0,52	0,49	0,61	0,65	0,58		**0,57**	0,52	0,42	
EM6	Hipo.	0,55	0,56	0,56	0,48	0,57		**0,54**	0,65	0,26	14%
	Norm.	0,57	0,59	0,65	0,5	0,58		**0,57**	0,68	0,31	
	Hiper.	0,63	0,54	0,62	0,53	0,63		**0,59**	0,57	0,6	

El primer que es pot observar dels resultats és que el percentatge de no classificats es manté estable al 23% en els escenaris EP1 (KNN). Aquest percentatge es redueix si s'augmenta el valor de K però es penalitzava molt la qualitat de les prediccions (classificava amb més freqüència però també més erròniament).

Per altre banda, en els escenaris EP2 (SVM) el percentatge de no classificats varia molt segons els generadors utilitzats (escenaris del EM1 al EM6) tot i que sol ser millor que en el KNN. Destaca l'escenari EM2 en que no s'ha pogut classificar cap instància. Evidentment, aquesta configuració queda descartada quan es treballi amb SVM.

Els millors resultats s'obtenen en l'escenari EM4, que correspon a SequenceModels + WordsModels + NormalizedModels + SimpleModels, tan per KNN com per SVM; tot i que per SVM son lleugerament superiors. Si s'analitzen la resta de mètriques el SVM és superior tan en la accuracy com en la F1 tot i que la tassa de no classificats és més elevada.

Si es comparen els resultats (en la AUC) per classe de l'escenari EM4 es pot veure que la millor puntuació es troba en les prediccions de hiperglucèmies. En general, en tots els escenaris les prediccions hiperglucèmia/no hiperglucèmia sempre tenen unes puntuacions més altes degut al desbalanceig de les dades.

11 Conclusions

Es tracte d'un treball de recerca, que té com a objectiu final determinar si es poden realitzar prediccions sobre pacients amb diabetis basant-se amb les corbes de glucosa que històricament han tingut aquests pacients.

Amb aquesta finalitat es va plantejar el desenvolupament de dos sistemes: un sistema intel·ligent per realitzar prediccions sobre l'estat de pacients amb diabetis i una aplicació client-servidor com a proposta de interfície per interactuar amb el sistema predictiu.

Respecte el desenvolupament dels dos sistemes, es pot considerar que s'han complert els objectius establerts:

- S'ha desenvolupat un sistema complet i personalitzable capaç de realitzar prediccions aplicant un seguit de tècniques de Machine Learning.
- S'ha dissenyat una api i un servidor que proporcionen un portal a través del qual interactuar amb el sistema predictiu. A través d'una interfície neta i senzilla es pot utilitzar el sistema predictiu i visualitzar els resultats obtinguts així com un seguit de dades addicionals sobre les tasques realitzades (vocabulari generat, seqüències similars trobades, etc.) que serveixen de marc demostratiu dels mètodes de recerca.

Respecte el plantejament inicial de recerca:
"*Estudiar la utilització de tècniques de Machine Learning sobre dades històriques de pacients per a la detecció i identificació de seqüències en les corbes de glucosa de pacients amb diabetis.*"

analitzant els resultats obtinguts i mostrats en l'apartat 10, queda clar que la precisió del sistema no és suficientment bona com per determinar que es classifiquen correctament les seqüències de sessions dels pacients.

Aquest fet pot ser degut a dos raons (o a una combinació d'elles):

- Realment no existeix una correlació entre les corbes de glucosa que impliqui que la aparició d'una comportí l'aparició de la següent. És a dir, que una corba de glucosa i l'anterior son independents entre elles i no segueixen cap patró.
- Incompletesa de les dades: durant el desenvolupament el projecte, s'ha detectat clarament que les dades amb les que es treballava eren força incompletes (es podia donar el cas que en un dia hi haguessin la mitat de lectures que en el dia següent, o fins i tot dies sense lectures), desbalancejades (hi havia moltes més sessions classificades com a hiperglucèmia que com a hipoglucèmia), i fins i tot incoherents (hi havia dies que un pacient realitzava fins a 5 àpats i dies en que per exemple només 2). Tots aquests aspectes s'han intentat solucionar durant el preprocés de les dades però tot i això afecta al sistema final.

  En altres projectes en que s'ha treballat amb les mateixes dades també s'han trobat amb aquest problema [29].

Tot i això, considero que el sistema desenvolupat podria ser un bon punt de partida a partir del qual continuar treballant en aquesta línia. Tot i que els resultats no permeten dir que es tracte d'un sistema que sap predir amb seguretat l'estat futur d'un pacient, han posat de manifest els límits de les dades actuals disponibles.

Finalment, com a valoració personal del treball, penso que ha estat una experiència molt constructiva, que m'ha permès adquirir molts coneixements en l'àmbit del Machine Learning i m'ha donat la oportunitat de veure com es treballa en un grup de recerca.

12 Treball Futur

Com a treball futur, hi ha diverses tasques i millores a realitzar tan per al sistema predictiu com per a l'aplicació client-servidor:

- Millores en seguretat en l'aplicació client-servidor. Les dades que s'envien del client el servidor són dades de caràcter protegit. Actualment el sistema desenvolupat envia aquestes dades sense prendre cap mesura addicional de seguretat com podria ser la encriptació d'aquestes dades. Si finalment s'implantés el sistema proposat en aquest treball, s'hauria de millorar la seguretat en l'enviament de les dades.
- Migrar a un sistema de gestió de bases de dades més potent. Tal i com s'ha explicat al llarg del present treball, la elecció de SqLitle es basava en gran part a la seva senzillesa. Aquesta senzillesa té un cost en el rendiment, que és més baix que en altres sistemes com Postgres o Mysql. Fins el moment, la relativa poca quantitat de dades amb les que es treballava no feia necessari aquesta migració, però si en un futur s'introduïssin més dades en el sistema acabaria sent necessari.
- Personalització dels algorismes de classificació. Per realitzar les prediccions s'han utilitzat els algorismes de KNN i SVM proporcionats en la llibreria Sckit-Learning utilitzant la mateixa configuració per a tots els pacients. Es podria personalitzar per a cada pacient una configuració diferent que s'ajustés més a les seves característiques en línia amb la medicina personalitzada.

13 Bibliografia

[1] NIDDK. Types of Diabetes (31/07/2016).
https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes
(Consultada el: 01/06/2017)

[2] NIDDK. Hypoglycemia (08/2016).
https://www.niddk.nih.gov/health-information/diabetes/overview/preventing-problems/low-blood-glucose-hypoglycemia
(Consultada el: 01/06/2017)

[3] Cristina García Cambronero, Irene Gómez Moreno. Algoritmos de aprendizage.
http://www.it.uc3m.es/jvillena/irc/practicas/08-09/06.pdf
(Consultada el: 01/06/2017)

[4] Tan, Steinbach, Kumar. The k.means algorithm.
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
(Consultada el: 01/06/2017)

[5] Fermin Pitol. KNN, el algoritmo del vecino mas cercano (22/3/2014).
http://ferminpitol.blogspot.com.es/2014/03/k-nn-k-nearest-neighbor-el-algoritmo.html
(Consultada el: 01/06/2017)

[6] Sunil Ray. Understanding Support Vector Machine algorithm (06/10/2015).
https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/
(Consultada el: 01/06/2017)

[7] Enrique J.Carmona Suárez. Tutorial sobre Máquinas de Vectores Soporte (11/07/2014).
http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdf
(Consultada el: 01/06/2017)

[8] Ernest Valveny, Jordi Gonzàlez Sabaté, Ramon Baldrich Caselles. Bag of Words- Support Vector Machine.
https://es.coursera.org/learn/clasificacion-imagenes/lecture/52lRD/support-vector-machines-svm-conceptos-basicos
(Consultada el: 01/06/2017)

[9] https://docs.python.org/3.5
(Consultada el: 01/06/2017)

[10] http://scikit-learn.org/stable/documentation.html
(Consultada el: 01/06/2017)

[11] https://www.sqlite.org/docs.html
(Consultada el: 01/06/2017)

[12] https://docs.python.org/2/library/sqlite3.html
(Consultada el: 01/06/2017)

[13] http://pandas.pydata.org
(Consultada el: 01/06/2017)

[14] https://lmfit.github.io/lmfit-py
(Consultada el: 01/06/2017)

[15] https://plot.ly/python/
(Consultada el: 01/06/2017)

[16] http://flask.pocoo.org/docs/0.12/
(Consultada el: 01/06/2017)

[17] Andrearrs. Flask: mnimalismo para el desarrollo web en Python (13/08/2014).
https://hipertextual.com/archivo/2014/08/flask-python/
(Consultada el: 01/06/2017)

[18] Jay. Python Flask JQuery Ajax POST (19/07/2015).
http://codehandbook.org/python-flask-jquery-ajax-post/
(Consultada el: 01/06/2017)

[19] https://bootstrapdocs.com/v3.3.6/docs/
(Consultada el: 01/06/2017)

[20] Kenneth Jensen (IBM). K-fold Cross-validation in IBM SPSS Modeler (02/03/2013).
https://developer.ibm.com/predictiveanalytics/2016/03/02/k-fold-cross-validation-ibm-spss-modeler
(Consultada el: 01/06/2017)

[21] Zeyi Wen, Bin Li, Rao Kotagiri, Jian Chen, Yawen Chen, Rui Zhang. Improving Efficiency of SVM k-fold Cross-validation by Alpha Seeding (23/11/2016)

[22] Zecchin, C.: Online Glucose Prediction in Type 1 Diabetes by Neural Network Models. Universit degli Studi di Padova (2014)

[23] Brun, J.F., Fedou, C., and Mercier, J.: Postprandial reactive hypoglycemia. Diabetes & Metabolism 26(5), 337–351 (2000)

[24] García-Jaramillo, M., Calm, R., Bondia, J., Tarín, C., and Vehí, J. (2009). Computing the risk of postprandial hypo- and hyperglycemia in type 1 diabetes mellitus considering intrapatient variability and other sources of uncertainty. Journal of Diabetes Science and Technology, 3(4):895-902.

[25] Jason Browniee. Multi-Class Classification Tutorial with the Keras Deep Learning Library (02/06/2016).
http://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
(Consultada el: 01/06/2017)

[26] Hanna M Wallach., Topic modelling: beyond bag-of-words, 2006

[27] J. Yang et al., Evaluating bag-of-visual-words representations in scene classification, 2007.

[28] Albert Pla, Natalia Mordvanyuk, Beatriz Lopez, Marco Raaben, Taco J. Blokhuis, Herman R. Holstlag. Bag-of-steps: Predicting Lower-limb Fracture Rehabilitation Length.

[29] Fabien Dubosson, Natalia Mordanyuk, Beatriz Lóopez2, and Michael Schumacher. Prediction of postprandial hypoglycemias from insulin intakes and carbohydrates: analysis and comparison between real and simulated datasets. 2nd Workshop on Artificial Intelligence for Diabetes, MIE, Vienna, 2017

14 Annexos

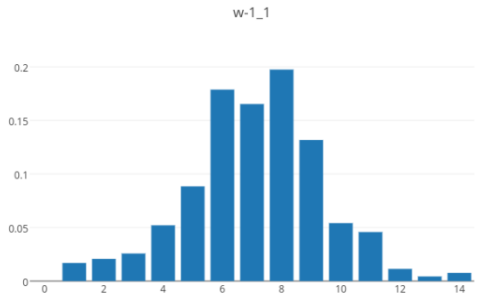
Els annexos d'aquesta memòria són:

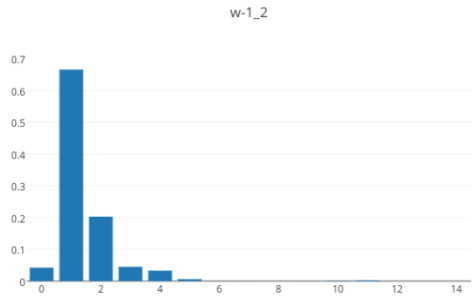
- Vocabularis de de mida 15 i 45 generats a partir dels histogrames de les corves de glucosa.
- Manual d'instal·lació i execució
- Codi Font

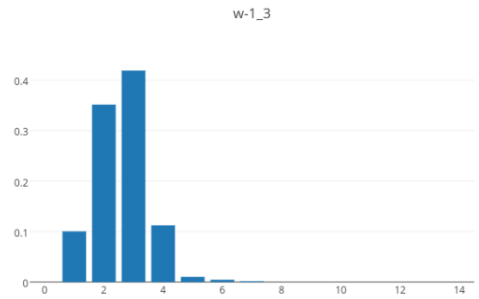
14.1 Vocabularis Generats

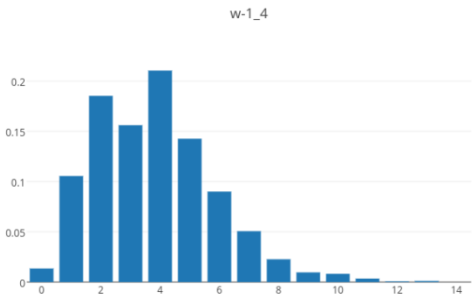
Mida 15

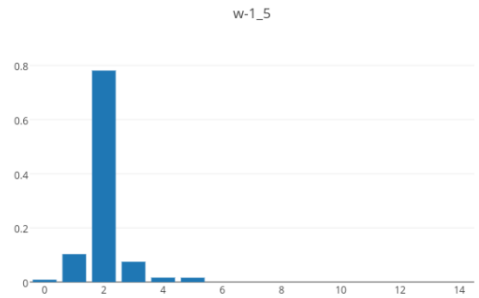
Paraules Hipoglucèmia



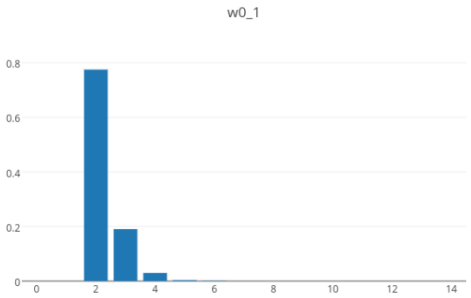


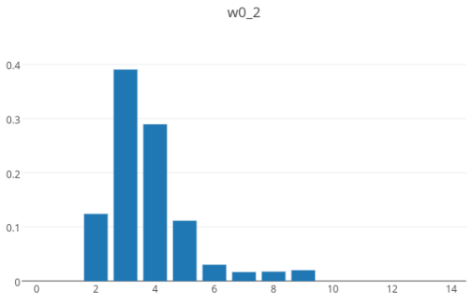


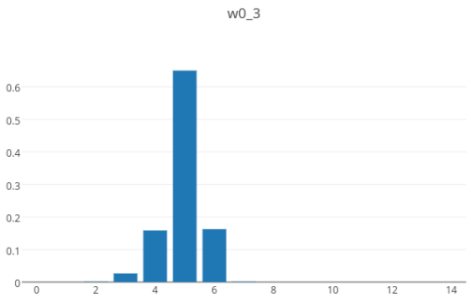


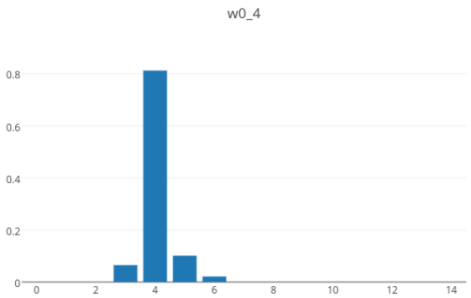


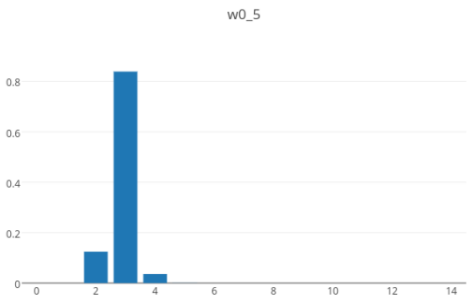
Paraules Normoglucèmia



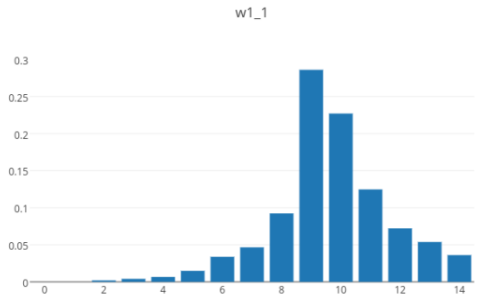


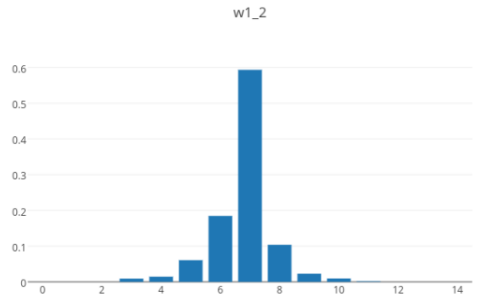


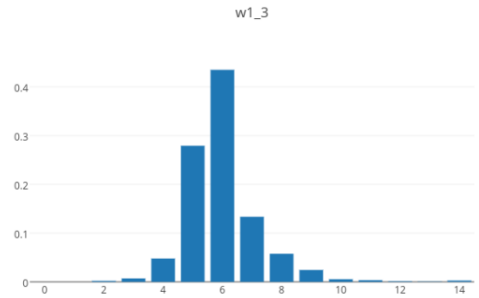


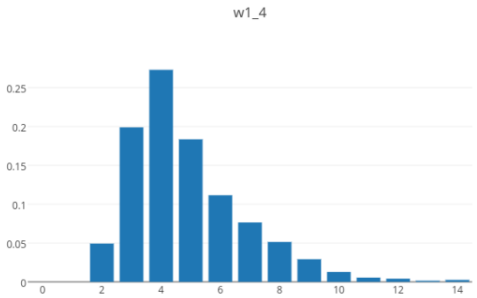


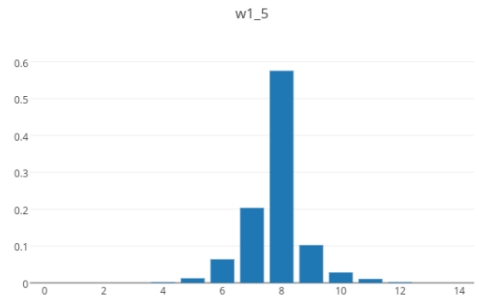
Paraules Hiperglucèmia





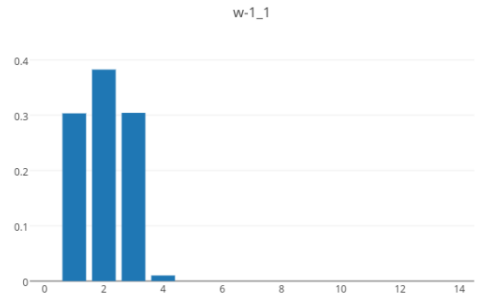


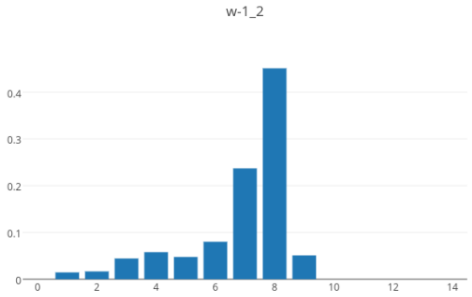


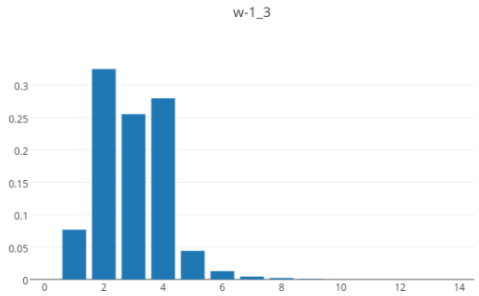


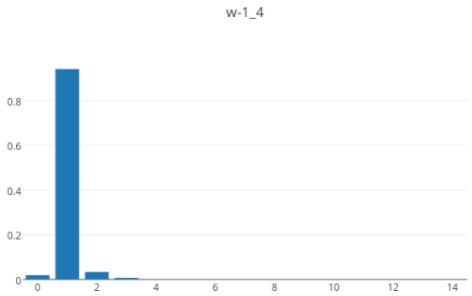
Mida 45

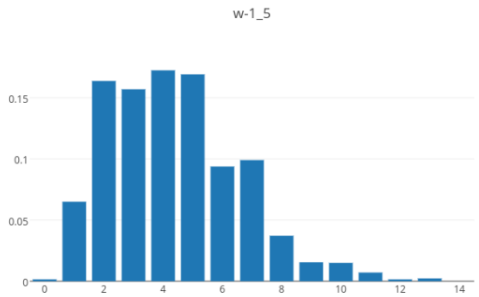
Paraules Hipoglucèmia

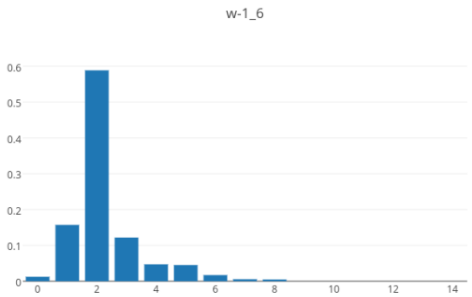


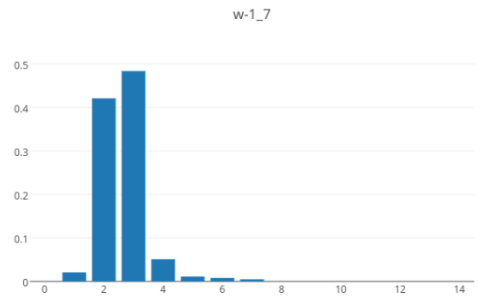


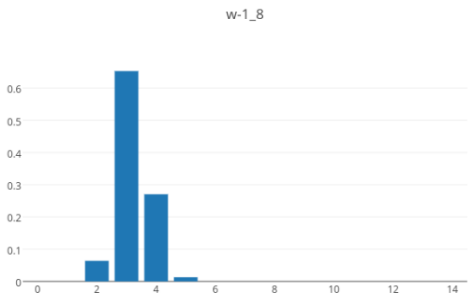




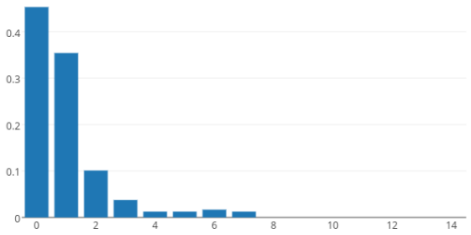




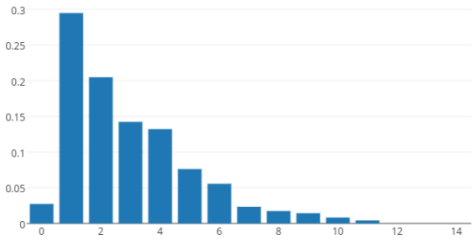




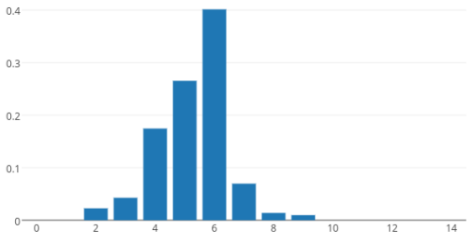
w-1_9



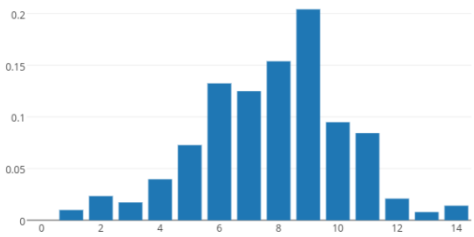
w-1_10



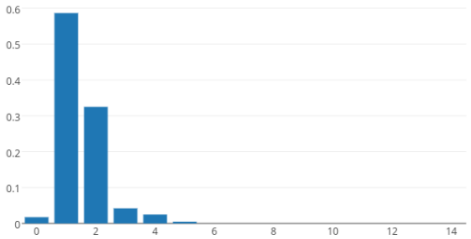
w-1_11



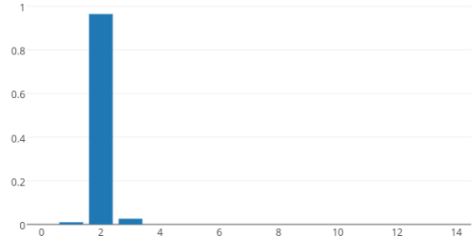
w-1_12



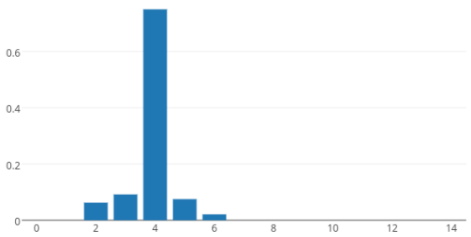
w-1_13



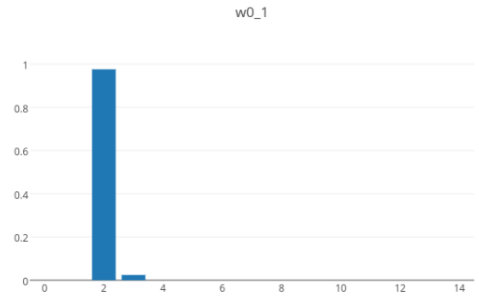
w-1_14

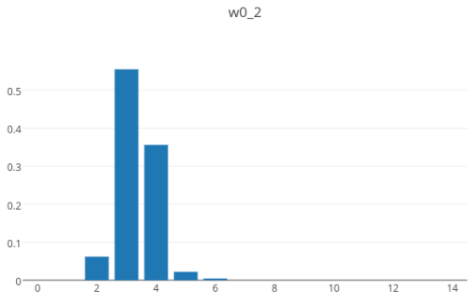


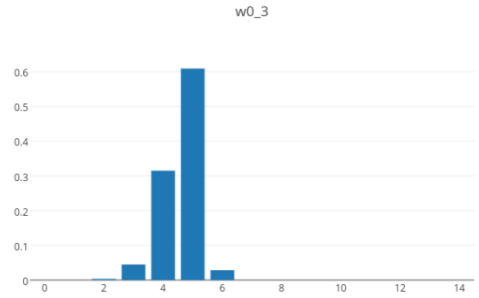
w-1_15

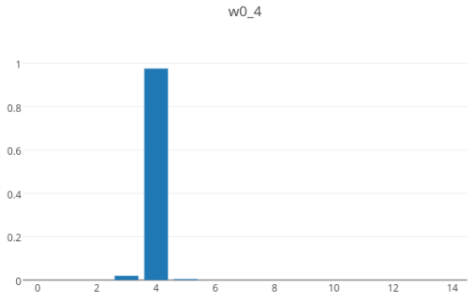


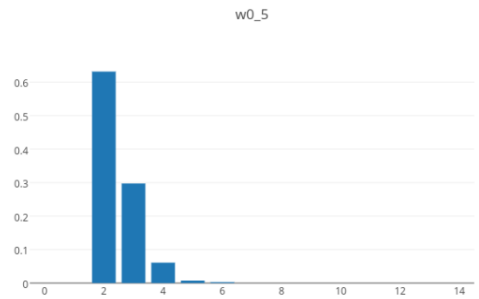
Paraules Normoglucèmia

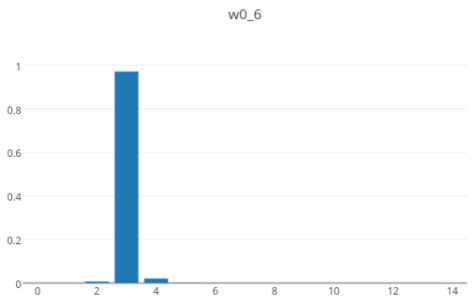


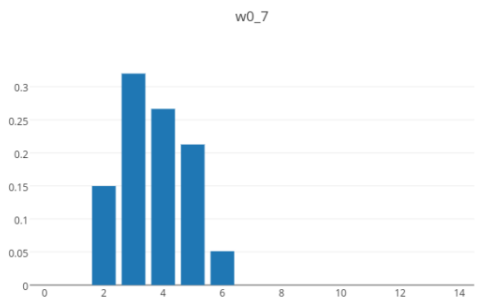


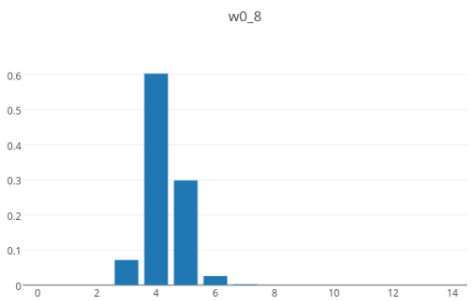




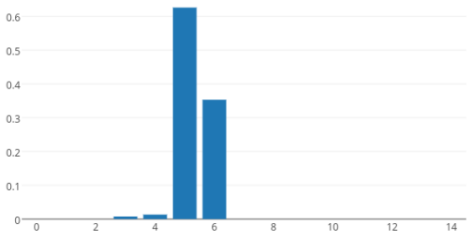




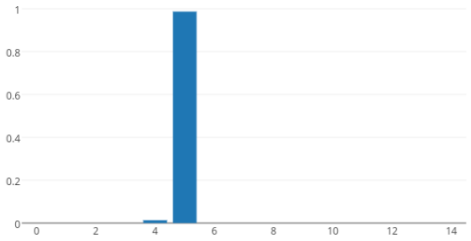




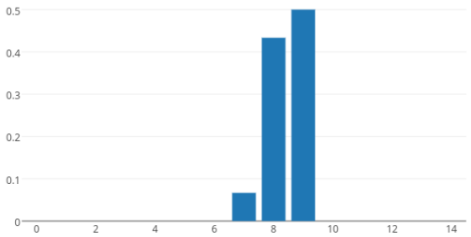
w0_9



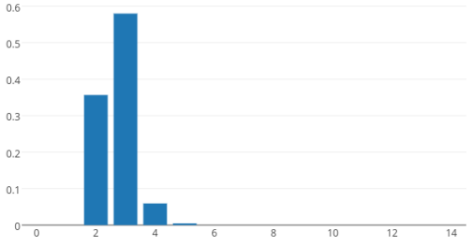
w0_10



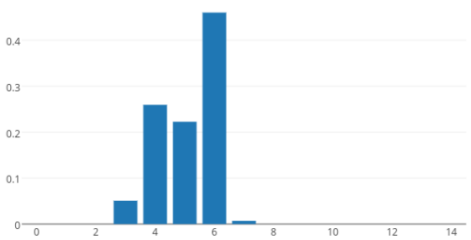
w0_11



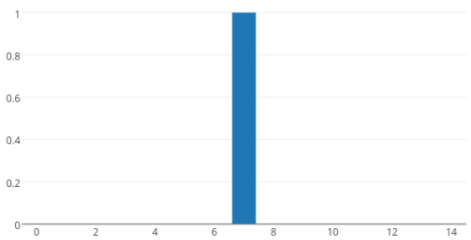
w0_12



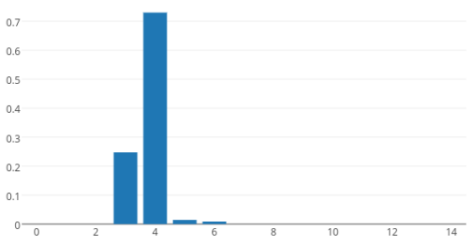
w0_13



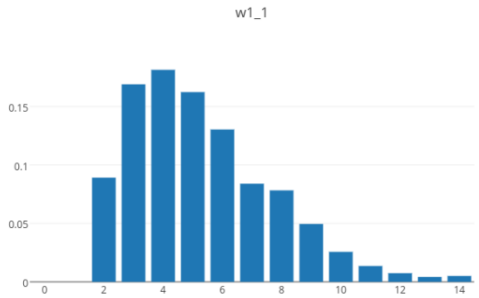
w0_14

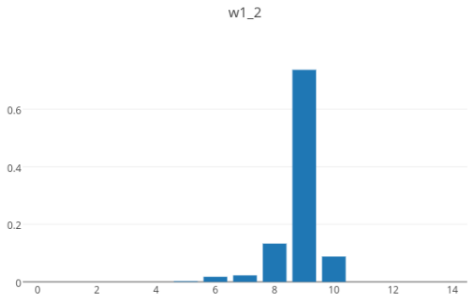


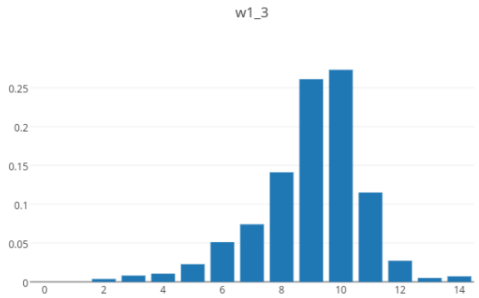
w0_15

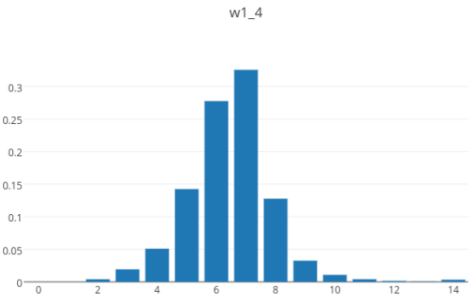


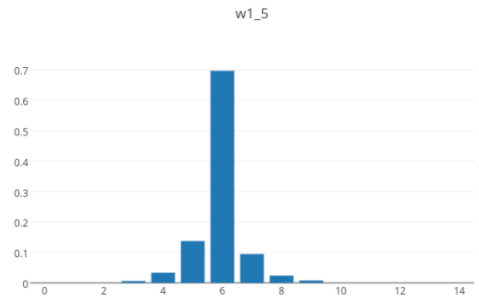
Paraules Hiperglucèmia

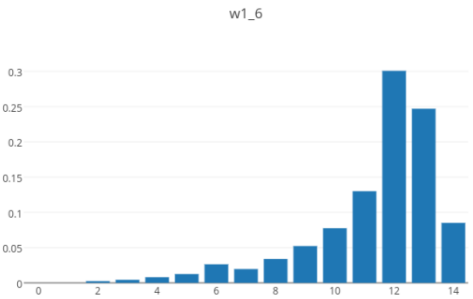


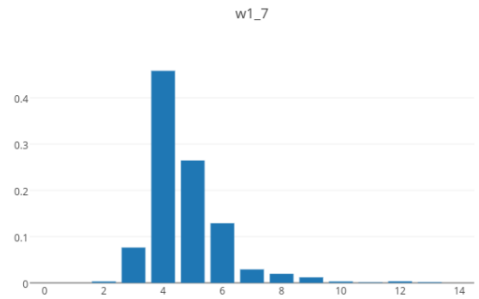


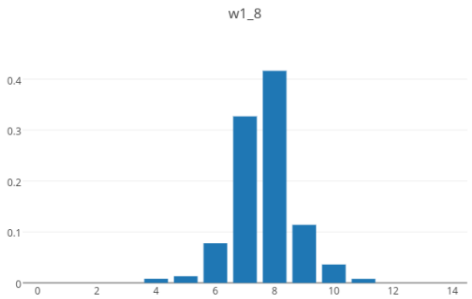




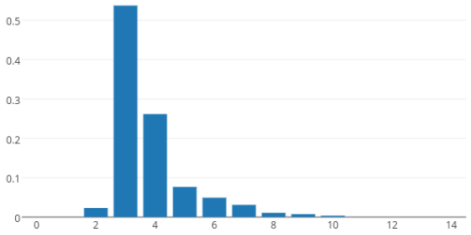




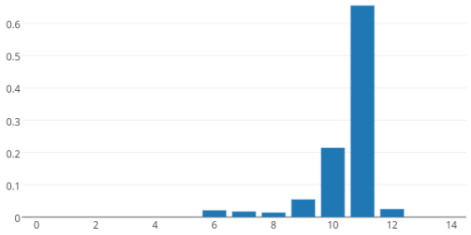




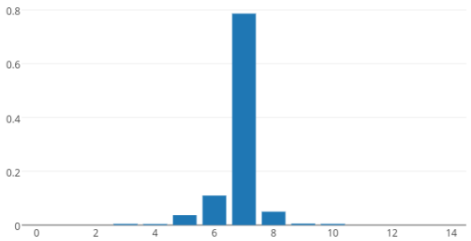
w1_9



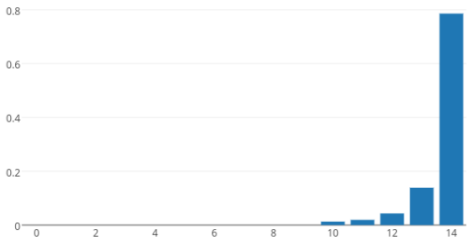
w1_10



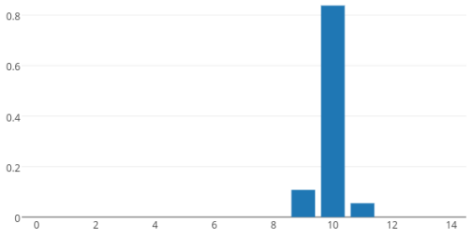
w1_11



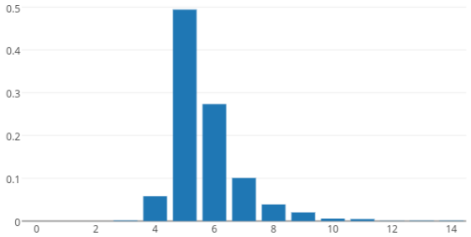
w1_12



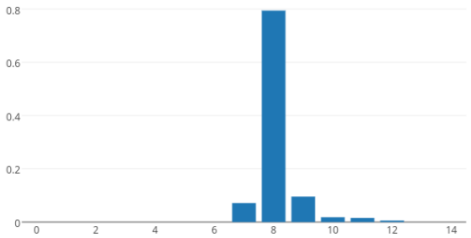
w1_13



w1_14



w1_15



14.2 Manual d'Instal·lació

És necessari tenir instal·lat python 3.5 i el seu sistema de gestió de paquets: pip.

Instal·lació

Es considera que es consta de la següent estructura de directoris i fitxers:

- WebCGM/
    - WebCGM.py
    - requirements.txt
    - PredictCGM/
        - CGMLoader.py
        - CGMPredicter.py
        - CGMTester.py
    - templates/
        - final_view.html
        - home.html
        - index.html
        - sequence_view.html
        - session_view.html
    - uploads/
    - static/
        - css/
            - bootstrap.min.css
            - bootstrap.min.css
        - js/
            - bootstrap.min.js
            - jquery.js

Tots els fitxers mostrats han estat desenvolupats en aquest projecte i es proporcionen en l'annex 14.3 a excepció dels corresponents a bootstrap i jquery de la carpeta "static". Aquests últims es poden obtenir a les respectives pàgines oficials (http://getbootstrap.com i https://jquery.com)

Un cop es té la estructura de fitxers tal i com s'ha mostrat, s'ha de seguir els següents passos per completar la instal·lació:

1. Situar-se al directori del projecte "WebCGM/".
2. Obrir un terminal i executar la commanda: ***pip install –r requirements.txt***

Aquesta última commanda instal·larà tots els paquets de python necessaris per al correcte funcionament dels sistemes desenvolupats en aquest treball.

Execució

Per iniciar el servidor web només s'ha de executar la commanda des del directori WebCGM:

python WebCGM.py <args>

Si la versió de python que es té instal·lada per defecte no és la 3.5, en comptes de "python" s'haurà de utilitzar "python3.5". Per una descripció detallada dels possibles paràmetres i el seu significat s'ha d'executar:

python WebCGM.py --help

Es mostra el següent:

```
usage: WebCGM.py [-h] [-ht HOST] [-p PORT] [-nf NEW_DB_FROM_FILE]
                 [-db DATABASE]
                 [-pd {KNNPredicter,SVMPredicter,SVM_KNN_Predicter}]
                 [-s SEQUENCES_LEN] [-v VOCABULAY_SIZE]

optional arguments:
  -h, --help            show this help message and exit
  -ht HOST, --host HOST
                        The hostname to listen on. Set this to '0.0.0.0' to
                        have the server available externally as well. Defaults
                        to '127.0.0.1'.
  -p PORT, --port PORT  Port where the server will run. By default it's 5000.
  -nf NEW_DB_FROM_FILE, --new_db_from_file NEW_DB_FROM_FILE
                        Name of file with information to create the new
                        database. It must be in uploads folder.
  -db DATABASE, --database DATABASE
                        Database name. If '--new_db_from_file' is passed, it's
                        the name of the new database. Otherwise it's the name
                        of the existing database that will be used. This
                        database must be in 'uploads' directory. By default
                        it's 'cgm_database.db'.
  -pd {KNNPredicter,SVMPredicter,SVM_KNN_Predicter}, --predicter {KNNPredicter,SVMPredicter,SVM_KNN_Predicter}
                        Predicter name. It must be 'KNNPredicter',
                        'SVMPredicter' or 'SVM_KNN_Predicter'. By default it's
                        'SVMPredicter'.
  -s SEQUENCES_LEN, --sequences_len SEQUENCES_LEN
                        Lenght of the sequences used in predictions.
  -v VOCABULAY_SIZE, --vocabulay_size VOCABULAY_SIZE
                        Size of the vocabularies of glucose histograms used in
                        predictions.
```

14.3 Codi Font

Es proporciona el codi font dels fitxers:

- WebCGM.py
- CGMLoader.py
- CGMPredicter.py
- CGMTester.py
- final_view.html
- home.html
- index.html
- sequence_view.html
- session_view.html
- requirements.txt

```
import os
from flask import Flask, request, render_template, send_from_directory, abort, json, make_response, Response, jsonify
from werkzeug.utils import secure_filename
from PredictCGM.CGMLoader import Loader
from datetime import datetime
import argparse


class CGMServer(object):

    # Prepare the server
    app = Flask(__name__)
    # Init vars
    loader = None
    predicter_name = None
    voc_size = None
    seq_len = None
    predicter_dict = {}

    def __init__(self, host, port, loader, predicter_name="SVMPredicter", voc_size=15, seq_len=3):
        # Create uploads foler
        if not os.path.exists('./uploads/'):
            os.mkdir('./uploads/')
        CGMServer.app.config['UPLOAD_FOLDER'] = './uploads/'

        # set loader
        CGMServer.loader = loader
        print("Conected to {0}".format(self.loader.db_connection))

        # predicter options
        CGMServer.predicter_name = predicter_name
        CGMServer.voc_size = voc_size
        CGMServer.seq_len = seq_len
        CGMServer.predicter_dict = {}

        # Start the server
        CGMServer.app.run(host=host, port=port)

    @staticmethod
    @app.route('/', methods=['GET'])
    def init():
        try:
            pacients = CGMServer.loader.get_all_pacients()
            date = datetime.now().strftime("%Y-%m-%d %H:%M")
            params = {'pacients': pacients, 'time': date}
            return render_template('home.html', params=params)
        except Exception:
            # return index.html if database can not be opened
            return render_template('index.html')

    @staticmethod
    @app.route('/file', methods=['POST'])
    def upload_file():
        try:
            if not request.files.get('arxiu', False):
                return json.dumps({'success': True}), 200, {'ContentType': 'application/json'}
            file = request.files['arxiu']
            if file.filename != '':
                filename = secure_filename(file.filename)
                file.save(os.path.join(CGMServer.app.config['UPLOAD_FOLDER'], filename))
                return json.dumps({'success': True}), 200, {'ContentType': 'application/json'}
        except Exception as e:
            return jsonify(message="Select a correct file."), 500

    @staticmethod
    @app.route('/session', methods=['POST'])
    def create_session():
        try:
            # Get and validate data of new session
            elems = request.json
            datetime.strptime(elems['date'], "%Y-%m-%d %H:%M")
            try: insuline = float(elems['insulin'])
            except: raise Exception("Wrong format of 'Insulin'.")
            assert insuline > 0, "Insulin must be greater than 0."
            try: carboh = float(elems['carboh'])
            except: raise Exception("Wrong format of 'Carbohydrates'.")
            assert carboh > 0, "Carbohydrates must be greater than 0."
            try: glucose = float(elems['glucose'])
            except: raise Exception("Wrong format of 'Glucose'.")
            assert glucose > 0, "Glucose must be greater than 0."
            data, hora = elems['date'].split(" ")
            data = "{0}/{1}/{2}".format(data.split("-")[2], data.split("-")[1], data.split("-")[0])
            # Create the id assigned to the new session
            id = "{0}_{1}_{2}".format(elems['pacient'], data, hora)
            # Create list with all the data of the new session
            session_info = [[
                id,
                elems['pacient'],
                elems['date'],
                carboh,
                insuline,
                0 if elems['exerciseBf'] == 'No' else 1,
                0 if elems['exerciseAf'] == 'No' else 1,
                0 if elems['alcohol'] == 'No' else 1,
                CGMServer.loader.get_tipus_apat(hora),
                glucose,
            ]]
            # Load info of other sessions from file
            if elems['fileToUpload']:
                filename = elems['fileToUpload']
                filename = filename.split("\\")[-1]
```

```
                CGMServer.loader.load_data_from_csv(filename)
                CGMServer.loader.generate_fdps(normalized=True)
            CGMServer.loader.commit()
            CGMServer.loader.add_new_sessions(session_info)
            CGMServer.loader.commit()
            return json.dumps({'success': True, 'data': id}), 200, {'ContentType': 'application/json'}
        except Exception as e:
            CGMServer.loader.rollback()
            return jsonify(message=str(e)), 500

    @staticmethod
    @app.route('/session/<string:id>', methods=['GET'])
    def get_session(id):
        pacient, data, hora = id.split("_")
        data = "{0}/{1}/{2}".format(data.split("-")[2], data.split("-")[1], data.split("-")[0])
        session_id = "{0}_{1}_{2}".format(pacient, data, hora)
        ant_sessions = CGMServer.loader.get_ant_sessions(session_id)
        image_div = CGMServer.loader.draw_sessions(ant_sessions)
        params = {'image': image_div, 'id': session_id}
        return render_template('session_view.html', params=params)

    @staticmethod
    @app.route('/predicter', methods=['POST'])
    def create_predicter():
        session_to_predict = request.form['session_id']
        if session_to_predict is None:
            return abort(500)
        from PredictCGM import CGMPredict as pred_module
        PredicterObj = getattr(pred_module, CGMServer.predicter_name)
        predicter = PredicterObj(loader=CGMServer.loader, seq_len=CGMServer.seq_len, words_num=CGMServer.voc_size)
        imatges = CGMServer.loader.draw_sequence(session_to_predict)
        all_words = CGMServer.loader.draw_vocabulary()
        types = CGMServer.loader.get_labels_of_sequence(session_to_predict)
        sequences_data = [(imatges[i], types[i]) for i in range(len(imatges))]
        CGMServer.predicter_dict.update({session_to_predict: predicter})
        params = {
            'sequences_data': sequences_data,
            'route': "/predicter/"+session_to_predict.replace("/", "+"),
            'vocabulary': all_words
        }
        return render_template('sequence_view.html', params=params)

    @staticmethod
    @app.route('/predicter/<string:id>', methods=['GET'])
    def show_result(id):
        session_to_predict = id.replace("+", "/")
        predicter = CGMServer.predicter_dict[session_to_predict]
        res_info = predicter.predict([session_to_predict])

        label = res_info[0]
        word = res_info[2][0][0]
        lconfidence = max(res_info[1][0])
        wconfidence = max(res_info[3][0][0])

        word_img = CGMServer.loader.draw_word(word, session_to_predict)

        all_words = CGMServer.loader.draw_vocabulary()

        seq = CGMServer.loader.draw_sequence(session_to_predict, names=False)
        types = CGMServer.loader.get_labels_of_sequence(session_to_predict)
        sequences_data = [(seq[i], types[i]) for i in range(len(seq))]
        word_img2 = CGMServer.loader.draw_word(word)
        sequences_data.append((word_img2, label))

        seqs_final = []
        similar_seq = predicter.get_similar_models(session_to_predict, 5)
        for s, c in similar_seq:
            words = CGMServer.loader.get_words_of_sequence(s)
            words = str(words).replace("'", "")[1:-1]
            seqs_final.append((words, c))
        params = {
            'res': word,
            'label': label,
            'lconfidence': lconfidence,
            'wconfidence': wconfidence,
            'result_img': word_img,
            'sequences_data':sequences_data,
            'vocabulary': all_words,
            'similar_seq': seqs_final
        }
        CGMServer.predicter_dict[session_to_predict] = None
        CGMServer.loader.delete_session(session_to_predict)
        return render_template('final_view.html', params=params)

    @staticmethod
    @app.route('/uploads/<filename>')
    def uploaded_file(filename):
        return send_from_directory(CGMServer.app.config['UPLOAD_FOLDER'], filename)


def main(args):
    if args.new_db_from_file:
        # Create a new database using the information of given file
        print("Creating new database...")
        loader = Loader(db_name=args.database, new_db=True, resources_path="./uploads/")
        print("Loading data from {0}".format(args.new_db_from_file))
        loader.load_data_from_csv(args.new_db_from_file)
        loader.generate_fdps(normalized=True)
    else:
        # Try to open a database from "uploads" folder
        loader = Loader(db_name=args.database, new_db=False, resources_path="./uploads/")

    CGMServer(args.host, args.port, loader, args.predicter, args.vocabulay_size, args.sequences_len)
```

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-ht", "--host", default="127.0.0.1",
                        help="The hostname to listen on. Set this to '0.0.0.0' to have the server available externally as well. Defaults
    parser.add_argument("-p", "--port", default=5000, type=int,
                        help="Port where the server will run. By default it's 5000.")
    parser.add_argument("-nf", "--new_db_from_file",
                        help="Name of file with information to create the new database. It must be in uploads folder.")
    parser.add_argument("-db", "--database", default='cgm_database.db',
                        help="Database name. If '--new_db_from_file' is passed, it's the name of the new database. Otherwise it's the nam
    parser.add_argument("-pd", "--predicter", default='SVMPredicter',
                        choices=['KNNPredicter', 'SVMPredicter', 'SVM_KNN_Predicter'],
                        help="Predicter name. It must be  'KNNPredicter', 'SVMPredicter' or 'SVM_KNN_Predicter'. By default it's 'SVMPred
    parser.add_argument("-s", "--sequences_len", default=3, type=int,
                        help="Lenght of the sequences used in predictions. Default is 3.")
    parser.add_argument("-v", "--vocabulay_size", default=5, type=int,
                        help="Size of the vocabularies of glucose histograms used in predictions. Default is 5.")
    main(parser.parse_args())
```

```
import csv
import pandas as pd
from datetime import datetime, timedelta
from matplotlib import pyplot
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
from sklearn.cluster import KMeans
import sqlite3
import os
from tempfile import NamedTemporaryFile
import plotly
import plotly.graph_objs as go
import lmfit.models as mdl
from sklearn import preprocessing


class Loader(object):
    """
        This class is used to extract the data about the sessions of the patients from a .csv file and store it a database.

        It has all the methods related with data extraction, normalization and plotting.

        Provides the interface of the method:
            * generate_models
            * get_models
    """

    def __init__(self, db_name='default_cgm_database.db', new_db=False, resources_path="./"):
        """
        Creates or connect to a database.

        :param db_name: name of the new database or of the database used if 'new_db' is False
        :param new_db: if True, create a new database instead of connecting to the one with 'db_name'
        :param resources_path: path where the files will be placed and searched
        """
        self.min_time = 90
        self.max_time = 390

        self.resources_path = resources_path
        self.path = self.resources_path + db_name
        if not os.path.exists(self.resources_path):
            raise Exception("Resources directory doesn't exist: {0}".format(self.resources_path))
        # Time between the lectures of glucose will be taken
        self.db_connection = None
        self.cursor = None

        # Round all lectues to intervals of round_val
        self.round_val = 0.5

        if new_db:
            self._create_db_data()
        else:
            self._connect_database()

        # Default generator
        self.mgen = SequenceModels(WordsModels(SimpleModels(loader=self)))

    def _connect_database(self):
        """Set the cursor and open a db_connection."""
        self.db_connection = sqlite3.connect(self.path)
        self.cursor = self.db_connection.cursor()

    def commit(self):
        self.db_connection.commit()

    def rollback(self):
        self.db_connection.rollback()

    def _create_db_data(self):
        """Creates a new database and its tables. If a database existed, it is removed before creating the new one."""
        try:
            os.remove(self.path)
            print("Base de dades previa borrada")
        except: pass
        # Create new DB
        self.db_connection = sqlite3.connect(self.path)
        self.cursor = self.db_connection.cursor()
        self._create_users_table()
        self._create_sessions_table()
        self._create_lectures_cgm_table()
        self._create_words_table()
        self._create_seq_table()

    def _create_users_table(self):
        """Creates the table of Users."""
        self.cursor.execute(
            "CREATE TABLE Pacients "
            "(id TEXT PRIMARY KEY, edat INTEGER, sexe INTEGER, lower_target FLOAT, upper_target FLOAT);"
        )

    def _create_sessions_table(self):
        """Creates the table of Sessions."""
        self.cursor.execute(
            "CREATE TABLE Sessions "
            "(id TEXT PRIMARY KEY, pacient_id TEXT,  date TEXT, carboh FLOAT , insulin FLOAT , exerciseBf INTEGER, "
            "exerciseAf INTEGER, alcohol INTEGER, tipus_apat INTEGER, glucose FLOAT, word_id TEXT, label INTEGER, "
            "FOREIGN KEY(pacient_id) REFERENCES Pacients(id));"
        )

    def _create_lectures_cgm_table(self):
        """Creates the table of LecturesCGM."""
```

```
        self.cursor.execute(
            "CREATE TABLE LecturesCGM "
            "(id INTEGER PRIMARY KEY AUTOINCREMENT, sessio_id TEXT, lectura FLOAT, mtimestamp INTEGER , "
            "FOREIGN KEY(sessio_id) REFERENCES Sessions(id));"
        )
        self.cursor.execute(
            "CREATE TABLE CleanLecturesCGM "
            "(id INTEGER PRIMARY KEY AUTOINCREMENT, sessio_id TEXT, lectura FLOAT, "
            "FOREIGN KEY(sessio_id) REFERENCES Sessions(id));"
        )

    def _create_words_table(self):
        """Creates the table of Words."""
        self.cursor.execute("CREATE TABLE Words (id TEXT PRIMARY KEY, label INTEGER);")

    def _create_seq_table(self):
        """Creates the table of Sequences."""
        self.cursor.execute("CREATE TABLE Sequences (sessio_id TEXT PRIMARY KEY);")

    def load_data_from_csv(self, filename):
        """
        Load the data from 'filename' about patients and glucose lectures,
        make a preprocess and store it in the database
        """
        self._generate_cgm_and_labels(filename)
        self.clean_data()

    def _generate_cgm_and_labels(self, filename):
        """
        Adds new patiens, sessions and lecures from 'filename' to tables 'Pacients', 'Sessions' and 'LecturesCGM'.
        Sessions without lectures are dissmissed.
        Each row is the information about a meal.
        Structure of data in file is expected as:

                patient_id; sex (1/0); lowertarget; middletarget; uppertarget; age; meal date; meal hour; timestamp;
                    glucose; carbihidrats; insulin; exercice before (1/0); exercice after (1/0); alchool (1/0); lectures;

        where lectures have the format:
              val_lect1-date_lect1; val_lect2-date_lect2; ... ; val_lectN-date_lectN;
        """
        f = open('{0}/{1}'.format(self.resources_path, filename))
        reader = csv.reader(f, delimiter=';')
        for r in reader:
            use = False
            num = 0
            eid = ""
            gshape = []
            tshape = []
            res, min, max = 0, 0, 0
            time_hiper = 0
            session_info = []
            for elem in r:
                if num == 0:
                    eid = elem.replace(' ', '')
                elif num == 2:
                    min = float(elem)
                elif num == 4:
                    max = float(elem)
                elif num == 6:  # construim el id de la corva
                    aux = ''
                    for d in elem.split("/"):
                        if len(d) == 1:
                            d = '0{0}'.format(d)
                        aux += "{0}/".format(d)
                    eid = "{0}_{1}".format(eid, aux[:-1]).replace(' ', '')
                elif num == 7:
                    aux = ''
                    for d in elem.split(":"):
                        if len(d) == 1:
                            d = '0{0}'.format(d)
                        aux += "{0}:".format(d)
                    eid = "{0}_{1}".format(eid, aux[:-1]).replace(' ', '')
                    tipus_apat = self.get_tipus_apat(elem)
                elif num in [1, 5, 9, 10, 11, 12, 13, 14]:
                    session_info.append(float(elem))
                elif num > 14:
                    use = True
                    glevel, time_data = elem.replace(',', '.').split('-')
                    gdate = datetime.strptime(time_data, "%d/%m/%Y %H:%M:%S")
                    gshape.append(float(glevel))
                    tshape.append(gdate)

                    # Def. Hiper/Hipo 2
                    if gshape[-1] >= max:
                        time_hiper += 5
                    else:
                        time_hiper = 0
                    if gshape[-1] <= min:
                        res = -1
                    elif time_hiper >= 60 and res == 0:
                        res = 1

                    # Def. Hiper/Hipo 1
                    # if res == 0:
                    #     if gshape[-1] >= max:
                    #         res = 1
                    #     elif gshape[-1] <= min:
                    #         res = -1

                num += 1
            if use:
                session_info.append(tipus_apat)
                # Insert Pacient if it didn't exist
                try:
```

```
                self.cursor.execute(
                    "INSERT INTO Pacients VALUES (?, ?, ?, ?, ?)",
                    (eid.split('_')[0], session_info[1], session_info[0], min, max)
                )
            except: pass
            # Insert Sessions
            day = eid.split("_")[1]
            hour = eid.split("_")[2]
            date_session = "{0}-{1}-{2} {3}:00".format(
                day.split("/")[2], day.split("/")[1], day.split("/")[0], hour
            )
            self.cursor.execute(
                "INSERT INTO Sessions VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
                (eid, eid.split('_')[0], date_session, session_info[3], session_info[4], session_info[5],
                 session_info[6], session_info[7], session_info[8], session_info[2], None, res)
            )
            # Insert Lect. values
            min = 5
            for value in gshape:
                self.cursor.execute(
                    "INSERT INTO LecturesCGM (sessio_id, lectura, mtimestamp) VALUES (?, ?, ?)", (eid, value, min)
                )
                min += 5

def get_tipus_apat(self, data_str):
    hora = int(data_str.split(":")[0])
    if hora in [3, 4, 5, 6, 7, 8, 9, 10]:
        return 1  # 'E'
    elif hora in [11, 12, 13, 14, 15, 16, 17]:
        return 2  # 'D'
    elif hora in [18, 19, 20, 21, 22, 23, 24, 0, 1, 2]:
        return 3  # 'S'

def clean_data(self):
    """ Apply some methods to 'clean' and 'normalize' the glucose lectures stored in 'LecturesCGM':
            * filter data
            * Smooth
            * Standarize
            * Rescale
            * Round values to intervals of 0.025
        A new table 'CleanLecturesCGM' with the 'cleaned' lectures is created.
        Sessions without 'clean' lectures are deleted.
    """
    self._filter_cgm()
    self._smooth_cgm()
    # self._standarize_cgm()
    # self._rescale_cgm()
    self._round_lectures()

def _filter_cgm(self):
    """
    Create table 'CleanLecturesCGM' with the lectures found between min_time and max_time of each session.
    The sessions without lectures between min_time and max_time are deleted.
    """
    cgm_ts = pd.read_sql(
        "SELECT sessio_id, lectura FROM LecturesCGM WHERE mtimestamp >= {0} AND mtimestamp < {1};".
            format(self.min_time, self.max_time),
        self.db_connection
    )
    cgm_ts.to_sql("CleanLecturesCGM", self.db_connection, index=False, if_exists="replace")
    sessions = list(self.cursor.execute("SELECT sessio_id FROM CleanLecturesCGM GROUP BY sessio_id;").fetchall())
    sessions = str(sessions)[1:-1].replace("(", "").replace("),", "").replace(",)", "")
    self.cursor.execute("DELETE FROM Sessions WHERE id NOT IN ({0})".format(sessions))

def _round_lectures(self):
    """ Round all the values of lectures in CleanLecturesCGM to intervals of round_val """
    cgm_ts = pd.read_sql("SELECT sessio_id, lectura FROM CleanLecturesCGM;", self.db_connection)
    cgm_ts.lectura = cgm_ts.apply(lambda row: myround(row['lectura'], base=self.round_val), axis=1)
    cgm_ts.to_sql("CleanLecturesCGM", self.db_connection, index=False, if_exists="replace")
    self.min_lect = self.cursor.execute("SELECT min(lectura) FROM CleanLecturesCGM;").fetchone()[0]
    self.max_lect = self.cursor.execute("SELECT max(lectura) FROM CleanLecturesCGM;").fetchone()[0]

def _rescale_cgm(self):
    """
    Rescaling of the data of CleanLecturesCGM from the original range so that all values are within
    the range of 0 and 1.
    A value is rescaled as follows: (x - min) / (max - min)
    All the sessions are used in the rescale.
    """
    # load the dataset
    cgm_ts = pd.read_sql("SELECT sessio_id, lectura FROM CleanLecturesCGM;", self.db_connection)
    series = pd.Series(cgm_ts.lectura.values, index=cgm_ts.sessio_id.values)
    values = series.values
    values = values.reshape((len(values), 1))
    # train the normalization
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(values)
    self.rescaler= scaler
    # normalize the dataset
    normalized = scaler.transform(values)
    cgm_ts['lectura'] = normalized
    cgm_ts.to_sql("CleanLecturesCGM", self.db_connection, index=False, if_exists="replace")

def _standarize_cgm(self):
    """
    Standardizing the data of CleanLecturesCGM involves rescaling the distribution of values so that the mean of
    observed values is 0 and the standard deviation is 1.
    A value is standardized as follows:
        y = (x - mean) / standard_deviation
        mean = sum(x) / count(x)
        standard_deviation = sqrt( sum( (x - mean)^2 ) / count(x))
    All the sessions are used.
    """
    # load the dataset
```

```
    cgm_ts = pd.read_sql("SELECT sessio_id, lectura FROM CleanLecturesCGM;", self.db_connection)
    series = pd.Series(cgm_ts.lectura.values, index=cgm_ts.sessio_id.values)
    values = series.values
    values = values.reshape((len(values), 1))
    # train the normalization
    scaler = StandardScaler()
    scaler = scaler.fit(values)
    self.standarizer = scaler
    # normalize the dataset
    standarized = scaler.transform(values)
    cgm_ts['lectura'] = standarized
    cgm_ts.to_sql("CleanLecturesCGM", self.db_connection, index=False, if_exists="replace")

def _smooth_cgm(self):
    """
    Smooth the data of CleanLecturesCGM using a window with requested size.
    This method is based on the convolution of a scaled window with the signal.
    """
    cgm_ts = pd.read_sql("SELECT sessio_id, lectura FROM CleanLecturesCGM;", self.db_connection)
    for sessio in self.cursor.execute("SELECT id FROM Sessions;"):
        serie = pd.read_sql(
            "SELECT * FROM CleanLecturesCGM WHERE sessio_id = '{0}'".format(sessio[0]), self.db_connection
        )
        lectures = serie.values[:, -1]
        w = 11
        if len(serie) < 12:
            w = len(serie)-1
            if w % 2 == 0:
                w -= 1
        new = smooth(lectures, w)
        cgm_ts.ix[cgm_ts['sessio_id'] == sessio[0], -1] = new
    cgm_ts.to_sql("CleanLecturesCGM", self.db_connection, index=False, if_exists="replace")


def generate_fdps(self, session_ids=None, normalized=False):
    """
    :param session_ids: sessions whose FDP will be generated. If none is given, the FDPs of all sessions in
                        database will be generated
    :param normalized: if true, normalize the values of the fdp between 1 and 0
    """
    fdp_data = {}
    if session_ids is None:
        cgm_ts = pd.read_sql("SELECT sessio_id, lectura FROM CleanLecturesCGM;", self.db_connection)
        sessions = self.cursor.execute("SELECT id FROM Sessions WHERE label IS NOT NULL;").fetchall()
    else:
        session_ids = str(session_ids)[1:-1]
        cgm_ts = pd.read_sql(
            "SELECT sessio_id, lectura FROM CleanLecturesCGM WHERE sessio_id IN ({0});".format(session_ids),
            self.db_connection
        )
        sessions = self.cursor.execute("SELECT id FROM Sessions WHERE id IN ({0});".format(session_ids)).fetchall()

    values = [round(float(x), 5) for x in np.arange(self.min_lect, self.max_lect+self.round_val, self.round_val)]
    for session in sessions:
        h = cgm_ts[cgm_ts.sessio_id == session[0]].groupby(['sessio_id', 'lectura']).size()
        indexos = [round(float(x), 5) for x in h[session[0]].index]
        new_index = pd.Series({x: 0 for x in values if x not in indexos})
        fdp_values = h[session[0]].append(new_index).sort_index().tolist()
        if normalized:
            num = sum(fdp_values)
            fdp_values = [round(x / num, 5) for x in fdp_values]
        fdp_data[session[0]] = [session[0]] + fdp_values
    fdp_data = pd.DataFrame(list(fdp_data.values()), columns=['id'] + values)
    if session_ids is None:
        fdp_data.to_sql("SessionsFDP", self.db_connection, index=False, if_exists="replace")
    else:
        fdp_data.to_sql("SessionsFDP", self.db_connection, index=False, if_exists="append")

def add_new_sessions(self, sessions_data):
    """
    :param sessions_data: list of tuples (one for session) with all the information about each session:
            * id
            * pacient_id
            * date
            * carboh
            * insulin
            * exerciseBf
            * exerciseAf
            * alcohol
            * tipus_apat
            * glucose
    Creates new sessions and adds it to the table 'Sessions'
    """
    session_ids = []
    for session_info in sessions_data:
        session_ids.append(session_info[0])
        self.cursor.execute(
            "INSERT INTO Sessions VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
            (session_info[0], session_info[1], session_info[2], session_info[3], session_info[4], session_info[5],
             session_info[6], session_info[7], session_info[8], session_info[9], None, None)
        )
    return session_ids

def delete_session(self, session_id):
    try:
        self.cursor.execute("DELETE FROM Sessions WHERE id = '{0}'".format(session_id))
        self.cursor.execute("DELETE FROM SessionsNorm WHERE id = '{0}'".format(session_id))
        self.cursor.execute("DELETE FROM Sequences WHERE sessio_id = '{0}'".format(session_id)).fetchall()
        self.commit()
        print("deleted: {0}".format(session_id))
    except Exception as e:
        print(e)
        pass
```

```
@staticmethod
def get_session_fields(name="SessionsNorm", weight=1):
    return "{0}.tipus_apat*{1}, {0}.insulin*{1}, {0}.glucose*{1}, {0}.carboh*{1}, {0}.alcohol*{1}, {0}.exerciseBf*{1} ".format(name,

def _get_all_sessions(self, pacient=None):
    """ Returns all session ids. If a patient is given return only the sessions of the patient."""
    if pacient is None:
        return [info[0] for info in self.cursor.execute("SELECT id FROM Sessions ORDER BY id;").fetchall()]
    else:
        return [info[0] for info in self.cursor.execute("SELECT id FROM Sessions WHERE pacient_id = '{0}' "
                                                        "ORDER BY id;".format(pacient)).fetchall()]

def get_all_sessions(self, pacient=None, for_train=False):
    """ Returns all session ids. If a patient is given return only the sessions of the patient. """
    sessions = self._get_all_sessions(pacient)
    if for_train:
        try:
            train_sessions = [info[0] for info in
                              self.cursor.execute("SELECT sessio_id FROM Sequences AS Sq JOIN Sessions AS S "
                                                  "ON Sq.sessio_id=S.id WHERE S.word_id IS NOT NULL ").fetchall()]
            sessions = [x for x in sessions if x in train_sessions]
        except:
            return sessions
    return sessions

def get_all_pacients(self):
    """ Returns all patients ids"""
    return [info[0] for info in self.cursor.execute("SELECT id FROM Pacients;").fetchall()]

def get_session_labels(self, session_ids):
    """ Returns the labels of given sessions """
    if not isinstance(session_ids, list):
        session_ids = [session_ids]
    return [info[0] for info in
            self.cursor.execute("SELECT label FROM Sessions WHERE id IN ({0}) "
                                "ORDER BY id;".format(str(session_ids)[1:-1])).fetchall()]

def get_ant_sessions(self, session_id, num=3):
    """ Return session ids of the 'num' previous sessions to 'session_id'"""
    pacient, date = self.cursor.execute(
        "SELECT pacient_id, date FROM Sessions WHERE id='{0}';".format(session_id)
    ).fetchone()
    return list(reversed([x[0] for x in
                          self.cursor.execute("SELECT id FROM Sessions WHERE pacient_id='{0}' AND date < '{1}' "
                                              "ORDER BY date DESC LIMIT {2};".format(pacient, date, num)
                                              ).fetchall()]))

def get_labels_of_words(self, words_list):
    """ Return the labels of the words in words_list"""
    if not isinstance(words_list, list):
        words_list = [words_list]
    word_labels = dict([(x[0], x[1]) for x in self.cursor.execute(
        "SELECT id, label FROM Words WHERE id IN ({0}) ORDER BY id;".format(str(words_list)[1:-1])).fetchall()])
    labels = [word_labels[x] for x in words_list]
    return labels

def get_labels_of_sequence(self, session_id):
    """ Return labels of sessions of sequence of session_id """
    seq_names = [tup[1] for tup in self.cursor.execute('PRAGMA TABLE_INFO(Sequences)').fetchall() if
                 tup[1] not in ['sessio_id', 'next_label']]
    seq_names = str(seq_names).replace("'", "")[1:-1]
    seq_info = self.cursor.execute(
        "SELECT {1} FROM Sequences WHERE sessio_id='{0}'".format(session_id, seq_names)
    ).fetchone()
    return self.get_session_labels([x for x in seq_info])

def get_words_of_sequence(self, session_id):
    seq_names = [tup[1] for tup in self.cursor.execute('PRAGMA TABLE_INFO(Sequences)').fetchall() if
                 tup[1] not in ['sessio_id', 'next_label']]
    seq_names = str(seq_names).replace("'", "")[1:-1]
    seq_info = self.cursor.execute(
        "SELECT {1} FROM Sequences WHERE sessio_id='{0}'".format(session_id, seq_names)
    ).fetchone()
    return self.get_session_words([x for x in seq_info] + [session_id])

def get_session_words(self, session_ids):
    """ Returns the word of given sessions """
    if not isinstance(session_ids, list):
        session_ids = [session_ids]
    return [info[0] for info in
            self.cursor.execute("SELECT word_id FROM Sessions WHERE id IN ({0}) ORDER BY id;".format(
                str(session_ids)[1:-1])).fetchall()]

def draw_sessions(self, session_ids):
    """ Draw all the lectures and clean_lectures of session ids in a single and continuous plot"""
    pacient = ''
    min = 0
    max = 0
    tdates = []
    tcgm = []
    filtered_plots = []
    y_marks = []
    x_marks = []
    for session_id in session_ids:
        cgm_filtered = pd.read_sql(
            "SELECT lectura FROM LecturesCGM WHERE sessio_id = '{0}' AND mtimestamp >= {1} AND mtimestamp < {2};"
                .format(session_id, self.min_time, self.max_time),  self.db_connection
        )
        cgm = pd.read_sql(
            "SELECT lectura FROM LecturesCGM WHERE sessio_id = '{0}';".format(session_id), self.db_connection
        )
        pacient = session_id.split("_")[0]
        min, max = self.cursor.execute(
            "SELECT lower_target, upper_target FROM Pacients WHERE id = '{0}';".format(pacient)
```

```
        ).fetchone()
        ini_date = datetime.strptime(session_id.split("_")[1]+" "+session_id.split("_")[2], "%d/%m/%Y %H:%M")
        dates = []
        for i in range(len(cgm)):
            if len(dates) == 0:
                dates.append(ini_date + timedelta(minutes=5))
            else:
                dates.append(dates[-1] + timedelta(minutes=5))
        tdates = tdates + dates
        tcgm = tcgm + cgm.lectura.tolist()

        i = 0
        for i in range(len(dates)):
            if (dates[i] - ini_date).seconds >= (self.min_time * 60):
                break
        j = 0
        for j in range(i, len(dates)):
            if (dates[j] - ini_date).seconds >= (self.max_time * 60):
                break
        f_dates = dates[i:j]
        f_cgm = cgm_filtered.lectura.tolist()
        filtered_plots.append(
            go.Scatter(x=f_dates, y=f_cgm, mode='lines', name=session_id.replace("_", " "),
                       line=dict(color='#ff8700'))
        )

        y_marks.append(cgm.lectura.tolist()[0])
        x_marks.append(dates[0])

    # Draw
    layout = {
        'title': pacient,
        'shapes': [{
            'type': 'rect',
            'xref': 'paper',
            'yref': 'y',
            'x0': 0,
            'y0': min,
            'x1': 1,
            'y1': max,
            'fillcolor': '#b3ffb3',
            'opacity': 0.2,
            'line': {'width': 0}
        }],
    }

    fig = {
        'data': [go.Scatter(x=tdates, y=tcgm, name="Glucose", legendgroup="g1"),
                 go.Scatter(x=tdates, y=[max] * len(tdates), name="Upper Target", legendgroup="g3", mode='lines',
                            line=dict(color='#cc0000')),
                 go.Scatter(x=tdates, y=[min] * len(tdates), name="Lower Target", legendgroup="g3", mode='lines',
                            line=dict(color='#000000'))]
                + filtered_plots
                + [go.Scatter(x=x_marks, y=y_marks, name="Carbohydrates/Insulin", mode='markers', legendgroup="g4",
                              line=dict(color='#40bf00'))],
        'layout': layout,
    }
    image = plotly.offline.plot(fig, auto_open=False, show_link=False, output_type='div')
    return image

def draw_hist(self, session_id):
    """ Draw the histogram of lectures of a session"""
    cgm_ts = pd.read_sql(
        "SELECT sessio_id, lectura FROM LecturesCGM WHERE sessio_id = '{0}' AND mtimestamp >= {1}"
        " AND mtimestamp < {2};".format(session_id, self.min_time, self.max_time),  self.db_connection
    )
    cgm_ts = cgm_ts.lectura
    layout = {
        'title': "Used data Histogram",
    }
    fig = {
        'data': [go.Histogram(x=cgm_ts, autobinx=False, xbins=dict(start=2.05, end=20.05, size=0.1))],
        'layout': layout,
    }
    return plotly.offline.plot(fig, auto_open=False, show_link=False, output_type='div')

def draw_word(self, word_id, session_id=None):
    """ Draw a word. If a session id is given, its used in the title of the plot """
    wcol_names = [tup[1] for tup in self.cursor.execute('PRAGMA TABLE_INFO(Words)').fetchall() if
                  tup[1] not in ['id', 'label']]

    wcol_names = str(wcol_names)[1:-1].replace("'", "")
    cols = wcol_names.replace("_", ".").replace("n", "")
    vals = self.cursor.execute("SELECT * FROM Words WHERE id = '{0}';".format(word_id)).fetchone()
    vals = vals[1:-1]
    name = word_id
    if session_id:
        name = "{0} / {1}".format(session_id, name)
    layout = {
        'title': name,
    }

    fig = {
        'data': [go.Bar(x=cols, y=vals, showlegend=False)],
        'layout': layout,
    }
    image = plotly.offline.plot(fig, auto_open=False, show_link=False, output_type='div')
    return image

def draw_vocabulary(self):
    """ Draw all the words of the table 'Words' """
    imatges = []
    for word_id in self.cursor.execute("SELECT id FROM Words;").fetchall():
        imatges.append(self.draw_word(word_id[0]))
```

```
        return imatges

    def draw_sequence(self, session_id, names=True):
        """ Draw all sessions of sequence of session_id"""
        seq_names = [tup[1] for tup in self.cursor.execute('PRAGMA TABLE_INFO(Sequences)').fetchall() if tup[1] not in
                     ['sessio_id', 'next_label']]
        seq_names = str(seq_names)[1:-1].replace("'", "")
        sessions = self.cursor.execute(
            "SELECT {1} FROM Sequences WHERE sessio_id = '{0}';".format(session_id, seq_names)
        ).fetchone()
        images = []
        for id in sessions:
            word = self.cursor.execute("SELECT word_id FROM Sessions WHERE id = '{0}';".format(id)).fetchone()[0]
            if not names:
                id = None
            images.append(self.draw_word(word, id))
        return images

    def add_generator(self, generator_name):
        """
        Adds a new 'label' to the model generator by creating a new generator 'generator_name' using the current
        generator.
        """
        self.mgen = generator_name(self.mgen)

    def set_generator(self, generator_obj):
        """ Replace current generator with generator_obj """
        self.mgen = generator_obj

    def generate_models(self, args_dict):
        """ Call 'generate_models' method of ModelGenerator"""
        self.mgen.generate_models(args_dict)

    def get_models(self, session_ids, args_dict=None):
        """ Call 'get_models' method of ModelGenerator"""
        return self.mgen.get_models(session_ids, args_dict)

    def get_models_label(self, session_ids):
        """ Call 'get_models_label' method of ModelGenerator"""
        return self.mgen.get_models_label(session_ids)

class ModelsGenerator(object):

    def __init__(self, model_generator=None, loader=None):
        if model_generator:
            self.loader = model_generator.loader
            self.child_mgen = model_generator
        elif loader:
            self.loader = loader
            self.child_mgen = None
        else:
            raise Exception("Loader required")
        self.cursor = self.loader.cursor

    def generate_models(self, args_dict):
        """ Generate models of sessions """
        if self.child_mgen:
            self.child_mgen.generate_models(args_dict)
        self._generate_models(args_dict)

    def get_models(self, session_ids, args_dict=None):
        """ Returns models of given sessions"""
        if args_dict is None:
            args_dict = {}
        return self._get_models(session_ids, args_dict)

    def get_models_label(self, session_ids):
        """ Returns labels of models of given sessions """
        return self._get_models_label(session_ids)

    def _generate_models(self, args_dict):
        return None

    def _get_models(self, session_ids, args_dict):
        return None

    def _get_models_label(self, session_ids):
        return self.loader.get_session_labels(session_ids)


class SimpleModels(ModelsGenerator):

    def _generate_models(self, args_dict):
        df = pd.read_sql("SELECT Sessions.* FROM Sessions", self.loader.db_connection)
        df.to_sql("SessionsNorm", self.loader.db_connection, if_exists="replace", index=False)

    def _get_models(self, session_ids, args_dict):
        return pd.read_sql("SELECT SessionsFDP.*, Sessions.label FROM Sessions INNER JOIN SessionsFDP "
                           "ON Sessions.id = SessionsFDP.id WHERE Sessions.id IN ({0});"
                           .format(str(session_ids)[1:-1]), self.loader.db_connection)

class NormalizedModels(SimpleModels):

    def _generate_models(self, args_dict):
        df = pd.read_sql("SELECT SessionsNorm.* FROM SessionsNorm", self.loader.db_connection)
        x = df.iloc[:, 3:-2].values
        min_max_scaler = preprocessing.MinMaxScaler()
        df_scaled = pd.DataFrame(min_max_scaler.fit_transform(x))
        final_df = df.iloc[:, :3]
        i = 0
        for col_name in df.iloc[0, 3:-2].index:
            final_df = final_df.assign(aux=df_scaled.iloc[:, i].values)
            final_df.columns = list(final_df.columns)[:-1] + [col_name]
            i += 1
```

```
        final_df = final_df.assign(aux=df.iloc[:, -2].values)
        final_df.columns = list(final_df.columns)[:-1] + ['word_id']

        final_df = final_df.assign(aux=df.iloc[:, -1].values)
        final_df.columns = list(final_df.columns)[:-1] + ['label']

        final_df.to_sql("SessionsNorm", self.loader.db_connection, if_exists="replace")

class WordsModels(ModelsGenerator):

    def _generate_models(self, args_dict):
        k = args_dict.get('k', 5)
        try:
            self.cursor.execute("DROP TABLE Words;")
        except:
            pass
        all_sessions = self.loader.get_all_sessions()
        models = self.child_mgen.get_models(all_sessions)
        for lb_value, lb_rep in [(lb, str(lb)) for lb in [-1, 0, 1]]:
            fdp_data = models.ix[models.label == lb_value]
            kmeans, words = self.create_vocabulary(k, fdp_data)
            new = NamedTemporaryFile(mode="w")
            new.write('id;')
            for index in fdp_data.columns[1:-1]:
                new.write("n{0};".format(index).replace(".", "_"))
            new.write('label\n')
            i = 0
            for word in words:
                i += 1
                new.write(("w{0}_{1};".format(lb_rep, i)+str(word[:].round(5).tolist())[1:-1]+"\n").replace(',', ';')
                          .replace("-0.0;", "0.0;").replace("0.0;", "0;"))
            labels = kmeans.predict(fdp_data[fdp_data.columns[1:]])
            for i in range(len(labels)):
                self.cursor.execute(
                    "UPDATE Sessions SET word_id='w{0}_{1}' WHERE id='{2}';"
                        .format(lb_rep, labels[i]+1, fdp_data.iloc[i, 0])
                )
            new.flush()
            new.seek(0)
            df = pd.DataFrame.from_csv(new.name, sep=';')
            df.to_sql("Words", self.loader.db_connection, if_exists="append")
            new.close()

    def _get_models(self, session_ids, args_dict):
        weight = args_dict.get("weight", 1)
        word_weight = args_dict.get("word_weight", 1)

        word_cols = str(["Words.{0}*{1}".format(tup[1], word_weight) for tup in
                         self.cursor.execute('PRAGMA TABLE_INFO(Words)').fetchall()
                         if tup[1] not in ['id', 'label']]).replace("'","")[1:-1]

        sstring = str(session_ids)[1:-1]
        pacients_str = "Pacients.sexe*{0}, ((1.0*Pacients.edat)/100)*{0}".format(weight)
        return pd.read_sql(
            "SELECT SessionsNorm.id, {3}, {0}, {1}, SessionsNorm.label FROM SessionsNorm INNER JOIN Sessions ON SessionsNorm.id = Session
            "  JOIN  Pacients ON Sessions.pacient_id=Pacients.id WHERE SessionsNorm.id IN ({2});"
                .format(self.loader.get_session_fields(weight=weight), word_cols, sstring, pacients_str), self.loader.db_connection
        )


    @staticmethod
    def create_vocabulary(k, cgm_models):
        """
        :param  k:  number of clusters for the kmeans algorithm
        :param  cgm_models: dataframe with the caracteristics of the models.
                            There is a row for each session with its respective model.
                            The dataframe has the folowing columns:
                              * unnamed: integer to identify each row
                              * id: string that identifies the session of the lecture: {PacientName}_{DD/MM/YYYY}_{h:m}.
                                The date and hour are the ones from the begining of the session.
                              * 1 column for each caracteristic of the model.
                              * label: says if its classified as hipo/hiper/normal
        :returns
        kmeans:     kmeans object made with the data of 'cgm_models'.
        centroids:  ndarray with the data about the centroids generated by 'kmeans'.
        """
        data = cgm_models[cgm_models.columns[1:]]
        if k > len(data):
            print("ERROR generating Vocabulary: words reduced from {0} to {1}".format(k, len(data)))
            k = len(data)
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(data)
        centroids = kmeans.cluster_centers_
        for c in centroids:
            c[-1] = round(c[-1])
        return kmeans, centroids


class SequenceModels(ModelsGenerator):

    def _generate_models(self, args_dict):
        seq_len = args_dict.get('ns', 3)
        new_data = []
        for sessio_info in self.cursor.execute("SELECT id, date, pacient_id from Sessions;").fetchall():
            anteriors = self.cursor.execute(
                "SELECT id FROM Sessions WHERE pacient_id='{0}' AND date < '{1}' ORDER BY date DESC LIMIT {2};"
                    .format(sessio_info[2], sessio_info[1], seq_len)
            ).fetchall()
            aux_list = []
            for i in anteriors:
                aux_list.append(i[0])
            if len(aux_list) != seq_len:
                continue
```

```
            aux_list = [sessio_info[0]] + list(reversed(aux_list))
            new_data.append(aux_list)
        indexs = []
        for i in range(1, seq_len + 1):
            indexs.append("s{0}".format(i))
        new_df = pd.DataFrame(new_data, columns=['sessio_id'] + indexs)
        new_df.to_sql("Sequences", self.loader.db_connection, if_exists="replace", index=False)

    def _get_models(self, session_ids, args_dict):
        new_data = []
        ant_model = None
        weight = args_dict.get("weight", 1)
        sfields = self.loader.get_session_fields(weight=weight)
        pacients_str = "Pacients.sexe*{0}, ((1.0*Pacients.edat)/100)*{0}".format(weight)
        for session_id in session_ids:
            aux = [x for x in self.cursor.execute("SELECT SessionsNorm.id, {2}, {0} FROM SessionsNorm JOIN Pacients ON pacient_id = Pacie
                                                  .format(sfields, session_id, pacients_str)).fetchone()]
            ant_sessions = self.cursor.execute(
                "SELECT * FROM Sequences WHERE sessio_id = '{0}';".format(session_id)
            ).fetchone()
            if ant_sessions:
                ant_sessions = ant_sessions[1:]
                for ant_id in ant_sessions:
                    args_dict.update({"word_weight": self.calc_sequence_weight(session_id, ant_id)})
                    ant_model = self.child_mgen.get_models([ant_id], args_dict=args_dict)
                    aux = aux + ant_model.iloc[:, 1:-1].values.tolist()[0]
                word = self.cursor.execute(
                    "SELECT word_id FROM Sessions WHERE id = '{0}';".format(session_id)
                ).fetchone()[0]
                aux.append(word)
                new_data.append(aux)
        names = ["id", "sexe", "edat"] + sfields.replace("SessionsNorm.", "").replace(" ", "").split(",")
        if ant_model is not None:
            aux = list(ant_model.columns.values)[1:-1]
            seq_names = [tup[1] for tup in self.cursor.execute('PRAGMA TABLE_INFO(Sequences)').fetchall()
                         if tup[1] not in ['sessio_id']]
            for seq in seq_names:
                for n in aux:
                    names.append("{0}_{1}".format(seq, n))
        names.append("label")
        return pd.DataFrame(new_data, columns=names)

    def calc_sequence_weight(self, session_id, ant_id):
        date1 = session_date(session_id)
        date2 = session_date(ant_id)
        days = (date1 - date2).days
        weight = 1/(days+1)
        return round(weight, 2)


class LmFitModels(ModelsGenerator):

    def _generate_models(self, args_dict):
        model_name = args_dict.get("lmfit_model", "GaussianModel")
        model_obj = getattr(mdl, model_name)
        all_sessions = self.loader.get_all_sessions()
        try:
            all_sessions = [x[0] for x
                            in self.loader.cursor.execute("SELECT id FROM LmFitModels WHERE id NOT IN ({0})"
                                                          .format(str(all_sessions)[1:-1])).fetchall()]
        except Exception as e:
            pass
        smodels = self.child_mgen.get_models(all_sessions, args_dict)
        res = []
        columns = None
        for i in range(len(smodels)):
            data = smodels.iloc[i, 1:-1]
            aux = self.fit_model(data, model_obj())
            if not aux:
                continue
            if not columns:
                columns = ['id'] + list(aux.keys())
            res.append([smodels.iloc[i, 0]] + list(aux.values()))
        new_df = pd.DataFrame(res, columns=columns)
        new_df.to_sql("LmFitModels", self.loader.db_connection, if_exists="append", index=False)

    def _get_models(self, session_ids, args_dict):
        return pd.read_sql(
            "SELECT LmFitModels.*, Sessions.label FROM Sessions INNER JOIN LmFitModels ON Sessions.id = LmFitModels.id"
            " WHERE Sessions.id IN ({0});".format(str(session_ids)[1:-1]), self.loader.db_connection)

    @staticmethod
    def fit_model(data, model):
        """
        :param
        data:   pandas.series with the histogram of the values occured in a session.
                The series has two unnamed columns:
                    * key: Glucose level value
                    * value: number of ocurences of the Glucose level value in this session

        model:  Model object from lmfit.models

        :return out:    Model object from lmfit.models constructed from the data passed
        """
        try:
            x = pd.Series([float(x) for x in data.index.values])
            y = pd.Series(data.values.tolist())
            pars = model.guess(y, x=x)
            out = model.fit(y, pars, x=x)
            return {par[0]: par[1].value for par in out.params.items()}
        except Exception as e:
            print(e)
            pass
```

```
class LmFitWordsModels(WordsModels):

    def _generate_models(self, args_dict):
        k = args_dict.get('k', 5)
        try:
            self.cursor.execute("DROP TABLE Words;")
        except:
            pass
        all_sessions = self.loader.get_all_sessions()
        models = self.child_mgen.get_models(all_sessions, args_dict)

        model_name = args_dict.get("lmfit_model", "GaussianModel")
        model_obj = getattr(mdl, model_name)
        res = []
        columns = None

        for lb_value, lb_rep in [(lb, str(lb)) for lb in [-1, 0, 1]]:
            fdp_data = models.ix[models.label == lb_value]
            kmeans, words = self.create_vocabulary(k, fdp_data)
            new = NamedTemporaryFile(mode="w")
            new.write('id;')
            for index in fdp_data.columns[1:-1]:
                new.write("n{0};".format(index).replace(".", "_"))
            new.write('label\n')
            i = 0
            for word in words:
                i += 1
                new.write(("w{0}_{1};".format(lb_rep, i)+str(word[:].round(10).tolist())[1:-1]+"\n").replace(',', ';')
                          .replace("-0.0;", "0.0;").replace("0.0;", "0;"))
            labels = kmeans.predict(fdp_data[fdp_data.columns[1:]])
            for i in range(len(labels)):
                self.cursor.execute(
                    "UPDATE Sessions SET word_id='w{0}_{1}' WHERE id='{2}';"
                        .format(lb_rep, labels[i]+1, fdp_data.iloc[i, 0])
                )
            new.flush()
            new.seek(0)
            df = pd.DataFrame.from_csv(new.name, sep=';', index_col=None)
            new.close()

            for i in range(len(df)):
                data = df.iloc[i, 1:-1]
                aux = self.fit_model(data, model_obj())
                if not aux:
                    continue
                if not columns:
                    columns = ['id'] + list(aux.keys()) + ['label']
                res.append([df.iloc[i, 0]] + list(aux.values()) + [df.iloc[i, -1]])

            new_df = pd.DataFrame(res, columns=columns)
            new_df.to_sql("Words", self.loader.db_connection, if_exists="append", index=False)

    @staticmethod
    def fit_model(data, model):
        """
        :param
        data:   pandas.series with the histogram of the values occured in a session.
                The series has two unnamed columns:
                    * key: Glucose level value
                    * value: number of ocurences of the Glucose level value in this session

        model:  Model object from lmfit.models

        :return out:    Model object from lmfit.models constructed from the data passed
        """
        try:
            x = pd.Series([float(x[1:].replace("_", ".")) for x in data.index.values])
            y = pd.Series(data.values.tolist())
            pars = model.guess(y, x=x)
            out = model.fit(y, pars, x=x)
            return {par[0]: par[1].value for par in out.params.items()}
        except Exception as e:
            print(e)
            pass

def myround(x, prec=3, base=.025):
    return round(base * round(float(x)/base), prec)


def session_date(sesion_name):
    """Takes a session id with format: {name}_{date}_{hour} and returns a date object"""
    name, date, hour = sesion_name.split('_')
    day, month, year = date.split('/')
    date_obj = datetime.strptime('{0}/{1}/{2} {3}'.format(day, month, year, hour), '%d/%m/%Y %H:%M')
    return date_obj


def smooth(x, window_len=11, window='hanning'):
    """smooth the data using a window with requested size.

    This method is based on the convolution of a scaled window with the signal.
    The signal is prepared by introducing reflected copies of the signal
    (with the window size) in both ends so that transient parts are minimized
    in the begining and end part of the output signal.

    input:
        x: the input signal
        window_len: the dimension of the smoothing window; should be an odd integer
        window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'
            flat window will produce a moving average smoothing.

    output:
        the smoothed signal
```

```
example:

t=linspace(-2,2,0.1)
x=sin(t)+randn(len(t))*0.1
y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve
scipy.signal.lfilter

TODO: the window parameter could be the window itself if an array instead of a string
NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.
"""

if x.ndim != 1:
    raise (ValueError, "smooth only accepts 1 dimension arrays.")

if x.size < window_len:
    raise (ValueError, "Input vector needs to be bigger than window size.")

if window_len < 3:
    return x

if not window in ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']:
    raise (ValueError, "Window is on of 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'")

s = np.r_[x[window_len - 1:0:-1], x, x[-1:-window_len:-1]]
if window == 'flat':  # moving average
    w = np.ones(window_len, 'd')
else:
    w = eval('np.' + window + '(window_len)')

y = np.convolve(w / w.sum(), s, mode='valid')
res = y[(int(window_len/2)-1):-(int(window_len/2))-1]
return res
```

CGMPredict.py

```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.neighbors import NearestNeighbors
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
import numpy as np


class Predicter(object):

    def __init__(self, loader, words_num=20, seq_len=3, weights=1):
        """
        :param loader:  object of the type Loader. It contains all the data about CGM and
                        is able to generate the models with which the classifier will work.
        :param words_num:   number of words of the vocabulary generated
        :param seq_len: number of sessions of the sequences generated
        """
        self.loader = loader
        self.loader.generate_models({'k': words_num, 'ns': seq_len})
        self.weights = weights

    def predict(self, session_ids, train_sessions=None):
        """
        :param session_ids:  ids of the session to predict
        :param train_index: ids of the data used as train to built the predict_obj.
                            If no train_index is given, it will be used all the data exept the ones of the 'session_ids'
        :return: labels predicted
        """
        if not train_sessions:
            train_sessions = self.loader.get_all_sessions(for_train=True)
            train_sessions = [x for x in train_sessions if x not in session_ids]
        res = self._predict(train_sessions, session_ids)
        return res

    def _predict(self, train_index, model_indexs):
        """
        :param model_indexs:  ids of the session to predict
        :param train_index: ids of the data used as train to built the predict_obj.
        :return labels predicted
        """
        """To be implemented by childs"""
        return None

    def get_similar_models(self, session_id, n=5):
        """
        :param session_id: id of a session
        :param n: number of sequences
        :return: ids of the n sessions most similar to session_id
        """
        neigh = NearestNeighbors(n_neighbors=n)

        train_sessions = self.loader.get_all_sessions(for_train=True)
        train_sessions = [x for x in train_sessions if x != session_id]
        models = self.loader.get_models(train_sessions, {'to_test': True})
        models_data = models.iloc[:, 1:-1]

        data_pred = self.loader.get_models([session_id], {'to_test': False})
        data_pred = data_pred .iloc[:, 1:-1]

        neigh.fit(models_data)
        res = neigh.kneighbors(data_pred)

        session_res = []
        for i in range(len(res)):
            session_index = res[1][0][i]
            session_name = models.iloc[session_index, 0]

            session_conf = res[0][0][i]

            session_res.append((session_name, session_conf))

        return session_res


class KNNPredicter(Predicter):

    def __init__(self, loader, k=5, words_num=20, seq_len=3, weights=1):
        """
        :param loader:  object of the type Loader. It contains all the data about CGM and
                        is able to generate the models with which the classifier will work.
        :param k:   number of neighbours used in the algortihm of KNN
        :param words_num:   number of words of the vocabulary generated
        :param seq_len: number of sessions of the sequences generated
        """
        super(KNNPredicter, self).__init__(loader, words_num, seq_len, weights)
        self._K = k

    def _predict(self, train_index, test_index):
        """
        :param train_index: list with the index of the models data used in the algorithm of SVM.
        :param test_index:  list with the index to predict.
        :return:    tuple with (label_prediction, label_score, word_prediction, word_score, bin_predictions) of test_index made by:
                        - predict Label (hiper/hipo/normal) with KNN (using all trainindex)
                        - predict word with KNN using only hiper/hipo/normal trainindex depending on the label predicted
        """
        # models to predict
        data_pred = self.loader.get_models(test_index, {'weight': self.weights})
        data_pred = data_pred.iloc[:, 1:-1]
```

```
        # Get models with words
        wdata = self.loader.get_models(train_index, {'weight': self.weights})

        # Get models hipo/hiper/norm labels
        ldata_labels = self.loader.get_labels_of_words(wdata.iloc[:, -1].tolist())

        # Predict label
        knn = OneVsRestClassifier(KNN(n_neighbors=self._K))
        y = label_binarize(ldata_labels, classes=[-1, 0, 1])
        knn.fit(wdata.iloc[:, 1:-1], y)
        bin_labels = knn.predict(data_pred)
        score_label = knn.predict_proba(data_pred)

        # Predict word using only vocabulary of label predicted
        res_word = []
        res_label = []
        score_word = []
        predicters = {}
        for i in range(len(bin_labels)):
            label = None
            for l in range(-1, 2):
                if bin_labels[i][l+1] == 1:
                    label = l
                    break
            if not label:
                maxscore = -1000
                for s in range(-1, 2):
                    score = score_label[i][s+1]
                    if score >= maxscore:
                        label = s
                        maxscore = score
            res_label.append(label)
            # Work only with sessions of the label
            sessions = [wdata.iloc[z, 0] for z in range(len(wdata)) if ldata_labels[z] == label]

            models = wdata[wdata.id.isin(sessions)]
            models_labels = models.iloc[:, -1]
            models = models.iloc[:, 1:-1]

            if not predicters.get(str(label), False):
                knn = KNN(n_neighbors=self._K)
                knn.fit(models, models_labels)
                predicters[str(label)] = knn

            knn = predicters.get(str(label))
            res_w = knn.predict(data_pred)
            score_w = knn.predict_proba(data_pred)

            res_word.append(res_w)
            score_word.append(score_w)

        return (res_label, score_label, res_word, score_word, bin_labels)


class SVMPredicter(Predicter):
    def __init__(self, loader=None, kernel='rbf', c=10, gamma=0.1, words_num=5, seq_len=3, wheights=1):
        """
        :param loader:  object of the type Loader. It contains all the data about CGM and
                        is able to generate the models with which the classifier will work.
        :param kernel: 'rbf', 'linear', 'poly', 'sigmoid'. Kernel of the algortihm SVM
        :param words_num:   number of words of the vocabulary generated
        :param seq_len: number of sessions of the sequences generated
        """
        super().__init__(loader, words_num, seq_len, wheights)
        self._kernel = kernel
        self._C = c
        self._gamma = gamma

    def _predict(self, train_index, test_index):
        """
        :param train_index: list with the index of the models data used in the algorithm of SVM.
        :param test_index:  list with the index to predict.
        :return:    tuple with (label_prediction, label_score, word_prediction, word_score, bin_predictions) of test_index made by:
                        - predict Label (hiper/hipo/normal) with SVM (using all trainindex)
                        - predict word with SVM using only hiper/hipo/normal trainindex depending on the label predicted
        """
        # models to predict
        data_pred = self.loader.get_models(test_index, {'weight': self.weights})
        data_pred = data_pred.iloc[:, 1:-1]

        # Get models with words
        wdata = self.loader.get_models(train_index, {'weight': self.weights})

        # Get models hipo/hiper/norm labels
        ldata_labels = self.loader.get_labels_of_words(wdata.iloc[:, -1].tolist())

        # Predict label
        svm = OneVsRestClassifier(SVC(kernel=self._kernel, C=self._C, gamma=self._gamma))
        y = label_binarize(ldata_labels, classes=[-1, 0, 1])
        svm.fit(wdata.iloc[:, 1:-1], y)
        bin_labels = svm.predict(data_pred)
        score_label = svm.decision_function(data_pred)

        # Predict word using only vocabulary of label predicted
        res_word = []
        res_label = []
        score_word = []
        predicters = {}
        for i in range(len(bin_labels)):
            label = None
```

```
            for l in range(-1, 2):
                if bin_labels[i][l + 1] == 1:
                    label = l
                    break
            if not label:
                maxscore = -1000
                for s in range(-1, 2):
                    score = score_label[i][s + 1]
                    if score >= maxscore:
                        label = s
                        maxscore = score
            res_label.append(label)
            # Work only with sessions of the label
            sessions = [wdata.iloc[z, 0] for z in range(len(wdata)) if ldata_labels[z] == label]

            models = wdata[wdata.id.isin(sessions)]
            models_labels = models.iloc[:, -1]
            models = models.iloc[:, 1:-1]

            if not predicters.get(str(label), False):
                svm = SVC(kernel=self._kernel, C=self._C, gamma=self._gamma)
                svm.fit(models, models_labels)
                predicters[str(label)] = svm

            svm = predicters.get(str(label))
            res_w = svm.predict(data_pred)
            score_w = svm.decision_function(data_pred)

            res_word.append(res_w)
            score_word.append(score_w)

        return (res_label, score_label, res_word, score_word, bin_labels)


class ApatsSVMPredicter(Predicter):
    def __init__(self, loader=None, kernel='rbf', c=1.0, gamma='auto', words_num=20, seq_len=3, wheights=1):
        """
        :param loader:  object of the type Loader. It contains all the data about CGM and
                        is able to generate the models with which the classifier will work.
        :param kernel: 'rbf', 'linear', 'poly', 'sigmoid'. Kernel of the algortihm SVM
        :param words_num:   number of words of the vocabulary generated
        :param seq_len: number of sessions of the sequences generated
        """
        super().__init__(loader, words_num, seq_len, wheights)
        self._kernel = kernel
        self._C = c
        self._gamma = gamma

    def _predict(self, train_index, test_index):
        """
        :param train_index: list with the index of the models data used in the algorithm of SVM.
        :param test_index:  list with the index to predict.
        :return:    tuple with (label_prediction, label_score, word_prediction, word_score, bin_predictions) of test_index made by:
                        - predict Label (hiper/hipo/normal) with SVM (using all trainindex of the same meal type)
                        - predict word with SVM using only hiper/hipo/normal trainindex depending on the label predicted
        """
        # models to predict
        data_pred = self.loader.get_models(test_index, {'weight': self.weights})
        data_pred = data_pred.iloc[:, 1:-1]

        # Get models with words
        wdata = self.loader.get_models(train_index, {'weight': self.weights})

        bin_labels = []
        score_label = []

        # Get Possible meal labels
        meal_types = list(wdata.iloc[:, 3].unique())
        for d in data_pred.iloc[:, 2].unique():
            if d not in meal_types:
                meal_types.append(d)

        # split models by type of meal
        for etiqueta_apat in meal_types:
            mdata_pred = data_pred.ix[data_pred.iloc[:, 2] == etiqueta_apat]
            mwdata = wdata.ix[wdata.iloc[:, 3] == etiqueta_apat]

            # Get models hipo/hiper/norm labels
            ldata_labels = self.loader.get_labels_of_words(mwdata.iloc[:, -1].tolist())

            # Predict label
            svm = OneVsRestClassifier(SVC(kernel=self._kernel, C=self._C, gamma=self._gamma))
            y = label_binarize(ldata_labels, classes=[-1, 0, 1])
            svm.fit(mwdata.iloc[:, 1:-1], y)
            mbin_labels = svm.predict(mdata_pred)
            mscore_label = svm.decision_function(mdata_pred)
            if len(bin_labels):
                bin_labels = np.concatenate((bin_labels, mbin_labels), axis=0)
                score_label = np.concatenate((score_label, mscore_label), axis=0)
            else:
                bin_labels = mbin_labels
                score_label = mscore_label

        ldata_labels = self.loader.get_labels_of_words(wdata.iloc[:, -1].tolist())
        # Predict word using only vocabulary of label predicted
        res_word = []
        res_label = []
        score_word = []
        predicters = {}
        for i in range(len(bin_labels)):
            label = None
```

```
            for l in range(-1, 2):
                if bin_labels[i][l + 1] == 1:
                    label = l
                    break
            if not label:
                maxscore = -1000
                for s in range(-1, 2):
                    score = score_label[i][s + 1]
                    if score >= maxscore:
                        label = s
                        maxscore = score
            res_label.append(label)
            # Work only with sessions of the label
            sessions = [wdata.iloc[z, 0] for z in range(len(wdata)) if ldata_labels[z] == label]

            models = wdata[wdata.id.isin(sessions)]
            models_labels = models.iloc[:, -1]
            models = models.iloc[:, 1:-1]

            if not predicters.get(str(label), False):
                svm = SVC(kernel=self._kernel, C=self._C, gamma=self._gamma)
                svm.fit(models, models_labels)
                predicters[str(label)] = svm

            svm = predicters.get(str(label))
            res_w = svm.predict(data_pred)
            score_w = svm.decision_function(data_pred)

            res_word.append(res_w)
            score_word.append(score_w)

        return (res_label, score_label, res_word, score_word, bin_labels)


class SVM_KNN_Predicter(Predicter):
    def __init__(self, loader=None, kernel='rbf', c=1.0, gamma='auto', words_num=20, seq_len=3, wheights=1):
        """
        :param loader:  object of the type Loader. It contains all the data about CGM and
                        is able to generate the models with which the classifier will work.
        :param kernel: 'rbf', 'linear', 'poly', 'sigmoid'. Kernel of the algortihm SVM
        :param words_num:   number of words of the vocabulary generated
        :param seq_len: number of sessions of the sequences generated
        """
        super().__init__(loader, words_num, seq_len, wheights)
        self._kernel = kernel
        self._C = c
        self._gamma = gamma

    def _predict(self, train_index, test_index):
        """
        :param train_index: list with the index of the models data used in the algorithm of SVM.
        :param test_index:  list with the index to predict.
        :return:    tuple with (label_prediction, label_score, word_prediction, word_score, bin_predictions) of test_index made by:
                        - predict Label (hiper/hipo/normal) with SVM (using all trainindex)
                        - predict word with KNN using only hiper/hipo/normal trainindex depending on the label predicted
        """
        # models to predict
        data_pred = self.loader.get_models(test_index, {'weight': self.weights})
        data_pred = data_pred.iloc[:, 1:-1]

        # Get models with words
        wdata = self.loader.get_models(train_index, {'weight': self.weights})

        # Get models hipo/hiper/norm labels
        ldata_labels = self.loader.get_labels_of_words(wdata.iloc[:, -1].tolist())

        # Predict label
        svm = OneVsRestClassifier(SVC(kernel=self._kernel, C=self._C, gamma=self._gamma))
        y = label_binarize(ldata_labels, classes=[-1, 0, 1])
        svm.fit(wdata.iloc[:, 1:-1], y)
        bin_labels = svm.predict(data_pred)
        score_label = svm.decision_function(data_pred)

        # Predict word using only vocabulary of label predicted
        res_word = []
        res_label = []
        score_word = []
        predicters = {}
        for i in range(len(bin_labels)):
            label = None
            for l in range(-1, 2):
                if bin_labels[i][l + 1] == 1:
                    label = l
                    break
            if not label:
                maxscore = -1000
                for s in range(-1, 2):
                    score = score_label[i][s + 1]
                    if score >= maxscore:
                        label = s
                        maxscore = score
            res_label.append(label)
            # Work only with sessions of the label
            sessions = [wdata.iloc[z, 0] for z in range(len(wdata)) if ldata_labels[z] == label]

            models = wdata[wdata.id.isin(sessions)]
            models_labels = models.iloc[:, -1]
            models = models.iloc[:, 1:-1]

            if not predicters.get(str(label), False):
                knn = KNN(n_neighbors=5)
```

```
            knn.fit(models, models_labels)
            predicters[str(label)] = knn

        knn = predicters.get(str(label))
        res_w = knn.predict(data_pred)
        score_w = knn.predict_proba(data_pred)

        res_word.append(res_w)
        score_word.append(score_w)

    return (res_label, score_label, res_word, score_word, bin_labels)
```

CGMTester.py

```
from CGMLoader import *
import CGMPredict
from sklearn.model_selection import StratifiedKFold
import pandas as pd
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score, auc, roc_curve
import signal
from sklearn.preprocessing import label_binarize
import random
import math

class CGMTester(object):

    def __init__(self, file):
        self.loader = Loader(new_db=True)
        self.loader.load_data_from_csv(file)
        self.loader.generate_fdps(normalized=True)
        self.best = None
        if not os.path.exists("results"):
            os.mkdir("results")

    @staticmethod
    def signal_handler(signum, frame):
        raise Exception("Timed out!")

    def test(self, params, predict_obj_names=["SVMPredicter"],
             model_generator_names=[("SequenceModels","WordsModels","SimpleModels")],
             nfolds=10, prefix=""):
        self.prefix = prefix
        import CGMLoader as models_module
        for model_gen_tuple in model_generator_names:
            for predict_obj_name in predict_obj_names:
                predict_obj = getattr(CGMPredict, predict_obj_name)
                model_filename = ""
                model_obj = None
                for gen in reversed(model_gen_tuple):
                    if gen not in ["SequenceModels","WordsModels","SimpleModels"]:
                        model_filename += "{0}_".format(gen)
                    new_model = getattr(models_module, gen)
                    if not model_obj:
                        model_obj = new_model(loader=self.loader)
                    else:
                        model_obj = new_model(model_obj)
                for para_list in params:
                    timeout = False
                    some_ok = False
                    second_fail = False
                    res = []
                    filename = predict_obj_name + "_" + model_filename + \
                               str(para_list)[1:-1].replace(",","_").replace(" ","").replace("'","")
                    filename += "_results"
                    if self.loader.round_val != para_list[0]:
                        self.loader.round_val = para_list[0]
                        self.loader.clean_data()
                        self.loader.generate_fdps(normalized=True)
                        try:self.loader.cursor.execute("DROP TABLE LmFitModels;")
                        except: pass
                    self.loader.set_generator(model_obj)
                    try:
                        predicter = predict_obj(self.loader, *para_list[1:])
                    except Exception as e:
                        print(e)
                    sessions_list = self.loader.get_all_sessions(for_train=True)
                    folds = self._get_stratified_kfolds(k=nfolds, sessions=sessions_list)
                    i = 0
                    for (train_ids, test_ids) in folds:
                        i += 1
                        signal.signal(signal.SIGALRM, self.signal_handler)
                        signal.alarm(60*30)  # 15 min
                        try:
                            info_pred = predicter.predict(test_ids, train_ids)
                        except Exception as msg:
                            print("ERROR in {0} fold {1}: {2}.".format(filename, i, msg))
                            timeout = True
                            if not second_fail:
                                second_fail = True
                                continue
                            else:
                                break

                        y_real = self.loader.get_models_label(test_ids)
                        res.append((info_pred, y_real))

                        some_ok = True
                        print("{0}: {1} fold DONE".format(filename, i))

                    if timeout and not some_ok:
                        print("ERROR in {0}. No folds success.".format(filename))
```

```
                    new = open('results/no_results_{0}.txt'.format(filename), 'w')
                    new.close()
                    break

                self.write_metrics(res, filename)
                print("Tested: {0} with {1} and {2}.".format(predict_obj_name, model_filename, para_list))
    if self.best:
        print("Best: {0}, with acuacity: {1}".format(self.best[0], self.best[1]))

def _get_stratified_kfolds(self, k, sessions):
    folds = []
    skf = StratifiedKFold(n_splits=k, random_state=False, shuffle=True)
    X = sessions
    y = pd.Series(self.loader.get_session_labels(X))
    X = pd.Series(X)
    for train, test in skf.split(X, y):
        y_train, y_test = y.iloc[train], y.iloc[test]
        # y_train, y_test = self.retallar_dataset(y_train, y_test, 1, 0, -1)
        X_train = X.iloc[y_train.index.values].tolist()
        X_test = X.iloc[y_test.index.values].tolist()
        folds.append((X_train, X_test))
    return folds

def write_metrics(self, fold_info_list, filename):
    new = open('results/{0}'.format(filename), 'w')
    # Get together the info of all folds
    #              y_pred  y_bin   y_scor  w_pred  w_scor  y_real
    total_info= [   [],     [],     [],     [],     [],     []     ]
    for fold_info, y_real in fold_info_list:
        total_info[0] += fold_info[0]
        total_info[1] += list(fold_info[4])
        total_info[2] += list(fold_info[1])
        total_info[3] += fold_info[2]
        total_info[4] += fold_info[3]
        total_info[5] += y_real
        # write fold info
        self.write_fold_metrics(new, fold_info[0], list(fold_info[4]), list(fold_info[1]), y_real)

    # write global info
    y_pred = total_info[0]
    y_bin = total_info[1]
    y_scor = total_info[2]
    y_real = total_info[5]
    ac = self.write_fold_metrics(new, y_pred, y_bin, y_scor, y_real)
    new.close()
    import os
    new_name = "{0}{1}_{2}.txt".format(self.prefix, ac, filename)
    os.rename('results/{0}'.format(filename), 'results/{0}'.format(new_name))
    print("New file: {0}".format(new_name))

def write_fold_metrics(self, nfile, y_pred, y_bin, y_scor, y_real):
    # Construct binari labels information
    y_pred_hipo = []
    y_real_hipo = []
    y_scor_hipo = []

    y_pred_hipe = []
    y_real_hipe = []
    y_scor_hipe = []

    y_pred_norm = []
    y_real_norm = []
    y_scor_norm = []

    y_pred_total = []
    y_real_total = []

    not_classified = 0

    for i in range(len(y_bin)):
        # Get label predicted from binary predictions
        label = None
        for l in range(-1, 2):
            if y_bin[i][l + 1] == 1:
                label = l
                break

        # Add to no classified if no label found and go to next iteration
        if label is None:
            not_classified +=1
            continue

        # Add labels to total lists
        y_pred_total.append(label)
        y_real_total.append(y_real[i])

        # Add labels to hipo lists
        target = '-1'
        self.add_binari_label_to_list(target, label, y_pred_hipo)
        self.add_binari_label_to_list(target, y_real[i], y_real_hipo)
```

```
            y_scor_hipo.append(y_scor[i][0])

            # Add labels to norm lists
            target = '0'
            self.add_binari_label_to_list(target, label, y_pred_norm)
            self.add_binari_label_to_list(target, y_real[i], y_real_norm)
            y_scor_norm.append(y_scor[i][1])

            # Add labels to hipe lists
            target = '1'
            self.add_binari_label_to_list(target, label, y_pred_hipe)
            self.add_binari_label_to_list(target, y_real[i], y_real_hipe)
            y_scor_hipe.append(y_scor[i][2])

        # Write metrics for binari labels ant total labels
        self.write_metrics_from_lists(nfile, y_pred_hipo, y_real_hipo, y_scor_hipo, "HYPOGLUCEMIA")
        self.write_metrics_from_lists(nfile, y_pred_norm, y_real_norm, y_scor_norm, "NORMALUCEMIA")
        self.write_metrics_from_lists(nfile, y_pred_hipe, y_real_hipe, y_scor_hipe, "HYPERGLUCEMIA")
        ac = self.write_metrics_from_lists(nfile, y_pred_total, y_real_total, None, "TOTAL")
        if math.isnan(ac) or not_classified > len(y_real)/2:
            ac = -1
        nfile.write("\nNO CLASSIFIED: {0}/{1}\n".format(not_classified, len(y_real)))
        ac2 = self.write_metrics_from_lists(nfile, y_pred, y_real, None, "Non Confidence Predictions")
        return max([round(ac, 4), round(ac2, 4)])

    @staticmethod
    def write_metrics_from_lists(new_file, y_pred, y_real, y_score, name):
        metrics = "\nAcuracity: {0}.\nPrecision: {1}.\nF1: {2}.\nRecall: {3}.\n----;\n"

        new_file.write("*---\n")
        new_file.write("{0} Classifier\n".format(name))

        cm = confusion_matrix(y_real, y_pred)
        text = str(cm).replace(' ', ',').replace(',,', ',').replace('[,', '[') + "\n"
        new_file.write(text)

        if y_score:
            try:
                fpr, tpr, _ = roc_curve(y_real, y_score)
                aucroc = auc(fpr, tpr)
                new_file.write("AUC: {0}".format(aucroc))
            except Exception as e:
                print(e)
                pass

        if y_score:
            acuracity = accuracy_score(y_real, y_pred)
            precision = precision_score(y_real, y_pred)
            f1 = f1_score(y_real, y_pred)
            recall = recall_score(y_real, y_pred)
        else:
            acuracity = accuracy_score(y_real, y_pred, normalize=True)
            precision = precision_score(y_real, y_pred, average='micro')
            f1 = f1_score(y_real, y_pred, average='micro')
            recall = recall_score(y_real, y_pred, average='micro')

        new_file.write(metrics.format(acuracity, precision, f1, recall))
        return acuracity

    @staticmethod
    def calc_roc_aucs(y_real, y_score):
        y_test = label_binarize(y_real, classes=[-1, 0, 1])
        aucs = []
        for i in range(3):
            fpr, tpr, _ = roc_curve(y_test[:, i], y_score[:, i])
            aucs.append(auc(fpr, tpr))
        return aucs

    @staticmethod
    def add_binari_label_to_list(target_label, label_predicted, plist):
        if str(label_predicted) == target_label:
            return plist.append(1)
        else:
            return plist.append(0)

    @staticmethod
    def calc_aucs(cm):
        tp_0 = cm[0][0]
        fn_0 = cm[0][1] + cm[0][2]
        fp_0 = cm[1][0] + cm[2][0]
        tn_0 = cm[1][1] + cm[1][2] + cm[2][1] + cm[2][2]

        tp_1 = cm[1][1]
        fn_1 = cm[1][0] + cm[1][2]
        fp_1 = cm[0][1] + cm[2][1]
        tn_1 = cm[0][0] + cm[0][2] + cm[2][0] + cm[2][2]

        tp_2 = cm[2][2]
        fn_2 = cm[2][0] + cm[2][1]
        fp_2 = cm[0][2] + cm[1][2]
```

```
    tn_2 = cm[0][0] + cm[0][1] + cm[1][0] + cm[1][1]

    # TPR = TP/(TP+FN)
    tpr_0 = tp_0 / (tp_0 + fn_0)
    tpr_1 = tp_1 / (tp_1 + fn_1)
    tpr_2 = tp_2 / (tp_2 + fn_2)

    # FPR = FP/(FP+TN)
    fnr_0 = fp_0 / (fp_0 + tn_0)
    fnr_1 = fp_1 / (fp_1 + tn_1)
    fnr_2 = fp_2 / (fp_2 + tn_2)

    auc0 = auc(fnr_0, tpr_0)
    auc1 = auc(fnr_1, tpr_1)
    auc2 = auc(fnr_2, tpr_2)
    return auc0, auc1, auc2

def retallar_dataset(self, y_train, y_test, class2, class1, class0):
    train_2_indices = np.array(np.where(y_train.values == class2)).tolist()[0]
    train_1_indices = np.array(np.where(y_train.values == class1)).tolist()[0]
    train_0_indices = np.array(np.where(y_train.values == class0)).tolist()[0]

    ordered_indices = sorted([train_2_indices, train_1_indices, train_0_indices], key=len)
    shortest = ordered_indices[0]
    shortest_len = len(shortest)
    for lindices in ordered_indices[1:]:
        random_train_selection = random.sample(lindices, shortest_len)
        shortest.extend(random_train_selection)
    random.shuffle(shortest)

    y_train_copy = y_train.iloc[shortest]

    test_2_indices = np.array((np.where(y_test.values[:] == class2))).tolist()[0]
    test_1_indices = np.array((np.where(y_test.values[:] == class1))).tolist()[0]
    test_0_indices = np.array((np.where(y_test.values[:] == class0))).tolist()[0]

    ordered_indices = sorted([test_2_indices, test_1_indices, test_0_indices], key=len)
    shortest = ordered_indices[0]
    shortest_len = len(shortest)
    for lindices in ordered_indices[1:]:
        random_train_selection = random.sample(lindices, shortest_len)
        shortest.extend(random_train_selection)
    random.shuffle(shortest)

    y_test_copy = y_test.iloc[shortest]

    return y_train_copy, y_test_copy
```

final_view.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Glucose Predicter</title>

    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
    <style>
        .arrow-up {
          width: 0;
          height: 0;
          border-left: 20px solid transparent;
          border-right: 20px solid transparent;

          border-bottom: 20px solid #cc0000;
            float: left;
        }

        .arrow-down {
          width: 0;
          height: 0;
          border-left: 20px solid transparent;
          border-right: 20px solid transparent;

          border-top: 20px solid #000000;
            float: left;
        }

        .square {
            width: 20px; height: 20px; background: green;
            float: left;
        }
    </style>
</head>
<body>
<script>
    window.onload = function(){
          show_vocabulary();
          show_vocabulary();
          show_seq();
          show_seq();
    };
    function home() {
        window.location.href = "/"
    }
    function show_vocabulary() {
        var elem = document.getElementById("myDiv");
        var state  = elem.style.display;
        if (state === "block") {
          elem.style.display = "none";
        }else {
            elem.style.display = "block";
        }
    }
    function show_seq() {
        var elem = document.getElementById("myDiv2");
        var state  = elem.style.display;
        if (state === "block") {
          elem.style.display = "none";
        }else {
            elem.style.display = "block";
        }
    }
</script>
<nav class="navbar navbar-default navbar-static-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Hyper/Hypo Predicter</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
            </ul>
        </div>
    </div>
</nav>
<div class="container">
    <h2>Results</h2>
    <hr>
    <div class="row">
        <h3>Prediction</h3>
        <div class="row">
            <div class="col-xs-4"><h4>Word predicted: {{ params.res }}.</h4></div>
            {% if params.label == -1 %}
                <div class="col-xs-1"> <div class="arrow-down"></div></div>
                <div class="col-xs-6"><h3 style="margin-left: 25px;margin-top: -5px;">Hypoglycemia   </h3>
            {% elif params.label == 1%}
                <div class="col-xs-1"> <div class="arrow-up"></div></div>
                <div class="col-xs-6"><h3 style="margin-left: 25px;margin-top: -5px;">Hyperglycemia   </h3>
            {% else %}
                <div class="col-xs-1"> <div class="square"></div></div>
                <div class="col-xs-6"><h3 style="margin-left: 25px;margin-top: -5px;">Normal   </h3>
            {% endif %}
            (Confidence: {{ params.lconfidence }})</div>
```

```
        </div>
        <div class="row">
            <div class="col-xs-12"> {{ params.result_img|safe }}</div>
        </div>
    </div>

    <div class="row">
        <h3>Full Sequence</h3>
        {% for i in params.sequences_data %}
            <div  class="col-xs-3">
                <div class="row"> {{ i[0]|safe }}</div>
                <div class="row">
                    {% if i[1] == -1 %}
                        <div class="col-xs-1"> <div class="arrow-down"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Hypoglycemia</h3></div>
                    {% elif i[1] == 1%}
                        <div class="col-xs-1"> <div class="arrow-up"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Hyperglycemia</h3></div>
                    {% else %}
                        <div class="col-xs-1"> <div class="square"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Normal</h3></div>
                    {% endif %}
                </div>
            </div>
        {% endfor %}
    </div>


    <div class="row" style="margin-top: 30px;margin-bottom: 30px;">
            <div class="btn-group col-xs-4" style="float: right; clear: right; margin-bottom: 20px;">
                <button class="btn btn-success col-xs-10" onclick="home();"><span>Continue</span></button>
            </div>
    </div>

    <div class="row" style="margin-top: 30px;margin-bottom: 30px;">
            <div class="btn-group col-xs-4" style="float: left; margin-bottom: 20px;">
                <button class="btn btn-info col-xs-10" onclick="show_vocabulary();"><span>Show/Hide Vocabulary</span></button>
            </div>
            <div class="btn-group col-xs-4" style="float: right; clear: right; margin-bottom: 20px;">
                <button class="btn btn-info col-xs-10" onclick="show_seq();"><span>Show/Hide Similar Sequences</span></button>
            </div>
{#            <div class="btn-group col-xs-4" style="float: right; clear: right; margin-bottom: 20px;">#}
{#                <form method=get enctype=multipart/form-data action="{{ params.route }}">#}
{#                        <input class="btn btn-success col-xs-10" type=submit value=Continue>#}
{#                </form>#}
{#            </div>#}
    </div>
</div>
<div class="container">
    <div id="myDiv2">
        <h2>Similar Sequences</h2>
        <hr>
        {% for seq in params.similar_seq %}
            <div class="row">
                <div class=" col-xs-4"> <h4>{{ seq[0] }}</h4></div>
                <div class=" col-xs-4"> <h4>Similarity: {{ seq[1] }}</h4></div>
            </div>
        {% endfor %}
        <br>
        <hr>
        <div class="btn-group col-xs-4" style="float: left; clear: both;margin-bottom: 20px;">
            <button class="btn btn-info col-xs-10" onclick="show_seq();"><span>Show/Hide Similar Sequences</span></button>
        </div>
    </div>
</div>
<div class="container">
    <div id="myDiv">
        <h2>Vocabulary</h2>
        <hr>
        {% for word in params.vocabulary %}
            <div class=" col-xs-4"> {{ word|safe }}</div>
        {% endfor %}
        <br>
        <hr>
        <div class="btn-group col-xs-4" style="float: left; clear: both;margin-bottom: 20px;">
            <button class="btn btn-info col-xs-10" onclick="show_vocabulary();"><span>Show/Hide Vocabulary</span></button>
        </div>
    </div>
</div>
</body>
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Glucose Predicter</title>

    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
</head>
<body id="main">
<script>
    function submit() {
        var inputFileImage = document.getElementById("fileToUpload");
        var file = inputFileImage.files[0];
        var data = new FormData();
        data.append('arxiu', file);
        $("#loader_message").css("display", "block");
        $.ajax({
            url:'/file',
            type:'POST',
            contentType:false,
            data:data,
            processData:false,
            cache:false,
            success: function(response) {
                var values = {};
                $('input').each(function() {
                    var id = $(this).attr('id');
                    values[id] =  $(this).val();
                });
                $('select').each(function() {
                    var id = $(this).attr('id');
                    values[id] =  $(this).val();
                });
                var myJsonString = JSON.stringify(values);
                console.log(myJsonString);
                $.ajax({
                    type: 'POST',
                    data: myJsonString,
                    url: '/session',
                    contentType: "application/json",
                    method: 'POST',
                    success: function(response) {
                        var sid = values["pacient"]+"_"+values["date"];
                        sid = sid.replace(" ", "_");
                        window.location.href = "/session/"+sid;
                    },
                    error: function(error) {
                        $("#loader_message").css("display", "none");
                        show_error(error);
                    }
                });
            },
            error: function(error) {
                $("#loader_message").css("display", "none");
                show_error(error);
            }
        });
    }

    function show_error(error) {
        var str = jQuery.parseJSON(error.responseText)['message'];
        console.log(str);
        $('#error_text').text("Error:  " + str);
        $("#error_div").show().delay(3000).fadeOut();

    }
</script>
<nav class="navbar navbar-default navbar-static-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Hyper/Hypo Predicter</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
            </ul>
        </div>
    </div>
</nav>
<div class="container">
    <div class="jumbotron">
        <h3>Upload raw data obtained from sensor</h3>
        <hr>
        <div class="form-horizontal">
            <div class="form-group">
                <label class="control-label col-xs-2" for="fileToUpload">File to upload:</label>
                <div class="col-xs-3">
                    <input type="file" id="fileToUpload">
                </div>
            </div>
            <hr>
            <div class="form-group">
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="pacient">Patient:</label>
```

```
                    <div class="col-xs-7">
                        <select id="pacient" class="form-control">
                            {% for pacient in params.pacients %}
                                <option>{{ pacient }}</option>
                            {% endfor %}
                        </select>
                    </div>
                </div>
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="date">Date:</label>
                    <div class="col-xs-7">
                        <input id="date" type="text" class="form-control" value="{{ params.time }}"> YYYY-MM-DD HH:MM
                    </div>
                </div>
            </div>
            <div class="form-group">
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="insulin">Insulin:</label>
                    <div class="col-xs-7">
                        <input id="insulin" type="number" step="0.01" class="form-control" value="0.00">
                    </div>
                </div>

                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="carboh">Carbohydrates:</label>
                    <div class="col-xs-7">
                        <input id="carboh" type="number" step="0.01" class="form-control" value="0.00">
                    </div>
                </div>
            </div>
            <div class="form-group">
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="exerciseBf">Exercice Before:</label>
                    <div class="col-xs-7">
                        <select id="exerciseBf" class="form-control">
                            <option>No</option>
                            <option>Yes</option>
                        </select>
                    </div>
                </div>
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="exerciseAf">Exercice After:</label>
                    <div class="col-xs-7">
                        <select id="exerciseAf" class="form-control">
                            <option>No</option>
                            <option>Yes</option>
                        </select>
                    </div>
                </div>
            </div>
            <div class="form-group">
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="alcohol">Alcohol:</label>
                    <div class="col-xs-7">
                        <select id="alcohol" class="form-control">
                            <option>No</option>
                            <option>Yes</option>
                        </select>
                    </div>
                </div>
                <div class="col-xs-5">
                    <label class="control-label col-xs-5" for="insulin">Glucose:</label>
                    <div class="col-xs-7">
                        <input id="glucose" type="number" step="0.01" class="form-control" value="0.00">
                    </div>
                </div>
            </div>
        </div>
        <hr>
        <div class="form-group">
            <div class="col-xs-8"></div>
            <div class="btn-group col-xs-2">
                <button class="btn btn-success col-xs-10" onclick="submit();"><span>Continue</span></button>
            </div>
        </div>
    </div>


    <div class="alert alert-danger fade in" id="error_div" style="display: none; position: absolute; z-index: 3">
        <div id="error_text"></div>
    </div>

    <div class="alert alert-info fade in" id="loader_message" style="display: none; position: absolute; z-index: 3">
        <strong>Loading</strong> Please wait.
    </div>



</div>
<script src="/static/js/bootstrap.min.js"></script>
<script src="/static/js/jquery.js"></script>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
There is a problem with the server.
Maybe the databes is not well inizialitzated.

</body>
</html>
```

sequence_view.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Glucose Predicter</title>

    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
    <style>
        .arrow-up {
          width: 0;
          height: 0;
          border-left: 20px solid transparent;
          border-right: 20px solid transparent;

          border-bottom: 20px solid #cc0000;
            float: left;
        }

        .arrow-down {
          width: 0;
          height: 0;
          border-left: 20px solid transparent;
          border-right: 20px solid transparent;

          border-top: 20px solid #000000;
            float: left;
        }

        .square {
            width: 20px; height: 20px; background: green;
            float: left;
        }
    </style>
</head>
<body>
<script>
    window.onload = function(){
          show_vocabulary();
          show_vocabulary();
    };
    function show_vocabulary() {
        var elem = document.getElementById("myDiv");
        var state  = elem.style.display;
        if (state === "block") {
          elem.style.display = "none";
        }else {
            elem.style.display = "block";
        }
    }
</script>
<nav class="navbar navbar-default navbar-static-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Hyper/Hypo Predicter</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
            </ul>
        </div>
    </div>
</nav>
<div class="container">
    <h2>Sequence detected</h2>
    <hr>
    <div class="row">
        {% for i in params.sequences_data %}
            <div  class="col-xs-4">
                <div class="row"> {{ i[0]|safe }}</div>
                <div class="row">
                    {% if i[1] == -1 %}
                        <div class="col-xs-1"> <div class="arrow-down"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Hypoglycemia</h3></div>
                    {% elif i[1] == 1%}
                        <div class="col-xs-1"> <div class="arrow-up"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Hyperglycemia</h3></div>
                    {% else %}
                        <div class="col-xs-1"> <div class="square"></div></div>
                        <div class="col-xs-1"><h3 style="margin-left: 25px;margin-top: -5px;">Normal</h3></div>
                    {% endif %}
                </div>
            </div>
        {% endfor %}
    </div>
{#    <div class="row">#}
{#            {% if params.type == -1 %}#}
{#                <div class="col-xs-1"> <div class="arrow-down"></div></div>#}
{#                <div class="col-xs-1"><h3 style="margin-top: -10px;">Hypoglycemia</h3></div>#}
{#            {% elif params.type == 1%}#}
{#                <div class="col-xs-1"> <div class="arrow-up"></div></div>#}
{#                <div class="col-xs-1"><h3 style="margin-top: -10px;">Hyperglycemia</h3></div>#}
{#            {% else %}#}
{#                <div class="col-xs-1"> <div class="square"></div></div>#}
{#                <div class="col-xs-1"><h3 style="margin-top: -10px;">Normal</h3></div>#}
```

```
{#            {% endif %}#}
{#        <hr style="clear: both; margin-top: 10px;">#}
{#    </div>#}
    <div class="row" style="margin-top: 30px;margin-bottom: 30px;">
{#        <div class="jumbotron">#}
            <div class="btn-group col-xs-4" style="float: left; margin-bottom: 20px;">
                <button class="btn btn-info col-xs-10" onclick="show_vocabulary();"><span>Show/Hide Vocabulary</span></button>
            </div>
            <div class="btn-group col-xs-4" style="float: right; clear: right; margin-bottom: 20px;">
                <form method=get enctype=multipart/form-data action="{{ params.route }}">
                        <input class="btn btn-success col-xs-10" type=submit value=Continue>
                </form>
            </div>
{#        </div>#}
    </div>
</div>
<div class="container">
    <div id="myDiv">
        <h2>Vocabulary</h2>
        <hr>
        {% for word in params.vocabulary %}
            <div class=" col-xs-4"> {{ word|safe }}</div>
        {% endfor %}
        <br>
        <hr>
        <div class="btn-group col-xs-4" style="float: left; clear: both;margin-bottom: 20px;">
            <button class="btn btn-info col-xs-10" onclick="show_vocabulary();"><span>Show/Hide Vocabulary</span></button>
        </div>
    </div>
</div>
</body>
</html>
```

session_view.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Glucose Predicter</title>

    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
<script>
beforeSubmit = function(){
        document.getElementById("loader_message").style.display = "block";
        console.log("JA S'HAURIA DE VEURE");
        document.getElementById("formid").submit();
    }
</script>
<nav class="navbar navbar-default navbar-static-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Hyper/Hypo Predicter</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
            </ul>
        </div>
    </div>
</nav>
<div class="container">
    <div class="jumbotron">
        {{ params.image|safe}}
        <br>
        {% for i in params.hist %}
            <div  class="col-xs-4">
                <div class="row"> {{ i|safe }}</div>
            </div>
        {% endfor %}
        <hr>
        <div class="btn-group col-xs-2" style="float: right; clear: both; margin-bottom: 20px;">
            <form method=post enctype=multipart/form-data action="/predicter" id="formid">
                <input class="btn btn-success col-xs-10" type=submit value=Continue onclick="beforeSubmit();">
                <div style="display: none;">
                    <input type=text name=session_id value="{{ params.id }}">
                </div>
            </form>
        </div>
        <div class="alert alert-info fade in" id="loader_message" style="display: none; position: absolute; z-index: 3">
            <strong>Loading</strong> Please wait.
        </div>
    </div>
</div>
</body>
</html>
```

requirements.txt

```
flask
werkzeug
pandas
matplotlib
sklearn
numpy
plotly
lmfit
```