Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Mòdul de predicció per al consum energètic

Document: Memòria

Alumne: Àlex López Diaz

Tutor: Llorenç Burgas
Departament: EEIA
Àrea: ENGINYERIA DE SISTEMES I AUTOMÀTICA

Convocatòria (mes/any): Setembre/2021

0. Taula de continguts

0.1. Índex de figures

1. Introducció i objectius del projecte

1.1 Introducció

El *forecasting* de sèries de temporals basat en tècniques d'intel·ligència artificial és una línia de recerca d'interès dins de diversos sectors, un d'ells és la demanda energètica d'edificis. Ja que si som capaços de preveure quin consum tindrem, serem capaços d'actuar en l'emmagatzematge d'energia de forma més òptima. A l'estat de l'art existeixen moltes tècniques i metodologies diferents per realitzar *forecastings*.

1.2 Objectius

L'objectiu d'aquest treball és elaborar una eina de *forecasting* autònoma que permeti entrenar els models de manera que sigui el màxim d'autònoma possible, emmagatzemar-los i posteriorment utilitzar-los per obtenir previsions d'energia consumida. En aquest treball s'implementarà un mòdul que s'integrarà dins la plataforma software del projecte europeu e-Land.

En aquest treball es provaran i utilitzaran diverses tècniques d'intel·ligència artificial per aconseguir els millors resultats de *forecasting* possibles. S'utilitzaran tècniques d'autoselecció de paràmetres per als diversos algorismes de *forecasting* seleccionats i es gestionarà la possibilitat de tenir diversos models / usuaris i el seu emmagatzematge. Finalment es testejarà la metodologia implementada en un cas d'ús utilitzant dades reals d'un dels 3 pilots del projecte e-Land.

El projecte es realitzarà amb la col·laboració del grup de recerca "eXiT" afiliat a la UdG, en la seva branca enfocada a l'eficiència i disminució en el consum energètic, més concretament, el treball es durà a terme amb en Llorenç Burgas, tutor del TFG. Tots els *datasets* i les dades reals utilitzades en aquest projecte són proporcionades pel grup de recerca.

1.3 Origen i motivacions personals

El projecte s'origina en base a la proposta del tutor d'incorporar al projecte e-Land una eina que permetés realitzar *forecastings* del consum d'energia consumida de qualsevol registre de dades (d'una empresa, un edifici, un port...).

Personalment, estic molt interessat en tots els temes i totes les tecnologies que engloben la intel·ligència artificial, per això em vaig interessar per les propostes d'aquest grup de recerca, que encara problemes complexes utilitzant la IA. Durant la carrera he tingut la sensació que no es dona molta importància a aquest àmbit, comprovant que hi ha poques assignatures on es realitzin activitats relacionades amb els tests estadístics i algorismes comuns a la IA. Per tant, és una oportunitat, una excusa, per poder endinsar-me en aquest món i aprendre a utilitzar algorismes predictors en un ambient on s'usen aquestes eines diàriament.

Un altre punt a favor a realitzar aquest projecte és el tòpic del consum energètic, la minimització de l'energia consumida i la cura mediambiental, que és un tema actual i de preocupació mundial. Si podem ajudar a reduir el consum energètic amb aquesta eina, així es farà.

2. Estudi de viabilitat

2.1. Recursos necessaris per desenvolupar el projecte

El desenvolupament del treball no ha requerit una gran quantitat de recursos, només requereix dels necessaris per poder generar els models, entrenar-los i emmagatzemar-los. Un ordinador convencional pot realitzar totes les tasques del projecte sense cap dificultat.

De totes maneres, és convenient llistar les característiques hardware i software del dispositiu on s'ha dut a terme el projecte.

Característiques hardware:

- **CPU**. AMD Ryzen 5 3600 3.6GHz.
- **RAM.** Kingston HyperX Fury Black 16GB DDR4 3200Mhz .
- **Disc dur**. Kioxia EXCERIA 480GB SSD SATA.
- **Targeta gràfica**. Gigabyte GeForce RTX 2060 OC 6GB GDDR6.

Característiques del software:

- **Sistema operatiu**. Windows 10 Pro.
- **Llenguatge de programació**. Python 3.8 o posterior (requisit obligatori).
- **Llibreries Python**. *Pandas, Sklearn i Tensorflow* entre d'altres (veure procés d'instal·lació al final del projecte per veure detalladament totes les llibreries, apartat 15).

Per finalitzar aquest apartat, comentar que és necessari entrar les dades en format .csv, amb les següents dades mínimes: una columna amb les mesures de consum d'energia i una columna ('índex') on s'especifiqui a quin dia i hora correspon cada mesura.

Totes les dades utilitzades en aquest projecte ha estat proporcionades pel grup de recerca eXiT.

2.2. Recursos humans

Els recursos humans requerits en aquest projecte són d'una persona, l'única que està realitzant aquest projecte, jo mateix.

Per poder mesurar el cost del projecte, tindrem el compte que estic treballant com a una persona jurídica, per tant, el cost humà seria el sou en hores que jo rebria per realitzar el projecte.

Suposarem un preu per hora d'un enginyer informàtic mig de 13 euros per hora. Si aproximadament s'han dedicat al projecte unes 290 hores, el preu final del projecte seria de 3.770 euros.

2.3. Viabilitat econòmica

El cost global del projecte constaria de la suma dels punts esmentats anteriorment: el cost del hardware, el cost del software y els dels recursos humans.

La suma final també pot dependre dels recursos inicials dels quals es disposi, per tant, suposarem que partim de zero. Sumem:

- El cost del hardware dependria de l'ordinador a comprar. Suposem que volem comprar l'ordinador especificat a l'apartat 2.1. El cost global de l'ordinador seria aproximadament d'uns 950 euros.
- El software és en la seva totalitat Open Source. Tant Python com les seves llibreries són gratuïtes.
- Els recursos humans, indicats en l'apartat 2.2, són de 3.770 euros.

El cost total del projecte seria aproximadament de 4.720 euros. Si ja es disposés d'un ordinador, no seria necessari invertir en comprar-ne un de nou i el cost total resultant només seria el cost dels recursos humans requerits, 3.770 euros.

2.4. Viabilitat tecnològica

Posant en perspectiva el que hem vist fins ara, podem concloure que el projecte és viable tecnològicament parlant.

Els recursos de hardware necessaris per a la realització de totes les tasques del treball es poden dur a terme en un ordinador amb característiques mitjanament acceptables (clarificar que es necessita un mínim, un ordinador amb components poc actuals podria tenir problemes). Busquem un equip amb un nivell de còmput mitjà.

Pel que fa al software, utilitzo Python, un llenguatge potent i actual, que posseeix una quantitat enorme de llibreries dedicades a la intel·ligència artificial. Utilitzo la llibreria *Sklearn* que és la més extensa i fiable del mercat en l'àmbit del *Machine Learning*, i Tensorflow que també és molt competent dins l'aprenentatge automàtic.

3. Metodologia

La metodologia/planificació que he volgut seguir des d'un bon principi ha sigut la metodologia SCRUM, deixant-me influenciar per assignatures de la carrera en les que hem seguit aquesta metodologia; i pel lloc de treball que actualment ocupo, que també s'organitzen d'aquesta manera.

En el meu cas no serà cent per cent efectiva, ja que aquesta metodologia està pensada per organitzar grups de treball, però també serveix per analitzar (en períodes de temps no molt llargs) el treball realitzat i detectar possibles desviacions.

La metodologia SCRUM reparteix dels projectes en *Sprints*. Els *Sprints* són períodes de temps fixes en els quals es volen realitzar un cert nombre de tasques. Les tasques es defineixen al principi de cada *Sprint*, es desenvolupen durant la duració de l'etapa i al final de cada un d'ells es repassa si s'han assolit els objectius, valorant el ritme de feina i la quantitat de tasques realitzades.

La planificació d'aquest projecte és la de realitzar *Sprints* d'1-2 setmanes. Les etapes no poden ser molt extenses, ja que es pot arribar a perdre l'objectiu de cada iteració.

Cal comentar que, per raons personals, no s'ha pogut seguir aquesta metodologia al peu de la lletra en alguns *Sprints*, tal com podeu veure en l'explicació de l'apartat següent.

4. Planificació

Aquest projecte es va iniciar el 29 de març, passada una setmana de l'aprovació del full del projecte. I s'ha acabat l'1 de setembre, coincidint amb la segona entrega disponible.

Per raons personals vaig haver d'aplaçar l'entrega del projecte.

Figura 1: Planificació del projecte.

He seguit la metodologia SCRUM i els *sprints* han quedat així:

- ***Sprint* 1**. El primer *sprint* és dedicat a investigar i aprendre sobre la regressió i aplicacions relacionades amb les sèries temporals. Aquesta fase s'ha expandit en els següents *sprints*, ja que no es deixa d'investigar sobre aquests temes.
- ***Sprint* 2**. S'han proporcionat les dades d'entrada al model. S'han d'interpretar i modificar per poder-les tractar durant el projecte.
- ***Sprint* 3**. Investigació i decisió sobre els models predictors aplicats a les sèries temporals. L'objectiu és informar-se sobre els millors models predictors i aplicar-los al projecte.

- ***Sprint* 4**. Muntatge de l'estructura de dades dins els models. Com emmagatzemem les dades a cada model.
- ***Sprint* 5**. Represa del projecte; veure què està fer i implementar el primer model predictor del sistema, SVR.
- ***Sprint* 6**. Implementació del SARIMAX.
- ***Sprint* 7**. Proves dels dos models implementats. Al veure els resultats es planteja l'opció afegir un nou model al sistema.
- ***Sprint* 8**. Implementació LSTM.
- ***Sprint* 9**. Simulació d'un cas d'ús real on entrem dades periòdicament al nostre sistema.
- ***Sprint* 10**. Redacció de la memòria.

No s'ha seguit estrictament els *sprints* de la mateixa llargada. Cal dir que un cop acabat l'objectiu de cada iteració, a mig *sprint* es poden començar noves tasques.

5. Marc de treball i conceptes previs

5.1 Sèries temporals

Una sèrie temporal o cronològica és una seqüència de dades, observacions o valors mesurats en determinats moments del temps, ordenats cronològicament i, normalment, espaiats entre si de manera uniforme.

L'anàlisi de les sèries de dades temporals s'enfoca en poder-les interpretar extraient-ne informació representativa per predir el seu comportament futur o referent als orígens de les dades.

Un dels usos més habituals de les sèries de dades temporals és la seva anàlisi per a predicció i pronòstic; com, per exemple, de les dades climàtiques o de les accions de borsa. En el nostre cas ens interessa predir el consum d'energia futur.

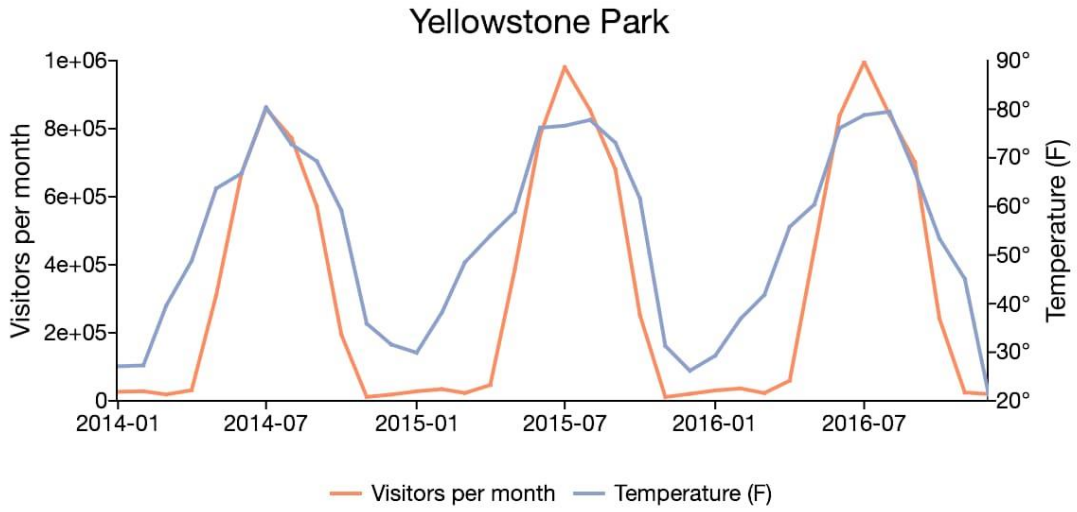


Figura 2: Exemple dades d'una sèrie temporal. Visitants per mes.

Per analitzar les sèries temporals podem definir les observacions obtingudes com la suma de les següents 4 variables:

- **Tendència secular o regular**: és un component de la sèrie que reflecteix l'evolució a llarg termini.

- **Variació estacional**: recull les oscil·lacions que es produeixen en aquests períodes de repetició diaris, setmanals, mensuals o anuals.
- **Variació cíclica**: és el component de la sèrie que recull les oscil·lacions periòdiques d'amplitud superior a un any.
- **Variació aleatòria**: observacions que no mostren cap regularitat.

5.2 Intel·ligència artificial

La intel·ligència artificial (IA) és una àmplia branca de la informàtica dedicada a construir màquines intel·ligents. Els llibres líders en intel·ligència artificial, defineixen aquesta com a l'estudi d'agents intel·ligents (AI).

Un agent intel·ligent és qualsevol cosa que percep el seu entorn, fa accions de manera autònoma per assolir objectius i pot millorar el seu rendiment amb l'aprenentatge o pot racionalitzar.

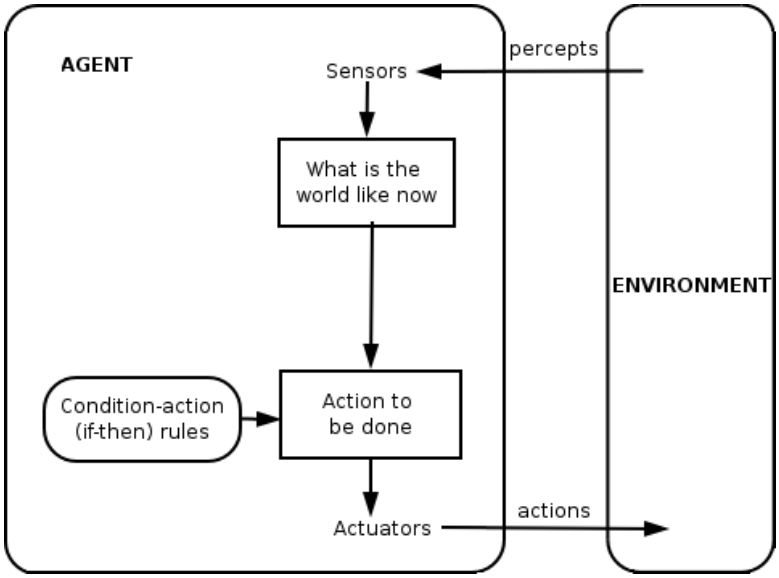


Figura 3: Funcionament d'un agent intel·ligent.

En la il·lustració anterior queda reflectit el funcionament iteratiu d'un agent intel·ligent. Primerament, rep informació de l'entorn a través d'uns "sensors", analitza què ha canviat i com està el món actualment; a continuació, determina quines accions ha de dur a terme i finalment indica als actuadors quina és l'acció a realitzar en l'entorn.

Per tant, definim una aplicació d'intel·ligència artificial com "*qualsevol sistema que percebi el seu entorn i faci accions que maximitzin les seves possibilitats d'assolir els seus objectius"* [3].

5.2.1 Machine Learning

Un subconjunt dins la intel·ligència artificial és el *Machine Learning* (ML). Entenem com a *Machine Learning* l'estudi d'algorismes informàtics que poden millorar automàticament a través de l'experiència i de l'ús de les dades.

Els algorismes ML construeixen "models" basats en un conjunt de dades, anomenat "dades d'entrenament" (*training data*), que com el seu nom indica, s'utilitzen per entrenar el nostre model i realitzar prediccions i/o decisions tot i que no l'hàgim programat explícitament per fer aquestes prediccions en específic. Per comprovar la veracitat d'aquestes prediccions es comparen les prediccions obtingudes amb dades no entrades en el procés d'entrenament (*test data*).

En aquest treball ens centrarem principalment en aplicar algorismes *Machine Learning* en el desenvolupament de l'aplicació predictora de consum d'energia.

Existeixen diversos tipus d'algorismes d'aprenentatge automàtic segons allò que s'espera que el programa aprengui i segons la interacció que ha de tenir amb l'usuari:

- **Aprenentatge supervisat.** És una tècnica que permet extreure una funció predictora a partir de les dades d'entrenament. La funció generada és capaç de predir gràcies a les dades d'entrenament. Les dades d'entrenament han de constar de parelles de dades on la primera component és una variable d'entrada (dades de *training*) i la segona és de sortida (dades de *test*).
- **Aprenentatge no supervisat.** A diferència de l'aprenentatge supervisat, tot el procés es duu a terme sobre el conjunt de dades d'entrenament, sense tenir en compte les dades de sortida. És capaç de reconèixer patrons estadístics per predir.

- **Aprenentatge semisupervisat.** Combina l'aprenentatge supervisat i el no supervisat per poder predir de la manera adequada depenent de les dades d'entrada.
- **Aprenentatge per reforç.** Aquest algorisme aprèn únicament de l'entorn. La seva entrada és *feedback* o retroalimentació que obté del món exterior cada cop que realitza una acció. Es basa en el model "prova i error".

5.3 Model de gestió per al consum energètic

La nostra proposta en aquest projecte per al problema de la gestió del consum energètic és d'un sistema on tractarem cada dia de la setmana per separat. És a dir, tindrem un model independent per als dilluns, un per als dimarts, etc.

Per què ens interessa dividir el model? Principalment per detectar i separar valors anòmals que puguin sorgir en dies concrets, marcar una "seasonality" clara i obtenir uns valors més nets, ja que el consum energètic d'un edifici no serà el mateix per a un dimarts que per a un dissabte, al ser entre setmana i cap de setmana els valors seran molt distants.

El sistema seria similar al representat en el següent diagrama:

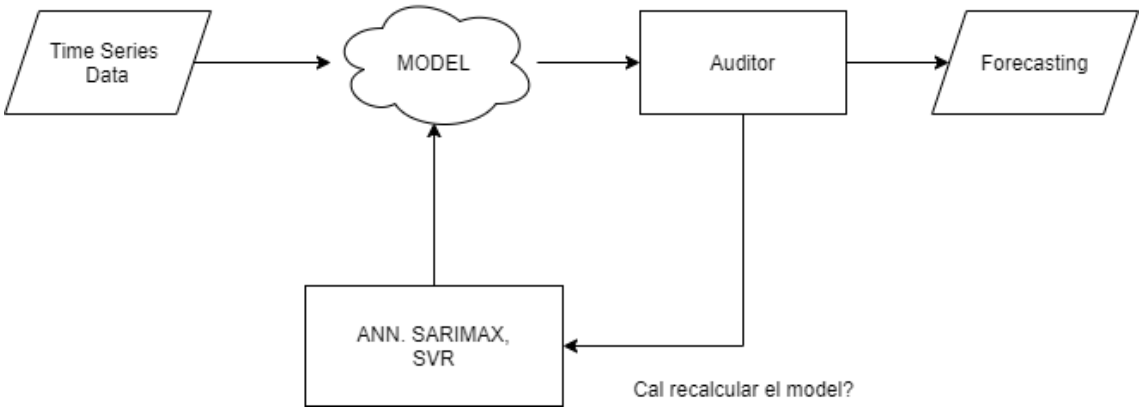


Figura 4: Model de gestió pel consum energètic.

Primerament, obtindrem una sèrie temporal de dades com a paràmetre d'entrada al sistema, de la qual generarem un primer model (dividint les dades per dies) i crearà la primera predicció del sistema. A continuació, entra en joc

una entitat (**auditor**) que determinarà si s'ha de recalcular el model o no, depenent de l'error que obtingui entre les noves dades entrades al model (dades reals) i la predicció generada pel model. Si l'error és prou significatiu, es recalcula el model.

Per fer més interessant la predicció, i la part més important de projecte, és que no ens centrarem en un sol algorisme predictor per obtenir les dades, sinó en aplicar-ne diversos i obtenir el millor resultat de tots ells.

Els algorismes a implementar són quatre:

- **ARIMA.** Acrònim de l'anglès *Autoregressive Integrated Moving Average*, és un model estadístic específic per predir sèries de dades temporals. És un mètode senzill però potent.
- **SVM.** Són un conjunt de mètodes d'aprenentatge supervisat utilitzats per classificació de dades, regressió i detecció de valors atípics. Utilitzen poca memòria i són molt ràpids.
- **Xarxa neuronal.** Una xarxa neuronal és un sistema informàtic inspirat en les xarxes neuronals biològiques que construeixen el cervell animal. Solen ser sistemes lents però obtenen bons resultats.
- **Model per mitjanes:** Aquest model és molt simple, serveix com a *back up* per si alguns dels algorismes anteriors falla, poder treure una predicció "decent". Consta de treure la mitjana del dia a predir de totes les dades guardades en el model.

A continuació s'explicarà detalladament cada un dels algorismes.

5.3.1 ARIMA

En estadística i econometria, en particular en sèries temporals, un model autoregressiu integrat de mitjana mòbil o ARIMA (acrònim de l'anglès *Autoregressive Integrated Moving Average*) és un model estadístic que utilitza variacions i regressions de dades estadístiques per tal de trobar patrons per a una predicció cap al futur. Es tracta d'un model dinàmic de sèries temporals, és a dir, les estimacions futures venen explicades per les dades del passat i no per variables independents.

El model ARIMA necessita identificar els coeficients i nombre de regressions que s'utilitzaran. Aquest model és molt sensible a la precisió amb què es determinin els seus coeficients.

Se sol expressar com ARIMA (p, d, q) on els paràmetres p, d i q són nombres enters no negatius que indiquen l'ordre de les diferents components del model:

- p: ordre d'autoregressió.
- d: ordre de diferència.
- q: ordre mitjà mòbil.

Quan algun dels tres paràmetres és zero, és comú ometre les lletres corresponents de l'acrònim. Per exemple, ARIMA (0,1,0) es pot expressar com I (1) i ARIMA (0,0,1) com a MA (1).

El model ARIMA (p, d, q) es pot representar com:

$$Y_t = -(\Delta^d Y_t - Y_t) + \phi_0 + \sum_{i=1}^{p} \phi_i \Delta^d Y_{t-i} - \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

Figura 5: Fórmula ARIMA.

La *Yt* són les dades de test, la *d* correspon a les *d* diferencies que són necessàries per convertir la sèrie original en estacionaria, *ϕ1... ϕp* són els paràmetres que pertanyen a la part autoregressiva del model, *ϕ0* és una constant, *θ1...θq* els paràmetres que pertanyen a les mitjanes mòbils del model i Et és l'error acumulat.

S'ha de tenir en compte que:

$$\Delta Y_t = Y_t - Y_{t-1}$$

Figura 6: Fórmula diferencia.

Aproximació per seasons (SARIMAX)

El model ARIMA es pot generalitzar encara més per considerar l'efecte de l'estacionalitat. En aquest cas, es parla d'un model SARIMA (en anglès *Seasonal Autoregressive Integrated Moving Average*).

Aquest nou model es representa com SARIMAX(p,d,q)x(P,D,Q)m

- P: ordre autoregressiu estacional.
- D: ordre de diferències estacionals.
- Q: ordre mitjà mòbil estacional.
- m: nombre de passos temporals per a un període estacional únic.

És important destacar que el paràmetre m influeix en els paràmetres P, D i Q. Per exemple, un m de 12 per a dades mensuals suggereix un cicle estacional anual.

Aquesta aproximació és l'escollida a implementar en aquest projecte referent a model ARIMA.

5.2.2 SVM

Les màquines de vectors de suport o màquines de vector suport (de l'anglès *Support Vector Machines*, SVM) són un conjunt d'algoritmes d'aprenentatge supervisat desenvolupats per Vladimir Vapnik i el seu equip en els laboratoris AT & T.

Una SVM construeix un hiperplà o un conjunt d'hiperplans que s'utilitzen per classificació, regressió i detecció de valors anòmals.

Un hiperplà és un conjunt de punts d'un espai de (n-1) dimensions d'un entorn d'n dimensions. Per exemple, donat un entorn unidimensional, una recta, un hiperplà seria un punt en aquella recta. En un entorn tridimensional, un hiperplà seria un pla bidimensional.

La funcionalitat principal d'un hiperplà dins les SVM és poder dividir les dades de les diferents classes de dades que puguin haver-hi. La seva aplicació més intuïtiva és la classificació, ja que exemplifica clarament el seu funcionament; suposant un model amb dues classes de dades, quan entra una nova mostra

en funció de l'espai que ocupa dins els hiperplans és classificada a una classe o a una altra.

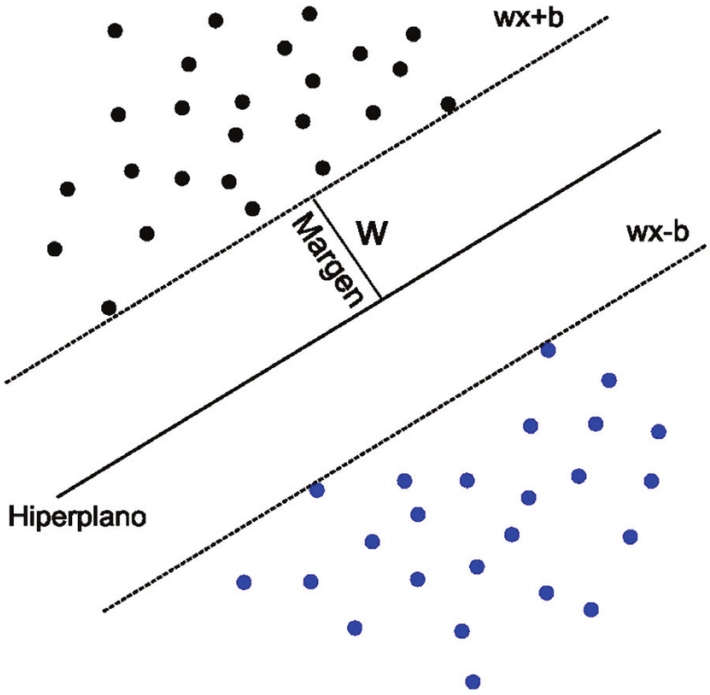


Figura 7: Exemple d'hiperplà.

El problema principal de l'SVM resideix en trobar la separació òptima dels hiperplans. En aquest tipus d'algorismes es busca la distància màxima (marge) amb els punts més propers a ell, d'aquesta forma tots els punts del vector que correspon a un costat de l'hiperplà és etiquetat d'una marera, i els de l'altre costat, d'una altra.

Aproximació per regressió (SVR)

En aquest projecte ens interessa més l'aproximació SVM enfocada a la regressió, ja que disposem de sèries temporals.

SVR o regressió de vectors de suport és la proposta de Vladimir Vapnik per a les màquines de vector de suport per a la regressió. El model produït per la classificació depèn únicament d'un *subset* de les dades d'entrenament, ja que el cost de la creació del model recau en els punts de l'espai més propers a l'hiperplà. De la mateixa manera, el cost del SVR depèn, també, d'un *subset* de les dades d'entrenament perquè la funció de cost per construir el model ignora qualsevol dada d'entrenament propera al model de predicció.

El funcionament de l'SVR s'explica amb:

$$\text{minimize } \frac{1}{2}\|w\|^2$$
$$\text{subject to } |y_i - \langle w, x_i \rangle - b| \leq \varepsilon$$

Figura 8: Formules SVR.

On la xi, yi són els valors de *training*. La w és el vector normal de l'hiperplà i l'objectiu en aquest cas és minimitzar el marge entre l'hiperplà i els valors, al contrari a la SVM per classificació, que busca maximitzar el marge. La predicció ve marcada per el producte vectorial més les dades interceptades *(w,xi) + b*. L'*E* és un valor lliure que marca que totes les prediccions han d'estar dins el llindar *0 < predicció < E*.

5.2.3 Xarxes neuronals

Definim una xarxa neuronal artificial (XNA), xarxa neuronal simulada o senzillament xarxa neuronal (denominada en anglès com "ANN", *Artificial Neuronal Network*) com:

"Un paradigma d'aprenentatge i processament automàtic inspirat en la forma en què funciona el sistema nerviós dels animals. Es tracta d'un sistema d'interconnexió de neurones en una xarxa que col·labora per produir un estímul de sortida." [23]

Les neurones o nodes són la mínima estructura dins el sistema neuronal. Cada node té una o més entrades i produeix una sola sortida que pot ser dirigida a múltiples neurones. La sortida dels nodes finals dels sistema (*output* final) és el resultat final del problema.

Un node pot rebre dades directament de les dades d'entrada com d'un altre node. Aquestes dades es coneixen més comunament com a "activacions". Cada entrada té associat una ponderació o pes (veure la Figura 8), que representa el grau d'influència que posseeix. Cada node calcula la suma ponderada de les entrades i, a continuació, aplica una funció d'activació

(s'explicarà a continuació) per determinar la sortida del node, així com el seu pes.

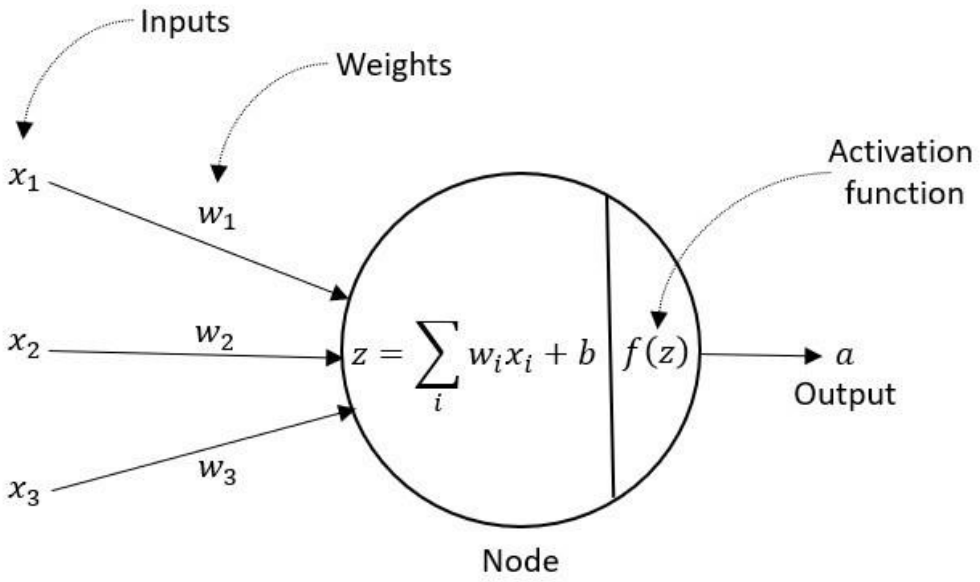


Figura 9: Funcionament intern d'una neurona.

La funció d'activació és una transformació no lineal dels valors d'entrada que és necessària per permetre el modelatge de tasques complexes encomanades al sistema. El sumatori de pesos

$$z = \sum_i w_i x_i + b$$

Figura 10: Sumatori de pesos.

és una transformació lineal, on el valor z es passa a la funció d'activació f(z), que és no linear. Les funcions d'activació poden ser de diversos tipus, les més comunes són les funcions sigmoide, la funció ReLU i la funció de tangent hiperbòlica:

- Sigmoide:

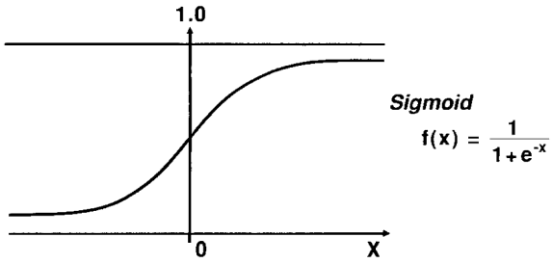


Figura 11: Funció sigmoide.

- Tangent hiperbòlica:

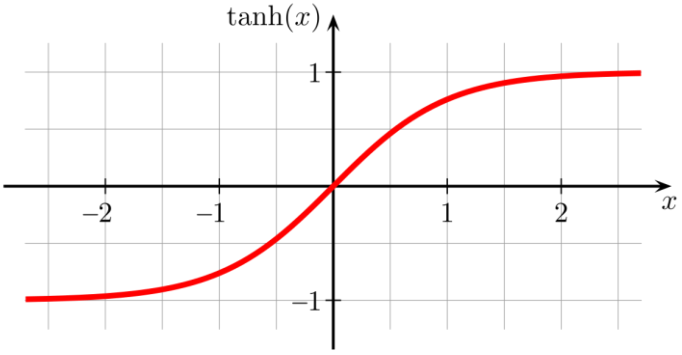


Figura 12: Funció tangent hiperbólica.

- Funció ReLU:

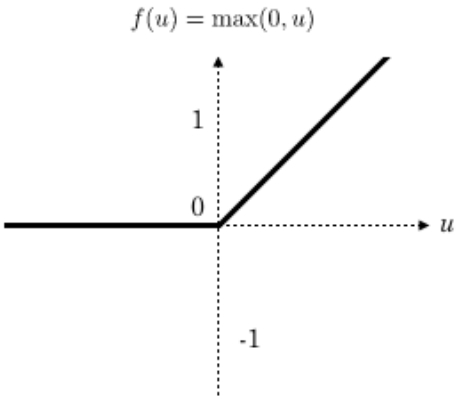


Figura 13: Funció ReLU.

En termes generals hi ha tres tipus de nodes: node d'entrada, node entremig o amagat (*hidden nodes*) o node de sortida. El seu tipus es determina en funció de la capa (en anglés *layer*) que ocupa, de les que prenen el mateix nom.

- Capa d'entrada.
- Capa amagada.
- Capa de sortida.

Hi poden haver tantes capes amagades com es vulgui (veure Figura 13), però un nombre elevat de capes pot empitjorar el rendiment, s'ha de trobar un bon equilibri.

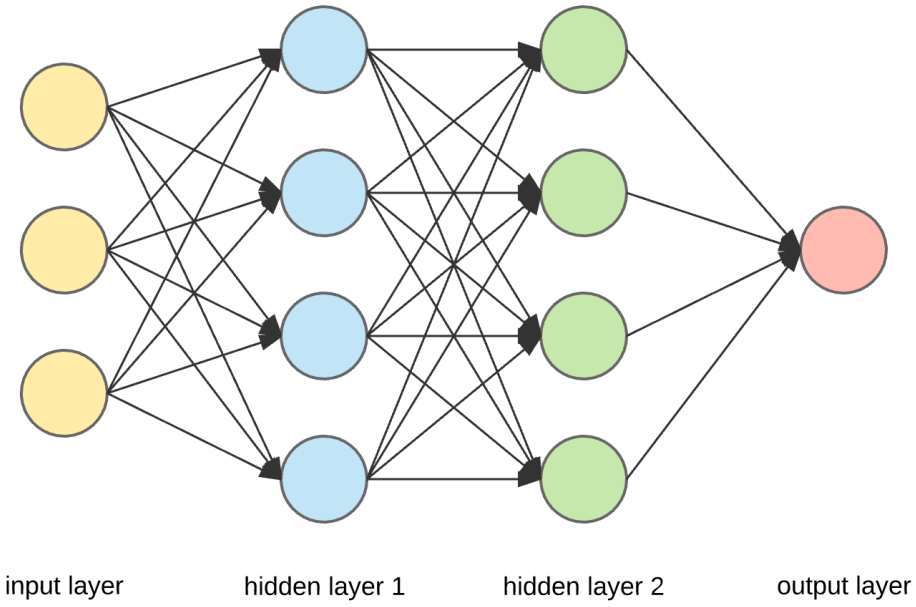


Figura 14: Estructura dins una xarxa neuronal directa.

Existeixen molts tipus d'algorismes sobre xarxes neuronals artificials, però per no fer el treball excessivament llarg comentarem per sobre l'algorisme aplicat.

Long short-term memory LSTM

El *Long Short-Term Memory* (acrònim LSTM en anglès) és una xarxa neuronal artificial recursiva (en anglès, *Recursive Neuronal Network*) que s'utilitza en el camp del *deep learning*. En comparació amb les xarxes neuronals directes (veure la Figura 13), els estats dels nodes són recurrents, és a dir, formen cicles entre les seves capes. S'apliquen en els camps de la classificació i predicció estadística sobre un conjunt de dades temporals.

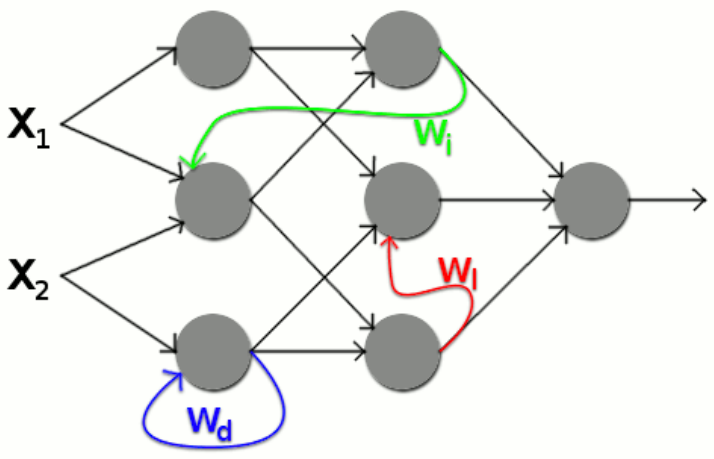


Figura 15: Estructura RNN.

La unitat bàsica LSTM està formada per una cèl·lula de memòria que conté una porta d'entrada, una porta de sortida i una porta d'oblit. També disposa d'un estat per comunicar-se entre les cel·les. Les cèl·lules tenen una forma similar a la següent:

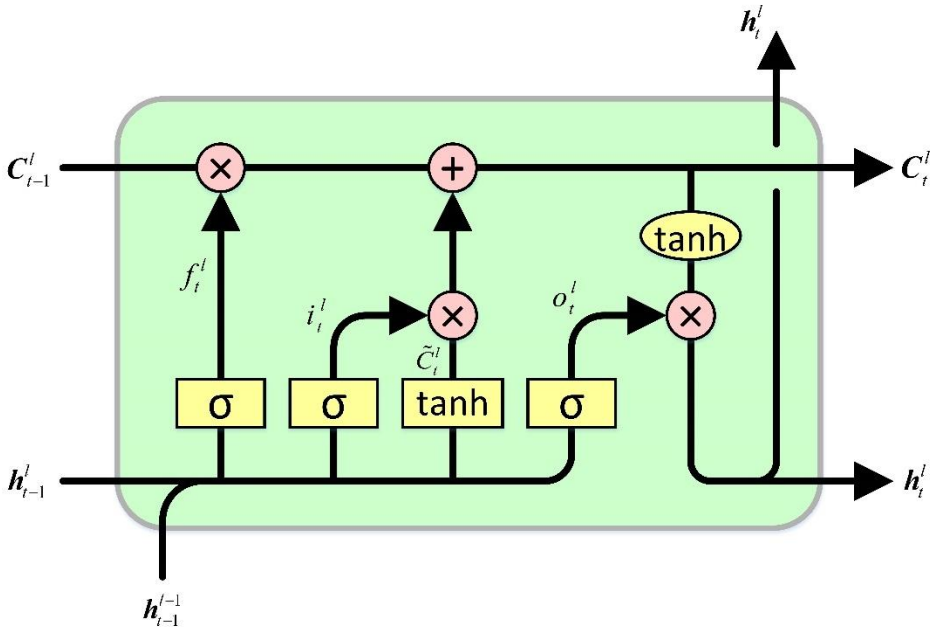


Figura 16: Cèl·lula LSTM.

No vull entrar molt en detall ja que, en definitiva, l'algorisme en si no el programem, sinó que utilitzarem l'algorisme LSTM dins la llibreria *Keras* (*Tensorflow*). Intentaré explicar el funcionament representat en la Figura 15.

Primerament, la primera funció (*σ1*) determina quina quantitat de les dades "passaran" per la cel·la. Es determina a partir de la funció sigmoidal amb les dades d'entrada (*hi-1*) un nombre entre 0 i 1 (dades a oblidar), on l'1 significa retenir totes les dades i 0 eliminar les dades. S'aplica a l'estat anterior (*Ct-1*).

A continuació, es passa per la segona fase (*σ2 i tanh*) on la funció sigmoidal i la funció tangencial creen un estat temporal encarregat d'actualitzar l'estat anterior.

Llavors, es calcula el nou estat *Ct* i les noves dades de sortida *ht*.

Per a més informació, veure l'article següent [16] on s'explica pas per pas l'execució d'una cel·la.

5.4 Dataset

Com ja s'ha comentat anteriorment en aquest projecte, les dades han sigut proporcionades pel tutor del TFG.

Comencem per veure el *dataset* amb nom "horari_p.csv":

	index	p_Edificio_1	p_Edificio_2	p_Edificio_3
0	2012-02-13 07:00:00	83923.333333	163788.033333	NaN
1	2012-02-13 08:00:00	81622.083333	187333.033333	NaN
2	2012-02-13 09:00:00	92052.500000	199666.150000	NaN
3	2012-02-13 10:00:00	90572.386667	209308.300000	NaN
4	2012-02-13 11:00:00	92103.750000	214241.650000	NaN
...	...	...	...	...
70218	2019-09-17 04:00:00	65700.000000	11300.000000	NaN
70219	2019-09-17 05:00:00	69700.000000	11166.666667	NaN
70220	2019-09-17 06:00:00	88750.000000	11300.000000	NaN
70221	2019-09-17 07:00:00	62750.000000	21266.666667	NaN
70222	2019-09-17 08:00:00	53500.000000	26933.333333	NaN

Figura 17: Estructura del dataset

Consta d'una columna 'index' amb la data exacta de la mostra de consum d'energia. S'especifica l'any, el mes, el dia, l'hora, els segons i els milisegons de la data en format 'yyyy-MM-dd hh:mm:ms'.

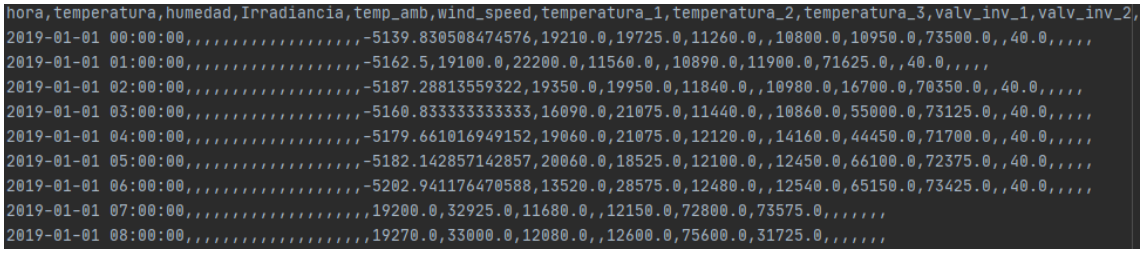
Tenim registrades les dades des del 2012 fins a mitjans del 2019. Cal remarcar que moltes de les dades són nul·les, i que les dades realment significatives per a realitzar la predicció són del 2018 i 2019.

El segueixen les dades del consum d'energia de diferents edificis. En aquest cas hi ha les dades de 3 edificis diferenciats amb els noms de 'p_Edificio_1', 'p_Edificio_2' i 'p_Edificio_3'. El *dataset* conté més informació, com la temperatura, la humitat, el vent, etc. però no ens interessen per a la nostra aplicació, ja que ens centrarem purament en el consum d'energia i no en les possibles variables extres com ara la pluja, el vent o la temperatura per fer les nostres prediccions.

Totes aquestes dades es troben recollides dins del projecte e-Land (https://elandh2020.eu/) un projecte a nivell europeu que busca un lliurament d'energia eficient, fiable i sostenible per a la salut i el benestar de totes les

persones. Per ser més específics, les dades dels edificis corresponen a edificis situats al parc tecnològic *Walqa* a Osca, Aragó (https://www.ptwalqa.com/).

Disposem de més dades, extretes del campus de Romania *UVTgv* (https://elandh2020.eu/pilot-site/romania/) que disposa de dades de mes edificis. Aquestes dades no estan limitades fins al 2019, tenim registre des del 2019 fins al 2021.



```
hora,temperatura,humedad,Irradiancia,temp_amb,wind_speed,temperatura_1,temperatura_2,temperatura_3,valv_inv_1,valv_inv_2,
2019-01-01 00:00:00,,,,,,,,,,,,,,,,,-5139.830508474576,19210.0,19725.0,11260.0,,10800.0,10950.0,73500.0,,40.0,,,,,
2019-01-01 01:00:00,,,,,,,,,,,,,,,,,-5162.5,19100.0,22200.0,11560.0,,10890.0,11900.0,71625.0,,40.0,,,,,
2019-01-01 02:00:00,,,,,,,,,,,,,,,,,-5187.28813559322,19350.0,19950.0,11840.0,,10980.0,16700.0,70350.0,,40.0,,,,,
2019-01-01 03:00:00,,,,,,,,,,,,,,,,,-5160.833333333333,16090.0,21075.0,11440.0,,10860.0,55000.0,73125.0,,40.0,,,,,
2019-01-01 04:00:00,,,,,,,,,,,,,,,,,-5179.661016949152,19060.0,21075.0,12120.0,,14160.0,44450.0,71700.0,,40.0,,,,,
2019-01-01 05:00:00,,,,,,,,,,,,,,,,,-5182.142857142857,20060.0,18525.0,12100.0,,12450.0,66100.0,72375.0,,40.0,,,,,
2019-01-01 06:00:00,,,,,,,,,,,,,,,,,-5202.941176470588,13520.0,28575.0,12480.0,,12540.0,65150.0,73425.0,,40.0,,,,,
2019-01-01 07:00:00,,,,,,,,,,,,,,,,,,19200.0,32925.0,11680.0,,12150.0,72800.0,73575.0,,,,,,
2019-01-01 08:00:00,,,,,,,,,,,,,,,,,,19270.0,33000.0,12080.0,,12600.0,75600.0,31725.0,,,,,,
```

Figura 18: Dataset edificis UVTgv.

6. Requisits del sistema

El requisit principal que ha de complir el sistema és que donat un seguit de dades de consum energètic sigui capaç de predir el consum futur d'un edifici a base de ser entrenat amb les dades recollides anteriorment.

L'eina ha de proporcionar les funcionalitats necessàries com per poder predir el consum, rebre dades periòdicament, guardar la predicció obtinguda, tenir una mesura que permeti que es re-calculi de forma automàtica.

El programa ha de complir els següents requisits:

- Processa les dades d'entrada i les modifica per poder-les utilitzar durant totes les execucions.
- Definir els models, les seves característiques i paràmetres. Models ARIMA, SVR i xarxa neuronal.
- Entrenar el model entrant-li les dades d'entrada.
- Computar els diversos paràmetres de cada model per extreure prediccions precises, realitzant proves tant manuals com automàtiques.
- Ser capaços de testejar els valors i completar una eina autònoma capaç de predir qualsevol data.

El sistema ha de disposar d'un mètode comparatiu que analitzi les dades predites amb les dades reals de test i pugui determinar si s'ha de recalcular el model, per tal de fer l'eina automàtica.

Altres requisits a tenir en compte són certs requisits hardware del programa. Hem de disposar d'una màquina capaç de realitzar les prediccions en un temps acceptable (per veure els temps veure l'apartat 9).

Es disposa d'una màquina prou potent com per dur a terme el projecte (veure l'aparat 2).

7. Estudis i decisions

7.1. Llenguatge de programació

Per aquest projecte s'ha d'escollir un llenguatge que sigui versàtil y potent en càlculs estadístics, gestió de dades (ja que no utilitzarem cap tipus de bases de dades) i eficient en qualsevol còmput.

Podríem realitzar l'aplicació en diversos llenguatges que compleixen aquestes característiques. Per comoditat només tindré en compte llenguatges orientats a objectes, ja que la programació funcional, programació lògica i derivats només complicarien el procés perquè no hi estic tan familiaritzat. A més, hem de tenir en compte les llibreries de les que disposem, que ens simplificaran molt el procés.

Si busquem els llenguatges més comuns per intel·ligència artificial apareixen llenguatges com:

- **Python.** *Open Source*, un llenguatge dinàmic i ple de llibreries orientades a la IA.
- **R.** Orientat a la estadística, que destaca per l'elaboració d'anàlisis i generació de gràfics.
- **Lisp.** Un dels llenguatges més antics pel que fa a IA.
- **Prolog.** Programació lògica i capaç de gestionar sistemes molt complexes (descartat).
- **Java.** Una àmplia varietat de projectes Machine Learning s'implementen en Java.

El factor principal pel qual he escollit **Python** és per l'àmplia varietat de llibreries que ofereix orientades a la intel·ligència artificial, com *Sklearn* o *Tensorflow*. A més, ja estic familiaritzat amb aquest entorn i la gestió de dades que ofereix *Pandas* degut a que durant la carrera l'hem utilitzat per implementar sistemes IA.

7.2. Llibreries

El projecte consta de les següents llibreries:

- **Math**. S'utilitza per realitzar operacions matemàtiques.
- **Calendar**. S'utilitza per tractar els dies de l'any. En el meu cas l'utilitzo per saber un nombre entre 0 i 6 que indica si és un dilluns, dimarts, dimecres...
- **Datetime**. Llibreria per emmagatzemar les dates en format de data.
- **Matplotlib**. S'utilitza per representar les dades en format gràfic.
- **Numpy**. S'utilitza per tractar dades com vectors. Moltes llibreries utilitzen numpy per crear llistes de dades i per utilitzar operacions matemàtiques que proporciona.
- **Scipy**. Conté mètodes relacionats amb optimització, àlgebra lineal, integració i interpolació entre altres.
- **Pandas**. És una llibreria que s'utilitza per carregar dades des de excels o csv i convertir-los en la seva estructura de dades, el *Dataframe*. Un Dataframe és una taula de dades fàcilment manipulable. Pot assignar valors com strings i dates a les columnes i files de la taula, a diferència de les matrius convencionals.
- **Statsmodels**. És un mòdul Python que conté molts tipus de models estadístics, tests d'hipòtesis i tipus d'exploració de dades. Ens interessa perquè conté la millor aproximació al model SARIMAX.
- **Sklearn**. És la llibreria és extensa pel que fa a models predictors. Ens permetrà entrenar els models regressius SVR.
- **Tensorflow**. És una API Python molt completa en quant a Deep Learning.
- **Pmdarima**. Llibreria que conté algorismes per provar automàticament diversos paràmetres del model SARIMAX.

Les decisions i estudis presos relacionats amb els algorismes utilitzats en el projecte es troben explicats amb les proves realitzades a l'apartat 9.

8. Anàlisi i disseny del sistema

8.1. Anàlisi de dades

El primer anàlisi de les dades que haurem de realitzar són les columnes dels *datasets* que necessitarem.

Tenim dos fitxers de dades a utilitzar: un anomenat *horari_p.csv* i un altre anomenat *dades_wqlqa_hora.csv*. Els *datasets* contenen més informació que la que necessitem, ja que només necessitem la columna que conté l'hora que s'ha realitzat la mesura i les columnes que contenen el consum energètic de cada edifici. Del primer fitxer ens fixarem en les dades del 2018 i del segon fitxer del 2019 i 2020.

La següent qualitat a tenir en compte del *dataset* és la quantitat de valors finits que conté el *dataset*. Entenem com a "nombre finit" un nombre que no és nul ni tendeix a infinit (en termes Python els valors nuls són *NaNs* i les valors molt grans es representen com a *inf*).

Veiem el tant per cent de les dades finites del 2018:

```
Edifici 1: % finits
True     93.504566
False     6.495434
```

```
Edifici 2: % finits
True     93.47032
False     6.52968
```

Figura 19: Tant per cent de nombres finits.

I les dades de tots els edificis del 2020:

- Edifici 'Observatory Building': tant per cent finits
  - False 54.166667
  - True 45.833333
- Edifici 'CTU Building': tant per cent finits
  - False 51.084475
  - True 48.915525
- Edifici 'FHA Building': tant per cent finits

    - False 56.986301
    - True 43.013699
- Edifici ‘Edificio AST’: tant per cent finits
    - False 100.0
- Edifici ‘Felix de Azara Building’: tant per cent finits
    - False 52.009132
    - True 47.990868
- Edifici ‘Miguel Servet Building’: tant per cent finits
    - False 51.780822
    - True 48.219178
- Edifici ‘Ramón y Cajal Building’: tant per cent finits
    - False 51.152968
    - True 48.847032

Les dades del 2019 de tots els edificis també són similars a les del 2020.

Com podem veure, les dades més significatives es troben en el primer fitxer, les dades del 2018. Són molt completes i només contenen dades nul·les en certs mesos.

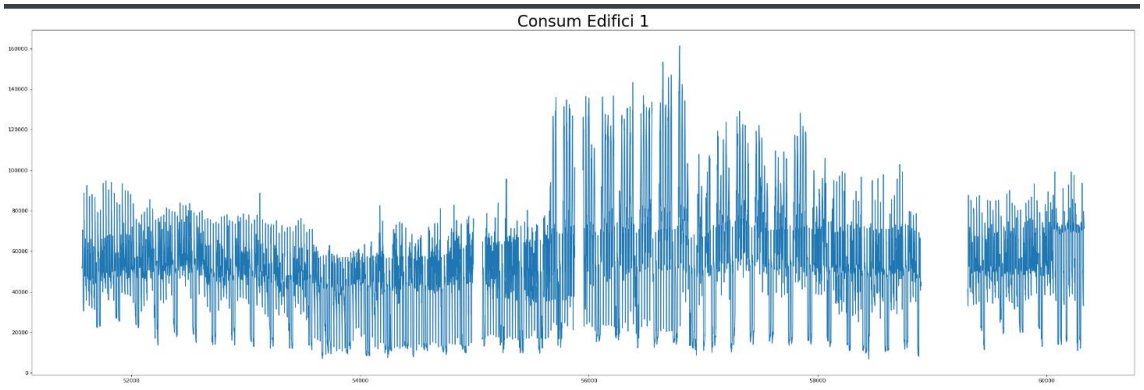


Figura 20: Dades del consum de l’edifici 1 del 2018.

En la figura 18 he representat el consum de tot l’any 2018 de l’edifici 1. Estan ordenades de gener (esquerra del tot) a desembre (a la dreta). No disposem de les mesures del consum del mes de novembre. La situació és similar en l’edifici 2.

En canvi, en el 2020 podem veure la següent representació:

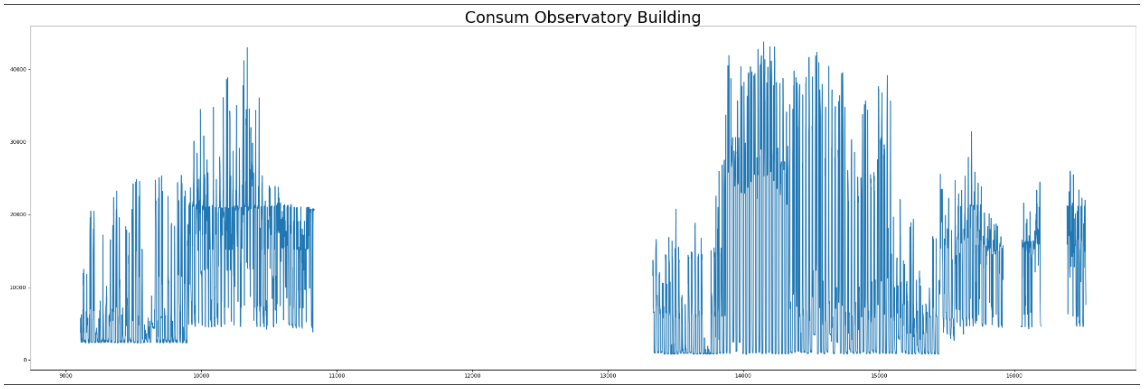


Figura 21: Dades del consum de l'edifici Observatory 2020.

Tots els edificis del 2020 tenen la mateixa estructura. Aquest any és molt interessant perquè coincideix amb l'aparició del SARS-COV-2 o més comunament conegut com a Covid-19. No es tenen registres des del mes de març fins al juliol degut a la quarantena a nivell global que vam patir.

8.2. Disseny del sistema

Els procediments a seguir del programa seran els següents:

- Entrar les dades.
- Processar les dades d'entrada.
- Crear un primer model.
- Entrenar el model.
- Crear els models ARIMA, SVM i RNN.
- Aplicar els algorismes predictors i guardar la predicció.
- En bucle farem:
    - Treure la predicció com a sortida del sistema.
    - Mostrar els resultats.
    - Controlar l'error de les noves dades d'entrada.
    - Quan l'error superi un cert llindar, tornar a re-calcular el model.
    - Crear els models ARIMA, SVM i RNN.
    - Aplicar els algorismes predictors.
- Fi del bucle.

Cal tenir en compte que el processament de dades d'entrada s'ha dut a terme en l'apartat 8.1., ja que si no disposem d'una certa qualitat de les dades d'entrada s'haurà de prendre la decisió d'obviar la predicció per la quantitat d'error que obtindrem.

La idea general del sistema és la representada en la Figura 20, rebem les dades del consum energètic en format de sèrie temporal, la tractem i decidim si les dades són vàlides. Creem el primer model i calculem una primera predicció, que la rep l'auditor. L'auditor va rebent dades fins que la predicció supera un llindar (un error màxim "suportable").

Un cop supera el llindar o ens quedem sense dades predites que hem guardat en el model, es re-calcula el model, aplicant tots els algorismes esmentats: LSTM, SARIMAX i SVR.

Es treu el *forecasting* o predicció sempre que es demani a l'auditor.

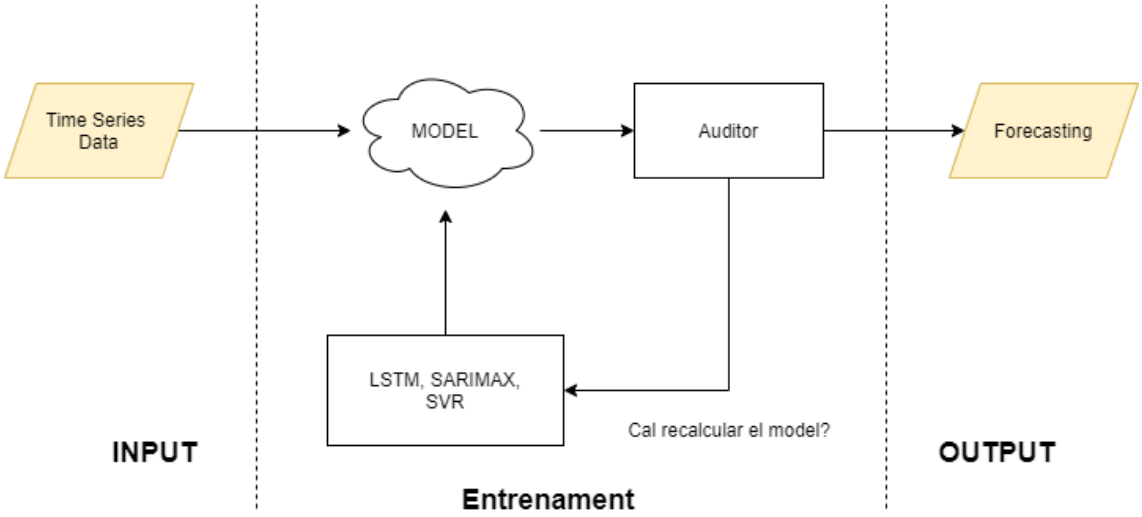


Figura 22: Blocs de l'aplicació.

Per veure més detalladament el procés a seguir, a continuació podem veure un diagrama del procés del sistema.

Procés del sistema

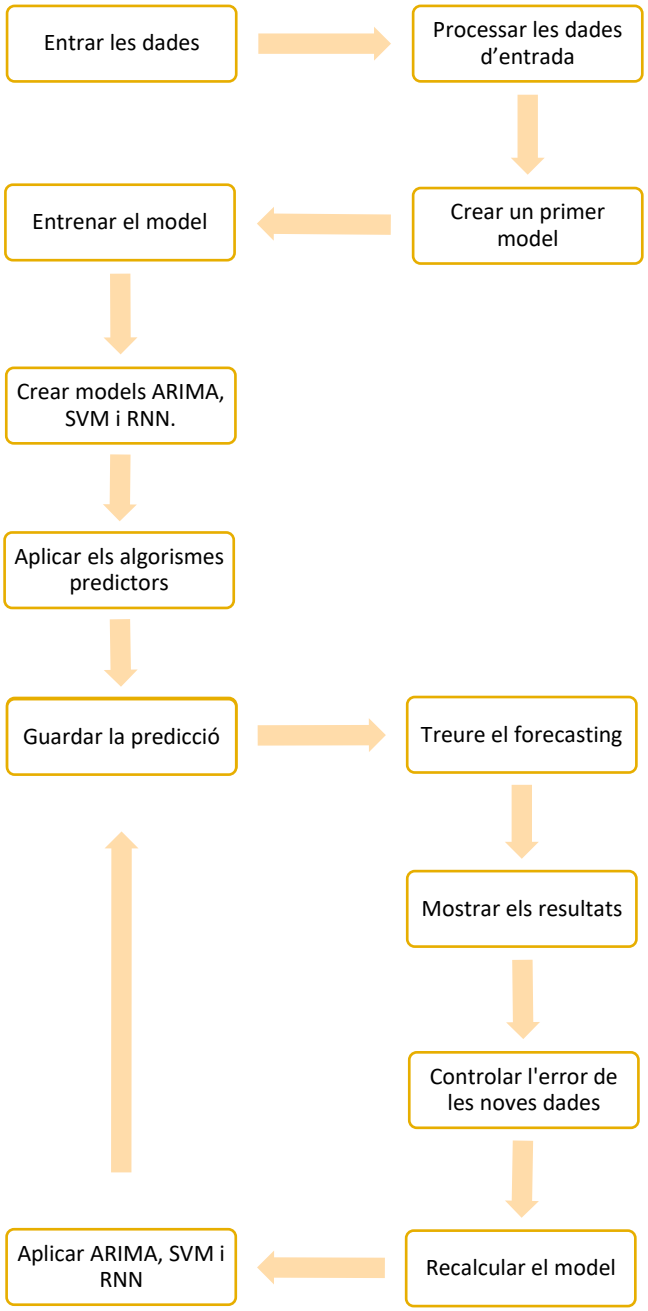


Figura 23: Diagrama del procés del programa.

9. Implementació i proves

Veurem la implementació per blocs. El primer bloc serà el tractament i processament de les dades. El segon serà la implementació dels models predictors regressius i l'últim bloc serà de representació i comprovació dels resultats.

9.1. Tractament de les dades

El primer pas en la implementació d'aquest sistema és llegir les dades. Carregarem les dades des d'un fitxer .csv amb el consum d'energia a un *dataframe* de la llibreria *pandes.*

Un cop carregat el *dataframe* ens quedarem amb les columnes necessàries per fer el càlcul, eliminant les restants. Ens quedarem amb l'hora (columna *index*) i les columnes respectives als edificis (columnes *p_Edificio_1* i *p_Edificio_2*).

	index	p_Edificio_1	p_Edificio_2
0	2012-02-13 07:00:00	83923.333333	163788.033333
1	2012-02-13 08:00:00	81622.083333	187333.033333
2	2012-02-13 09:00:00	92052.500000	199666.150000
3	2012-02-13 10:00:00	90572.386667	209308.300000
4	2012-02-13 11:00:00	92103.750000	214241.650000
...	...	...	...
70218	2019-09-17 04:00:00	65700.000000	11300.000000
70219	2019-09-17 05:00:00	69700.000000	11166.666667
70220	2019-09-17 06:00:00	88750.000000	11300.000000
70221	2019-09-17 07:00:00	62750.000000	21266.666667
70222	2019-09-17 08:00:00	53500.000000	26933.333333

Figura 24: Dataframe amb les dades carregades.

Al tenir moltes dades, reduirem també el nombre de línies per no fer execucions excessivament llargues filtrant per any. Les dades de test seran del 2018 i les de test del 2019.

Per evitar problemes, procedirem a sanejar les dades, detectant els valors *NaNs* dels edificis. Contemplem diverses solucions. Una idea podria ser eliminar les línies que continguin valors *NaN* i controlar aquests valors abans de realitzar l'entrenament dels models perquè no es desquadrin les dades, ja que ens interessa tenir com a mínim 24 mostres per dia per controlar el nostre

model dia a dia. Una altra solució és afegir a cada valor *NaN* la mitjana de les dades, però s'ha de tenir en compte que si les dades d'entrada són poc sanes pot comportar en *underfitting* o en prediccions pobres.

Alguns algorismes no permeten entrenar amb valors *NaN* en les dades d'entrenament.

Com hem comentat a l'apartat 8.1., les dades referents al 2018 dels dos edificis superen el 93% de nombres finits, per tant, si canviem els valors *NaN* per les mitjanes de cada dia no s'haurien de produir problemes.

Implementació

- **Importació de llibreries.**
  Utilitzarem les llibreries explicades en l'apartat 7.2. No necessitarem totes les funcionalitats de cada llibreria, només les marcades per cada import

```
import numpy as np
from scipy.stats import ttest_ind
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
import itertools
from datetime import datetime
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import math as math
from math import sqrt
```

Figura 25: Paquets importats de les llibreries.

- **Importar les dades.**
  Importarem les dades dels dos fitxers que disposem: *horari_p.csv* i *dadees_walqa_hora.csv* (Figura 26). Aprofitem el procés de importar per indicar quines son les columnes que ens interessen en el nostre model de cada fitxer. Per comprovar que les dades s'han carregat correctament es poden mostrar per pantalla (Figura 27).

```
#Llegim les dades
ds = pd.read_csv(r'.\horari_p.csv', sep=';')
df = pd.DataFrame(ds, columns=['index', 'p_Edificio_1', 'p_Edificio_2', 'p_Edificio_3'])

df_full = pd.read_csv(r'.\dades_walga_hora.csv', sep=',')
df_full = pd.DataFrame(df_full, columns=['hora','Observatory Building','CTU Building',
                                         'FHA Building','Edificio AST','Felix de Azara Building',
                                         'Miguel Servet Building','Ramón y Cajal Building'])
```

Figura 26: Codi de lectura del dataset.

	hora	Observatory Building	CTU Building	FHA Building	\
0	2019-01-01 00:00:00	19210.0	19725.0	11260.0	
1	2019-01-01 01:00:00	19100.0	22200.0	11560.0	
2	2019-01-01 02:00:00	19350.0	19950.0	11840.0	
3	2019-01-01 03:00:00	16090.0	21075.0	11440.0	
4	2019-01-01 04:00:00	19060.0	21075.0	12120.0	

	Edificio AST	Felix de Azara Building	Miguel Servet Building	\
0	NaN	10800.0	10950.0	
1	NaN	10890.0	11900.0	
2	NaN	10980.0	16700.0	
3	NaN	10860.0	55000.0	
4	NaN	14160.0	44450.0	

Figura 27: Representació de les primeres files del primer fitxer.

	index	p_Edificio_1	p_Edificio_2	p_Edificio_3
0	2012-02-13 07:00:00	83923.333333	163788.033333	NaN
1	2012-02-13 08:00:00	81622.083333	187333.033333	NaN
2	2012-02-13 09:00:00	92052.500000	199666.150000	NaN
3	2012-02-13 10:00:00	90572.386667	209308.300000	NaN
4	2012-02-13 11:00:00	92103.750000	214241.650000	NaN

Figura 28: Representació de les primeres files del segon fitxer.

- **Eliminar línies sobrants i dividir les dades de *training* i test.**

  El següent pas és eliminar les dades que no siguin del 2018. Cal explicar que es poden modificar les dates per ajustar el model a les dades d'entrada. Al mateix temps que eliminem les files (Figura 27) estem dividint les dades d'entrada en dades d'entrenament i dades de testeig.

```
#Dividim les dades per entrenar i per testejar
df_training = df.loc[df['index'].str[:7] == '2018']
df_test = df.loc[df['index'].str[:7] == '2019']
```

Figura 29: Codi de filtratge de files.

- **Anàlisi de dades sobre *NaNs*.**

  Utilitzant la funció *is_finite()* de la llibreria *numpy* podem saber si els valors no són ni *NaN* ni *inf*. Retorna una llista de *booleans* i la funció

value_counts() ens conta tots els valors d'aquesta llista que son *True* o *False*.

```
'''
Anàlisi de dades
'''
print('Edifici 1: % finits')
print(np.isfinite(df_training['p_Edificio_1']).value_counts(normalize=True).mul(100))
print('Edifici 2: % finits')
print(np.isfinite(df_training['p_Edificio_2']).value_counts(normalize=True).mul(100))

for i in df_full.columns:
    if i != 'hora':
        print('Edifici %s: tant per cent finits' % i)
        print(np.isfinite(df_full[i]).value_counts(normalize=True).mul(100))
```

Figura 30: Codi tant per cent dels finits.

```
Edifici 1: % finits
True     93.504566
False     6.495434
Name: p_Edificio_1, dtype: float64
Edifici 2: % finits
True     93.47032
False     6.52968
Name: p_Edificio_2, dtype: float64
Edifici Observatory Building: tant per cent finits
True     52.31102
False    47.68898
Name: Observatory Building, dtype: float64
Edifici CTU Building: tant per cent finits
True     51.673497
False    48.326503
Name: CTU Building, dtype: float64
```

Figura 31: Resultat de l'anàlisis.

- **Emplenar NaNs.**

  La funció *fillna()* de la llibreria *pandes* permet fer la funció d'emplenar tots els valors NaNs d'un *dataset* amb uns valors *default.*

```
def fillnan(self, X):
    return X.fillna(X.mean(), inplace=True)
```

Figura 32: Fill NaNs

- **Mostratge de dades.**

  Per representar les dades utilitzaré la llibreria *matplotlib.* Permet fer gràfics on se li pot indicar el títol, la mida del gràfic, el color i el tipus (línies, punts..) entre d'altres. Per exemplificar com serà un gràfic simple veure la Figura 33.

```
#Mostro les dades per veure el consum total
plt.figure(num=None, figsize=(30, 10), dpi=80, facecolor='w', edgecolor='k')
plt.title('Consum Observatory Building', fontsize=30)

plt.plot(df_training['Observatory Building'])
plt.show()
```

Figura 33: Codi per la representació d'un gràfic simple

9.2. Implementació del sistema predictor

Un cop tenim les dades entrades i tractades correctament, procedim a la part més important del sistema: la implementació de la creació, definició i entrenament del model.

Abans de veure els diferents algorismes aplicats, anem a comentar alguns aspectes importants com l'estructura de les dades i funcionalitats recurrents.

Estructura dins del model.

En el model ens guardarem les següents variables:

- ***dfPerfils***. Conté les dades entrades inicialment al model i les que s'afegeixin posteriorment. És un *dataframe* que conté 7 **dataframes,** un per a cada dia, essent el *dataframe* a la posició 0 el que conté les dades del dilluns i el 6, el diumenge. El resultat és una matriu de *dataframes*. S'afegeixen les noves dades entrades al model al dia que correspongui. Per moltes dades entrades dins *dfPerfils* sempre s'utilitzen les dades dels 3 últims mesos.

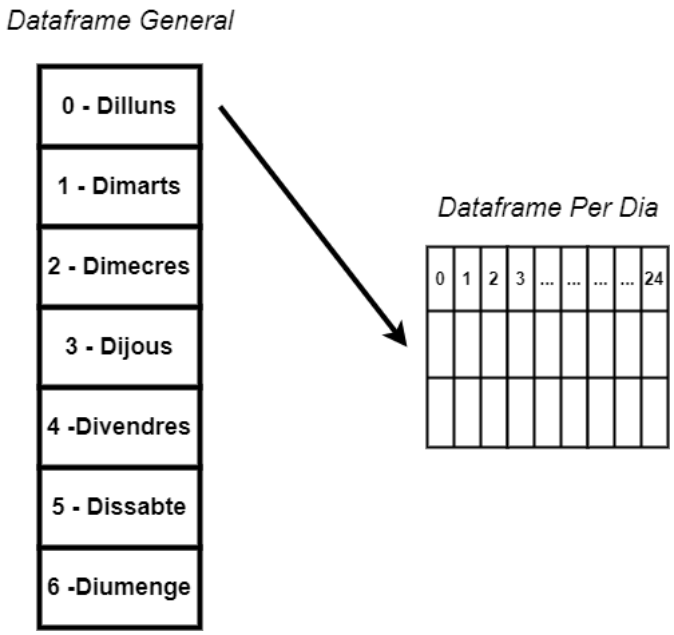


Figura 34: Estructura dfPerfils.

- **dfMeans**. És una matriu *d'np.arrays* que conté les mitjanes de cada hora de cada dia. Cada vegada que entra una nova dada es calculen les noves mitjanes de cada hora i es guarden en la posició que pertoqui. Repetim, 0 indica dilluns i 6, diumenge.

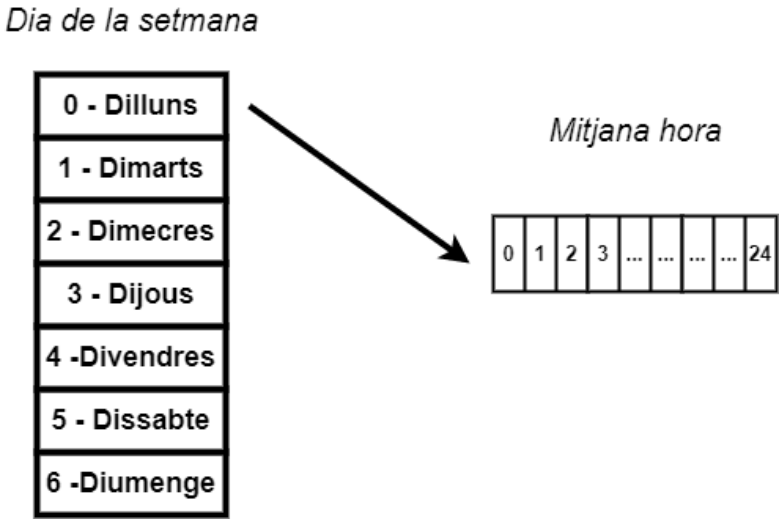


Figura 35: estructura dfMeans.

- **Predicció**. És la variable on es guarda la predicció resultant del procediment d'executar tots els algorismes i extreure'n el model amb menor MAPE (veure a continuació).

Càlcul d'errors.

En tot el projecte, sempre que haguem de calcular algun tipus d'error s'utilitzarà la mesura MAPE o Mitjana de l'Error Absolut en Percentatge (en anglès *Mean Absolute Percent Error*). És una mesura molt comuna en mètodes de *forecasting,* que retorna el tant per cent d'error entre dos *datasets.* L'utilitzarem per calcular l'error entre la predicció del nostre model i els valors reals entrats al sistema.

```
# Calcula MAPE entre dos datasets
def mape(self, actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.nanmean(np.abs((actual - pred) / actual)) * 100
```

Figura 36: Codi MAPE.

Inserció de nous dies al model.

Per inserir noves dades al model simplement afegim les dades dins *dfPerfils*. La funció *train* és l'encarregada de dur a terme aquesta feina, la qual rep el dataset de les dades i a quin dia corresponen (Figura 37).

```
# Train
def train(self, dfDia, dia):
    if 0 <= dia <= 6:
        self.dfPerfils[dia] = self.appendRow(self.dfPerfils[dia], dfDia.values)
    return None

# Afegeix i retorna el dataframe entrat amb la fila entrada
def appendRow(self, df, row):
    if len(df.columns) != len(row):
        return df
    else:
        dictOfValues = {i: row[i] for i in range(0, len(row))}
        return df.append(dictOfValues, ignore_index=True)
```

Figura 37: Codi inserció de dades al model

9.2.1 Model de mitjanes

El primer model que veurem serà el més simple, el model que serveix com a *back up* per si les demés prediccions fallen.

Simplement dins de la estructura *dfMeans* ens guardem el càlcul de la mitjana de les hores de cada dia, i en el moment d'utilitzar els algorismes aquests seran els escollits (Figura 38).

```
# Actualitza totes les mitjanes
def updateMeans(self):
    for i in range(0, 24):
        for j in range(0, 7):
            self.dfMeans[j][i] = np.nanmean(self.dfPerfils[j][i])
```

Figura 38: Actualtizador de les mitjanes.

El procés d'actualització es durà a terme cada vegada que es cridi a la funcionalitat principal del sistema, que una de les crides és la següent funció. Retorna les mitjanes i *l'accuracy* (el grau d'encert de la predicció, es calcula fent 100 – error MAPE).

```
def fitMitjanes(self, df_training, dia, df_test):
    # Actualitzem les mitjanes per poder fer el càlcul actualitzat
    self.updateMeans()

    print('ACURACY:  ')
    print(100 - self.mape(df_test[self.dadesColumna][:24], self.dfMeans[dia]))

    return 100 - self.mape(df_test[self.dadesColumna][:24], self.dfMeans[dia]), self.dfMeans[dia]
```

Figura 39: Codi fitMitjanes.

9.2.2 Entrenament SVR

Utilitzarem l'algorisme SVR implementat en la llibreria *Sklearn*. El model té diversos paràmetres a regular tot i que molts dels que proporciona no els modificarem, deixant-los amb el valor per defecte.

- **Kernel**. (array (n,m)) Indica la funció que aplicarà el nostre algorisme SVR. Disposa de 5 tipus: 'linear', 'poly', 'rbf', 'sigmoid' i 'precomputed'.
- **C**. (array (n,)) És el paràmetre de regularització. Serveix per indicar al nostre model si ha de ser molt estricte o no amb les nostres dades, modificant el marge i penalitzant els valors a una distància major a C (veure apartat 5.2.2).

- **Gamma**. (float) És el coeficient per a la funció *kernel*. Disposa de dos tipus: 'scale' i 'auto'.
- **Epsilon**. (float) Indica el grau de tolerància. Quant més gran, més errors admets que tingui la solució.

És molt interessant veure quina combinació de paràmetres és la més òptima per treure'n els millors resultats. Ho veurem posteriorment en l'apartat de proves.

Un exemple de creació del model seria el de la Figura 37.

```
svr = SVR(kernel='rbf', C=100, gamma='auto', epsilon=.0001)
```

Figura 40: Inicialització del model SVR.

SVR d'*skelarn* no accepta valors NaN, per tant, haurem de tenir en compte que no n'hi hagi cap. Anteriorment en aquest projecte hem parlat sobre el tractament d'aquests valors buits, i ho hem fet precisament per aquest cas (a apart de que és una bona pràctica eliminar els valors buits). Eliminem els valors buits amb la comanda *fillna*().

Una bona pràctica comuna dins el *Machine Learning* és la d'estandarditzar les dades. Permet obtenir les dades escalades i evitem que hi hagin problemes amb dades molt distants. Aplicant la fórmula de la figura 38 obtenim el valor "z", el valor d'estandarditzar el valor.

$$z = \frac{value - mean}{standard\ deviation}$$

Figura 41: Metode d'estandaridtzació Z-score.

Sklearn ja té implementada aquesta transformació dins la carpeta *preprocessing*, anomenada *StandardScaler*. Aquesta classe permet aplicar la transformació amb el mètode *fit_transform*() que estandarditza els valors i la funció *inverse_transform*() que transforma les dades al seu estat normal.

Només cal tenir en compte que has de tenir un transformador per cada set de dades transformades, si no els valors poden variar dels que tocarien.

FIT AND PREDICT

Cal dividir les nostres dades de *training* en dos blocs. Un primer bloc (X) que són els vectors d'entrenament i un altre bloc (y) que son els valors "objectiu" o valors predictors. Els vectors són la relació entre dos valors, en el nostre cas relacionarem la data en què es va mesurar el consum i el consum energètic en sí. El nostre objectiu és predir el consum energètic, per tant els nostres valors predictors (y) seran el consum energètic.

La llargada de X i y ha de ser la mateixa. Suposant una mida de X (n_files,2) Y haurà de ser (n_files,)

Aquesta partició d'X i y la utilitzarem per ajustar i entrenar el nostre model. Amb la comanda *fit*() dins el model SVR ajustarem el model a les dades (Figura 42).

```
svr.fit(X, y)
predict = svr.predict(X_test_t)
```

Figura 42: Fit i predict SVR.

Amb la comanda *predict*() podem extreure la predicció del model. Com a paràmetre ha de rebre unes dades de test en format similar a la X mencionada anteriorment. No han de ser de la mateixa mida la X de training i la X de test.

9.2.3 Proves SVR

Les proves es centren en trobar la millor predicció possible a l'hora de comparar els tres models.

Les primeres proves que realitzarem seran sobre els **paràmetres** del model.

KERNEL

Fem proves entrant totes les dades del 2018 al sistema. La resta de paràmetres de l'SVR es quedaran amb valors *default*.

- **RBF.**

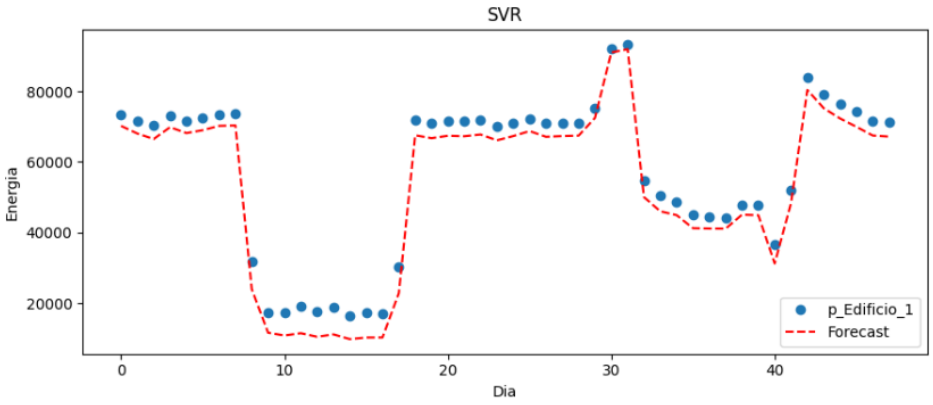


Figura 43: SVR RBF.

ACURACY: 87.87%

- **LINEAR.**

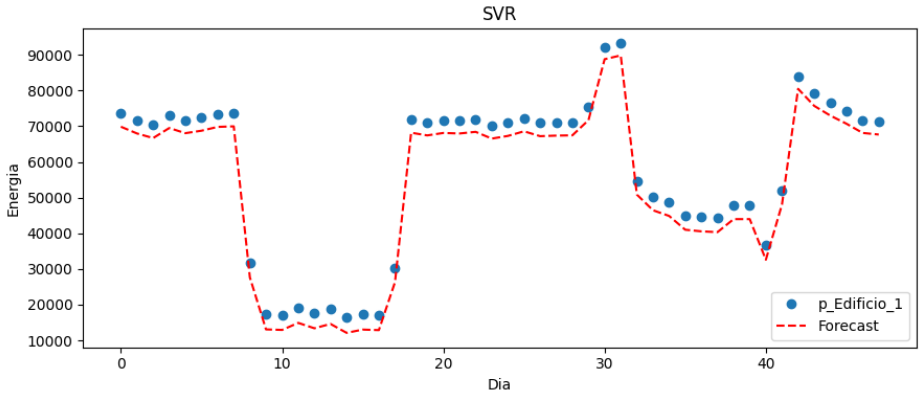


Figura 44: SVR lineal.

ACURACY: 90.85%

- **POLY.**

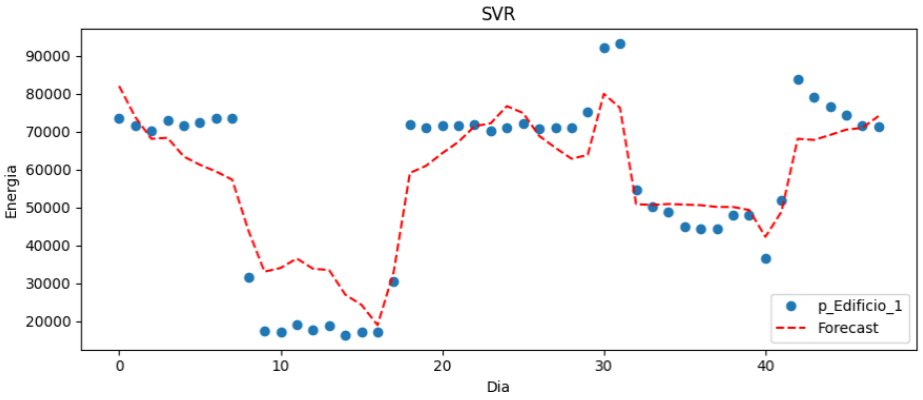


Figura 45: SVR polinomial.

ACURACY: 79.72%

No obté bons resultats comparant amb les dues funcions anteriors.

- **SIGMOID.**

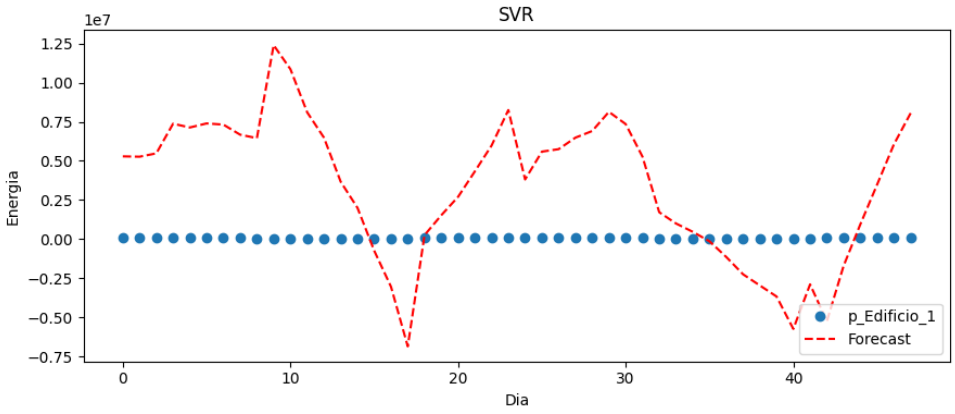


Figura 46: SVR sigmoidal.

ACURACY: -11574.26%

No obté bons resultats, fa que les prediccions es disparin en quant a escala. El primer valor de la predicció és 73500.0 i el predit per la sigmoidal és 5289595.08. Descartem aquesta opció.

Definitivament els dos primers *kernels*, *rfb* i *linear* obtenen molt bons resultats. Per acabar de comprovar quin dels dos és millor anem a provar amb dades de tots els mesos del 2018, predient les dades dels següents dos dies.

I obtenim els següents errors:

Mes \ MAPE	**RBF**	**Linear**
Gener	11.44541	12.64506
Febrer	8.68909	9.41286
Març	27.49472	28.20332
Abril	6.68636	10.64552
Maig	15.06592	18.59227
Juny	12.27010	14.18232
Juliol	21.13698	23.89781
Agost	68.21848	89.46997
Setembre	16.52590	14.53954
Octubre	28.15726	33.52911
Novembre	24.39796	24.66492
Desembre	7.50055	9.95785

Figura 47: Comparació errors RBF i Linear

Per determinar el que obté millors resultats, si fem l'error relatiu entre els dos *kernels* obtenim que l'*rbf* és millor en un 14.5481%.

C

Si mirem les combinacions del paràmetre C amb les dades del desembre obtenim:

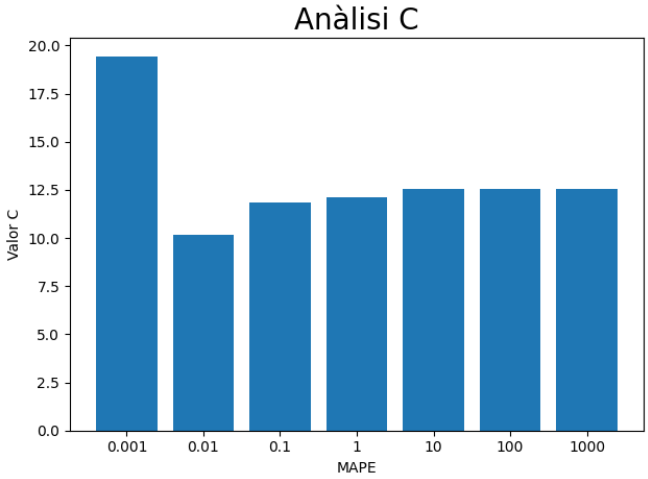


Figura 48: Anàlisi paràmetre C.

El millor valor per C és 0.01. Anem a fer les mateixes proves fetes pel *kernel*, a veure si millora en algun cas. Per comprovar que realment aquest canvi seria beneficiós en tots els casos, anem a provar amb un altre mes.

Si representem les dades, per exemple del maig, ens trobem que no intenta predir els valors molt distants de la mitjana dels valors.

Amb C=1:

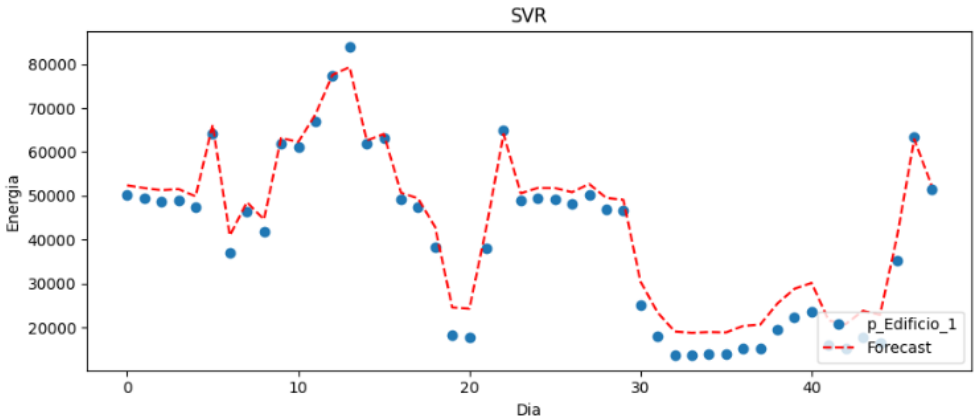


Figura 49: Forecast dell maig amb C=1.

Amb C=0.01:

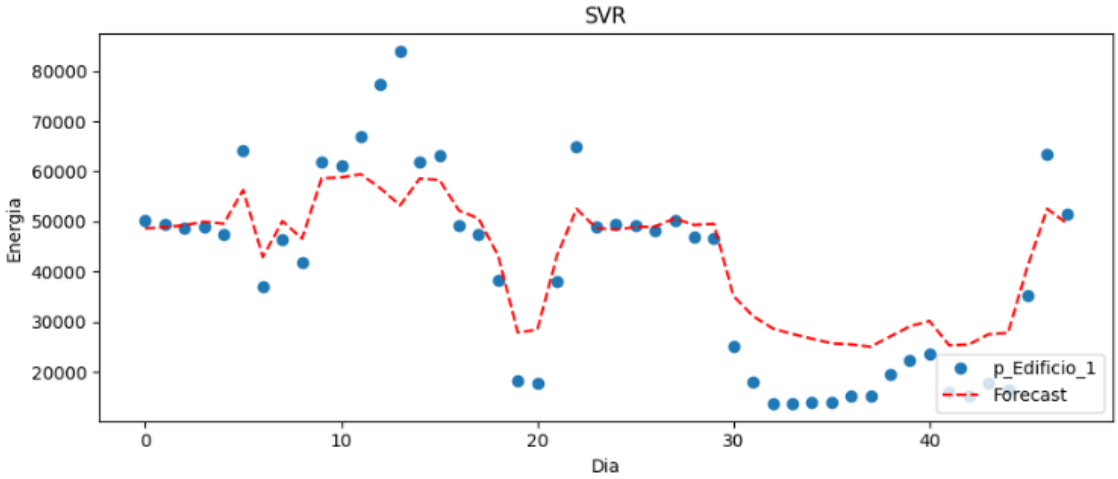


Figura 50: Forecast C = 0.01 del maig.

L'error resultant del canvi és incrementa l'error en més del doble. Per tant, hauríem de quedar-nos amb un valor C que no corrompi la resta de dades. Deixarem assignat el valor C amb el seu valor per defecte, 1.

Gamma

El tipus de gamma no aplica cap canvi en els resultats.

9.2.4 Entrenament SARIMAX

A diferència del model SVR no necessitem marcar quines son les variables a predir en format X i y, simplement s'empassa totes les dades del consum energètic directament sense indicar-li índexs.

El model SARIMAX de la llibreria *statsmodels* té els següents paràmetres principals:

- **Endog**. (array n,m) El conjunt de dades a processar.
- **Exog**. (array n,m) Conjunt de variables externes que poden afectar a les dades.
- **Order**. (iterable) Són una llista de paràmetres (p,d,q) que corresponen als paràmetres p, d i q del model ARIMA convencional (veure apartat 5.3.1).

- **Seasonal_order**. (iterable) Corresponen als paràmetres P, D, Q i s del model SARIMA (verue apartat 5.3.1)

La resta de paràmetres que proporciona no ens interessaran molt.

Un exemple de creació del model seria el següent:

```
sarimax = sm.tsa.statespace.SARIMAX(endog=df,
                                    trend='n',
                                    order=(1, 0, 0),
                                    seasonal_order=(1, 1, 1, 24))
```

Figura 51: Inicialització del model SARIMAX.

FIT

A diferència de l'SVR no necessitem indicar quines són les dades d'entrada el model a l'hora d'ajustar-lo ja que ja ha estat indicat a l'hora de crear el model inicial.

Llavors, com canviem les dades d'entrada al *fit*? Hem de generar cada cop un model? La resposta és no, conté una propietat anomenada *extend*() que permet reajustar les dades de *l'endog* entrades al model inicial. De cares al nostre projecte, no ens interessa aquesta funció *extend*() ja que cada vegada que volem re-calcular els models els tornem a crear, ja que no hi ha cap necessitat de guardar-los en memòria degut a que ens guardem les prediccions obtingudes. Si s'hagués de re-calcular el model molt sovint s'hauria de replantejar si és millor pràctica guardar els models SVR, LSTM i SARIMAX en memòria i crear-lo cada cop.

PREDICT

SARIMAX treu els resultats del model en una classe anomenada *SARIMAXResults*, que conté la funció *forecast*().

```
predict = res.forecast(steps=168, exog=test[:168])
```

Figura 52: Exemple forecast de SARIMAX d'una semana.

Forecast necessita que indiquem quants passos ha de predir. Cada pas és un valor predit. El nostre model de dades reparteix les dades per hores de cada dia, per tant, si volem obtenir la predicció d'un dia haurem d'indicar "setps=24".

També se li pot indicar valors *exog* per millorar la predicció, afegint-los com paràmetres externs.

9.2.4. Proves SARIMAX

Una de les parts més importants del SARIMAX és l'elecció dels paràmetres amb els que treballarem. Com és una tasca complexa, existeixen diversos algorismes que t'ajuden a trobar els millors paràmetres.

La llibreria *pmdarima* conté una funcionalitat anomenada *auto_arima()* que permet crear diversos models amb totes les comparacions que li indiquis i et troba automàticament els paràmetres òptims per al teu sistema.

Conté molts paràmetres per indicar l'anàlisi, recomano veure la documentació.

```
from pmdarima.arima import auto_arima
stepwise_model = auto_arima(df,start_p=0,d=1,start_q=0,
                            max_p=5,max_d=5,max_q=5, start_P=0,
                            D=1, start_Q=0, max_P=5,max_D=5,
                            max_Q=5, m=24, seasonal=True,
                            error_action='warn',trace=True,
                            supress_warnings=True,stepwise=True,
                            random_state=20,n_fits=50)

print(stepwise_model.aic())
```

Figura 53: Inicialització auto arima.

Obtenim un grau de viabilitat que incorpora el model ARIMA (AIC) que generalment busquem que aquest valor sigui mínim. No ens hem de preocupar molt perquè l'*auto arima* l'escull per nosaltres.

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[24]             : AIC=1907.479, Time=0.02 sec
 ARIMA(1,1,0)(1,1,0)[24]             : AIC=1909.040, Time=0.22 sec
 ARIMA(0,1,1)(0,1,1)[24]             : AIC=1907.886, Time=0.23 sec
 ARIMA(0,1,0)(1,1,0)[24]             : AIC=1909.962, Time=0.16 sec
 ARIMA(0,1,0)(0,1,1)[24]             : AIC=1943.777, Time=0.15 sec
 ARIMA(0,1,0)(1,1,1)[24]             : AIC=1916.071, Time=0.31 sec
 ARIMA(1,1,0)(0,1,0)[24]             : AIC=1908.355, Time=0.03 sec
 ARIMA(0,1,1)(0,1,0)[24]             : AIC=1908.470, Time=0.04 sec
 ARIMA(1,1,1)(0,1,0)[24]             : AIC=1909.794, Time=0.10 sec
 ARIMA(0,1,0)(0,1,0)[24] intercept   : AIC=1909.158, Time=0.02 sec

Best model:  ARIMA(0,1,0)(0,1,0)[24]
Total fit time: 1.269 seconds
1907.4785570843524
```

Figura 54: Resultat consola auto arima.

Si provem a realitzar la predicció de tots els dilluns del 2018 envers el primer dilluns del 2019 utilitzant el model indicat per *l'auto arima* ens trobem una sorpresa. La predicció és molt dolenta (Figura 52), on l'error és superior al 300%.

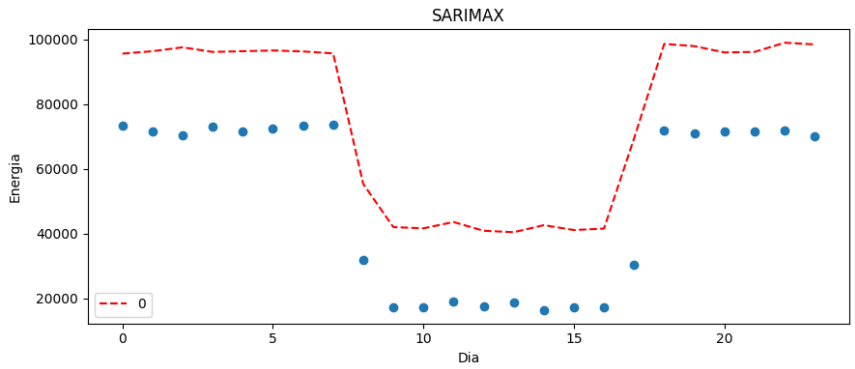


Figura 55: Predicció de l'auto arima.

I si provem de predir 7 dies més, podem veure que no obté gaires bons resultats, tenint una tendència positiva al llarg del temps (Figura 56).

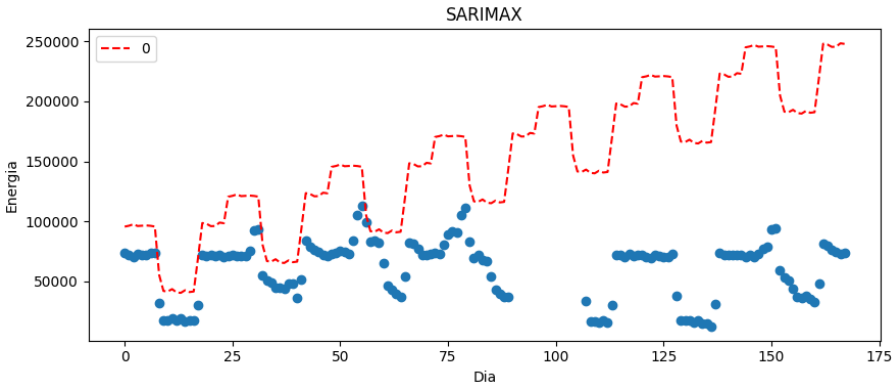


Figura 56: Predicció auto arima a larg termini.

Al veure que l'algorisme principal per trobar els millors paràmetres no ens treu bons resultats, haurem de realitzar diverses modificacions per intentar millorar el model. Per començar provarem d'estandarditzar les dades, però avanço que no produeix cap canvi, els resultats són els mateixos.

He implementat un algorisme que fa totes les combinacions dels valors p, d i q i manualment executa cada SARIMAX i obté un altre indicador de viabilitat (BIC), que està relacionat amb l'AIC, però per descartar que l'error ve donat per utilitzar l'AIC.

El codi és el següent:

```
for combinacio in pdq:
    for combinacioS in pdqs:
        mod = sm.tsa.statespace.SARIMAX(df,
                                        order=combinacio,
                                        seasonal_order=combinacioS,
                                        enforce_stationarity=False,
                                        enforce_invertibility=False)

        res = mod.fit(maxiter=50)
        ans.append([combinacio, combinacioS, res.bic])
```

Figura 57: Auto arima manual.

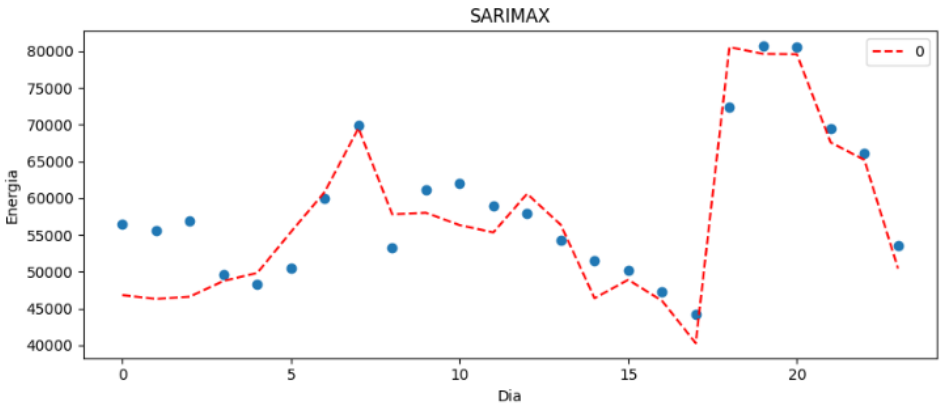
I obtenim els següents combinacions com a top 5 de les millors trobades:

	pdq	pdqs	bic
598	(2, 1, 1)	(0, 1, 1, 24)	21.170533
607	(2, 1, 1)	(1, 1, 1, 24)	25.404639
616	(2, 1, 1)	(2, 1, 1, 24)	26.646637
364	(1, 1, 1)	(1, 1, 1, 24)	182.066234
355	(1, 1, 1)	(0, 1, 1, 24)	183.309878

Figura 58: Resultats auto arima manual.

Cap dels 5 models em baixen la tendència positiva de les dades de tots el 2018.

Anem a comprovar que el problema siguin les dades d'entrada, ja que els dos algorismes han fallat. Si provem amb les dades del gener obtenim:



Amb un MAPE del 19.76%, aplicant el model de *l'auto arima* de la llibreria *pmdarima*. És un indicador que el nostre sistema realment funciona i que era cosa de les dades. Anem a veure els MAPE de tots els mesos del SARIMAX.

Mes \ MAPE	**SARIMAX**
Gener	19.76181
Febrer	22.55590
Març	108.63417
Abril	200.45636
Maig	46.32899
Juny	NAN
Juliol	5.19163
Agost	82.51300
Setembre	16.52590
Octubre	28.15726
Novembre	24.39796
Desembre	300.50055

Han sorgit MAPEs molt elevats. Això és degut a la comparació de valors petits en el model. Per exemple, si tenim un valor real, 200, i el valor predit és 500, obtenim que el MAPE és del 250%, ja que la diferència és 300, més del doble. En canvi, fer aquest mateix error amb valors 20000 i 20500, el MAPE és del 2.5%.

És a dir, les prediccions amb MAPE molt alt representen que fallen molt en valors a petita escala, cosa que no passa amb l'SVR.

9.2.4 Entrenament LSTM

Seguint aquests exemples [21].

El primer pas es inicialitzar el model. Utilitzarem el model més senzill de LSTM, el qual inicialitzem com a *Sequential()* que serveix per agrupar una sèrie de capes (*layers*) en un model.

- La primera capa que crearem serà naturalment la de LSTM, el qual requereix que li indiquem el nombre de cèl·lules o unitats. També indicarem la mida de les dades entrades ja que si no provoca problemes.
- La segona capa serà representada per una *Dense*(1) que representa una capa connectada NN normal. Es fa de mida 1 per obtenir els resultats de sortida. Si es volguessin fer més capes podríem tenir una capa de *Dense*(12) i per acabar una *Dense*(1), però això paletitzaria el temps d'execució però milloraria els resultats del procés.

Veure el codi de les layers a la figura 59.

```
model = Sequential()
model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X, y, epochs=nb_epoch, batch_size=batch_size, verbose=1, shuffle=False)
```

Figura 59: Inicialització model LSTM

En aquesta mateixa figura es troba el *fit* del model, i arribat a aquest punt ja sabem com va, fa el *fit* sobre les dades en format X i y. La peculiaritat que trobem és els paràmetres *epoch, verbose* i *shuffle*.

- ***Epoch.*** Indica quants passos d'entrenament realitzarà el model. S'ha de trobar un nombre de passos raonable, ja que si indiquem molts *epochs* el temps d'execució es dispara. A més, si posem un nombre molt gran es pot produir *overfitting*, per tant, haurem de controlar la pèrdua (los) en cada pas. Sempre busquem la pèrdua mínima.
- ***Verbose.*** Indica la quantitat de *logs* que volem que surtin per pantalla. Essent 0 que no surti cap informació i 2 que surti tota la informació en cada moment de l'execució.
- ***Shuffle.*** Per defecte el *fit* barreja les dades per començar a entrenar el model amb les dades desordenades, però en aquest cas no ens

interessa ja que volem entrenar el model amb les dades ordenades per dia i hora.

I per últim, per fer el *forecast* del nostre model haurem de fer-lo passant-li les dades de test en format X.

```
y = model.predict(X, batch_size=batch_size)
```

Figura 60: Predict LSTM.

9.2.5 Proves LSTM

Les proves en aquest algorisme són poques, ja que l'algorisme ha funcionat bé des del primer moment. A part, l'objectiu d'aquest projecte no és optimitzar un algorisme com LSTM al màxim, sinó desenvolupar una eina que faci *forecast* de les dades del consum d'energia dels edificis.

No necessitem la màxima optimització però la predicció ha de ser competent envers la resta d'algorismes.

Si provem de predir les dades dels dos primers dies del 2019 entrant totes les dades del 2018 obtenim:

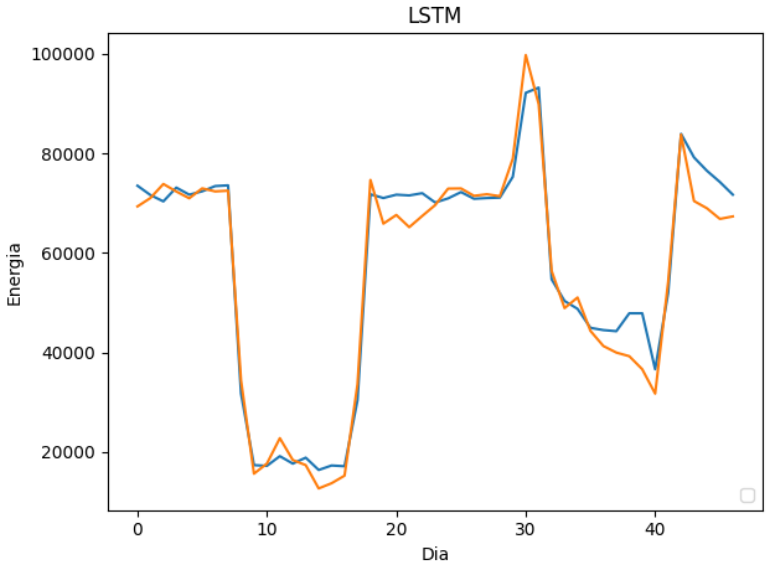


Figura 61: Predicció LSTM del 2019.

Aplicarem només 10 *epochs*. La pèrdua és:

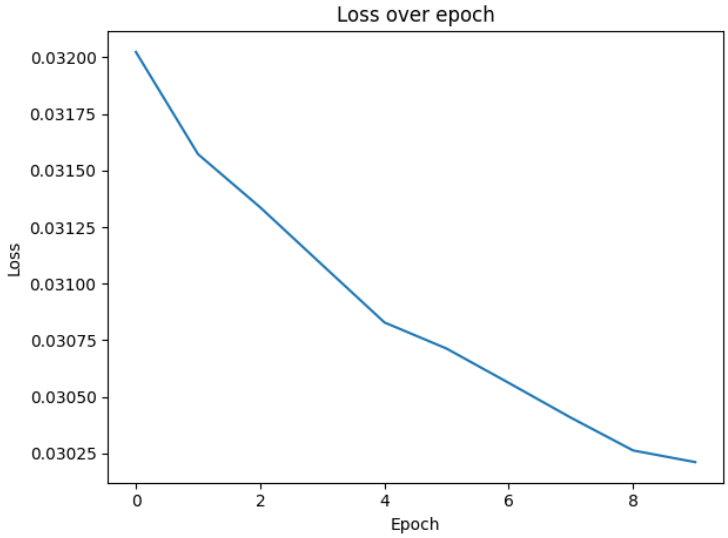


Figura 62: Pèrdua en 100 epochs.

No disminueix molt però tampoc puja, que és el que no volem.

Per veure on es produeix l'*overfitting*, farem una execució amb 1.000 *epochs* per determinar el màxim nombre de passes.

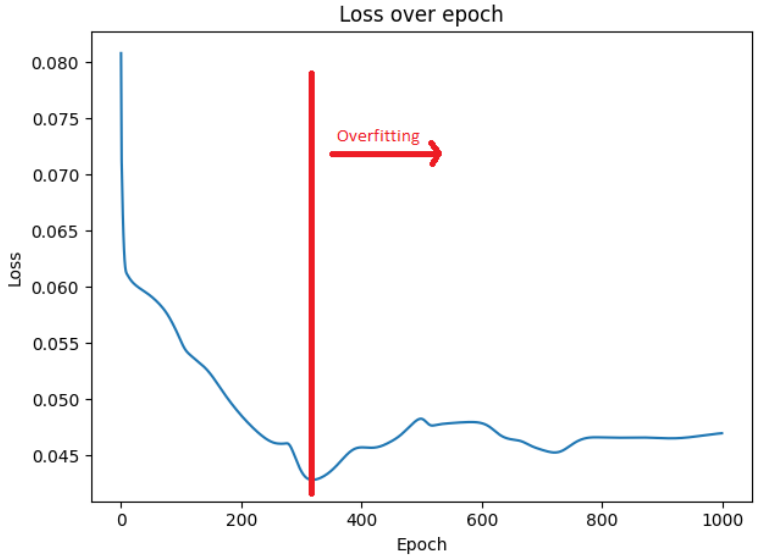


Figura 63: Pèrdua en 1000 epochs.

Per no provocar *overfitting*, no hem de sobrepassar els 300-400 *epochs*. També s'ha de tenir em compte que si volem entrenar el model amb moltes dades és possible que 300 *epochs* resultin en execucions de més d'una hora.

Tenint en compte l'ordinador indicat a l'apartat 2 d'aquest projecte, l'execució de 1.000 *epochs* on les dades d'entrenament corresponen només al mes de gener, obtenim els resultats passats 25 minuts de rellotge.

La decisió final és d'indicar que s'executin 150 *epochs* en totes les execucions.

10. Cas d'ús real

Aquesta eina està pensada per ser executada en bucle. De moment només hem vist la seva execució parcial dels blocs més importants, però és interessant veure com es comportaria si anem entrant dies d'un en un i que el propi algorisme vagi re-calculant el model.

Les dades que utilitzarem seran les de l'edifici 1 del 2018. Les prediccions del nostre model seran d'una setmana.

El primer pas és carregar les dades. El nostre model requereix de com a mínim dades dels 2 últims mesos (conveni acordat amb el tutor del TFG) per funcionar, ja que fer prediccions amb només un mes representen 4 files de dades per cada dia de la setmana, que són pocs per obtenir un bon resultat.

Un cop tenim les dades carregades en el model fem la primera predicció, que correspondrà a la primera setmana d'abril. Apliquem els algorismes i retornem el següent:

```
ACCURACY Mitjanes: 89
ACCURACY SARIMAX: 63
ACCURACY SVR: 96
ACCURACY LSTM: 88
HA SIGUT SVR
```

Figura 64: Predicció primera iteració.

En aquest cas, els errors dels models de mitjanes i l'LSTM han estat molt propers, però SVR obté la millor predicció. SARIMAX es queda amb una precisió petita en comparació a la resta.

La predicció té el següent aspecte, essent la gràfica blava les dades de test i la groga, la predicció.

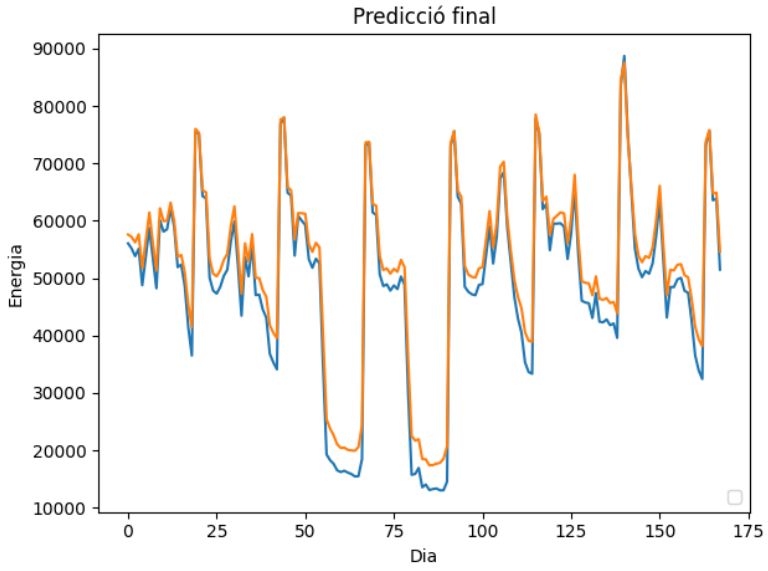


Figura 65: Representació de la primera iteració.

El model ha predit les dades dels següents 7 dies, la primera setmana d'abril. L'auditor entra les dades i treu per pantalla si cal re-calcular el model comparant les nous dies entrats amb els dies emmagatzemats a la predicció.

```
Dia 0
CAL REACALCULAR? False
MAPE: 11.710007
Dia 1
CAL REACALCULAR? False
MAPE: 17.441799
Dia 2
CAL REACALCULAR? True
MAPE: 107.180223
```

Figura 66: Errors de cada dia de la primera iteració.

Veiem que al tercer dia s'ha de re-calcular el model perquè s'ha obtingut un error més gran al 30% en el tercer dia.

Tornem a calcular el model amb els tres dies entrats i obtenim les següents *accuracys*.

```
ACCURACY Mitjanes: 61
ACCURACY SARIMAX: 50
ACCURACY SVR: 88
ACCURACY LSTM: 88
HA SIGUT SVR
```

Figura 67: Predicció segona iteració.

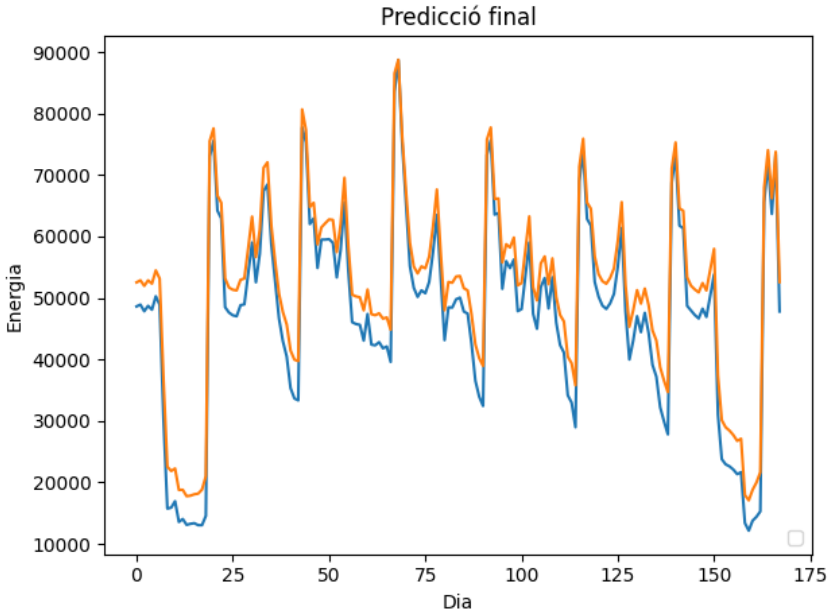


Figura 68: Representació de la segona iteració.

Es re-calcula el primer dia, que és un diumenge. Els errors a petita escala es disparen amb el MAPE, cosa que el fa re-calcular cada cap de setmana (millora al treball futur).

Fem una nova iteració només per veure un cop més el funcionament, però crec que la idea del funcionament de l'eina ha quedat ben reflectida amb aquestes iteracions.

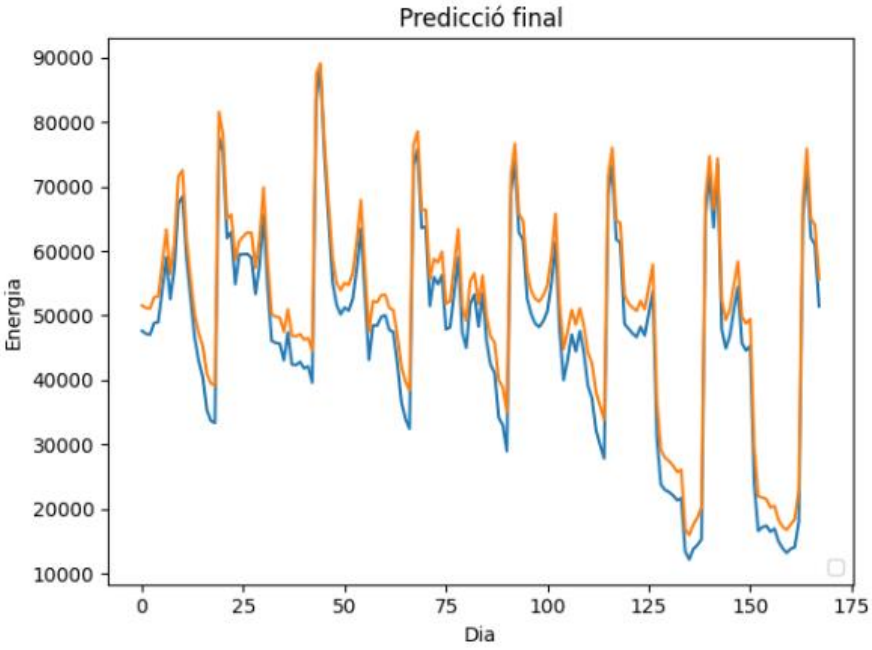


Figura 69: Predicció de la tercera iteració.

11. Conclusions

11.1. Conclusions

11.1.1. Conclusions personals

A nivell personal estic molt content d'haver pogut realitzar aquest treball principalment perquè m'ha permès jugar i provar amb algorismes enfocats a d'intel·ligència artificial, així com acabar de consolidar els coneixements apresos abans de realitzar aquest projecte. Com he comentat al principi de la memòria, estic molt interessat en la IA i vull endinsar-me en aquest àmbit de la informàtica i aquest projecte ha sigut un bon impuls per tirar endavant.

També ha sigut molt interessant la gran part de recerca que he dut a terme abans d'implementar l'aplicació, ja que he descobert algorismes, mètodes i aplicacions que no coneixia. És increïble la quantitat d'algorismes que existeixen en quant a predicció regressiva, i n'existeixen molts més per a altres finalitats que aquí no hem vist.

Haver participat en un projecte a nivell europeu com és *e-Land* ha sigut una motivació a l'hora de tirar endavant l'eina. Ha sigut el primer contacte amb el món de la recerca i he d'admetre que ha estat una experiència molt interessant i enriquidora.

11.1.2. Conclusions tècniques

L'objectiu del projecte era crear una eina que permetés predir el consum d'energia dels edificis d'una forma autònoma i, vist globalment, estic content amb el resultat.

Pel que fa a la recerca, he après a utilitzar algorismes com l'SVR, SARIMAX i xarxes neuronals senzilles. Apart de consolidar conceptes com el tractament de les dades amb la llibreria *pandas*, utilitzar funcionalitats que proporciona *sklearn* i aprofundir, encara que a més petita escala, en la llibreria *Tensorflow*. Considero que *Tensorflow* és una eina molt potent i digna d'aprendre molt més profundament.

He d'admetre que després d'haver investigat tant l'algorisme SARIMAX i haver fet tantes proves, sobretot amb els paràmetres del model, trobo que l'algorisme treu prediccions força dolentes en comparació amb la resta, fins i tot, un model tan senzill com calcular les mitjanes de cada dia per separat obté prediccions més bones. Sens dubte és un punt en contra que no he pogut millorar.

La resta dels objectius del projecte, considero que els he acomplit, l'eina fa el que promet i els algorismes treuen prediccions acceptables. El temps d'execució dels tres models és acceptable en relació a la magnitud de les dades.

11.2. Problemes trobats

11.2.1 Poca flexibilitat amb el dataset

Haver treballat amb les dades dels dos edificis del *Parc Walqa* no ha comportat cap problema, és molt complet i les dades tenen sentit. Em refereixo a que les dades dels caps de setmana són molt diferents als dies entre setmana (a excepció d'algun dia festiu, que contaria com a dada atípica).

El problema l'he trobat al voler incorporar les dades de la resta d'edificis, que conté moltes dades nul·les i intercalades amb dades reals, em refereixo que hi ha dies on hi ha registre de les 12 del matí fins a les 6 de la tarda, deixant prou valors nuls com per no poder-ne treure bones prediccions. Per aquesta raó, la majoria de proves del projecte es fan amb les dades dels edificis del *Parc Walqa* d'Osca.

A més, les dades dels caps de setmana són molt diferents entre si, fent les prediccions del cap de setmana siguin dolentes. A més, coincideix que el MAPE es dispara al calcular l'error dels valors petits, cosa que resulta en que el model es re-calcula cada cap de setmana.

11.2.2. Bug trobat al SARIMAX i el seu baix rendiment

Durant la fase de proves del projecte, em vaig trobar amb un error del SARIMAX que es van resoldre amb versions futures de la llibreria.

El *bug* consistia en que un cop creat el model SARIMAX, si volia estendre el model amb la funció *extend*() sortia un error que deia que les dades d'entrada al model no eren finites, que contenien dades *NaN*. Però si comprovava el nombre de valors *NaN* en les dades d'entrada, em sortia que totes les dades eren finites. Vaig provar de "debuggar" el programa endinsant-me en els fitxers de la llibreria *statsmodels* però no vaig ser capaç de trobar cap error.

També sortia aquest error si es creava un model SARIMAX, s'eliminava i es creava un de nou.

En relació a aquest problema comentar que la primera proposta del projecte només havia de tenir dos algorismes, el SARIMAX i l'SVR, però al veure que SARIMAX no es podia cridar dos cops ni estendre el model i que no treia resultats competents en comparació a l'SVR, es va decidir ampliar el ventall d'algorismes.

12. Treball futur

L'eina dissenyada en aquest projecte és fàcilment ampliable en quant a algorismes predictors. Els mètodes predictors estan modelats per poder-ne acoblar molts sense moltes complicacions. Per tant, una millora immediata al treball seria adjuntar més algorismes perquè competeixin per la millor predicció. Per exemple, un algorisme fàcil d'aplicar seria un algorisme de regressió lineal simple o un model LASSO.

Una altra millora a curt termini seria, de la mateixa manera que comparem diversos algorismes per trobar la millor predicció, aplicar diverses mesures d'error per trobar un error mig entre tots.

També hagués volgut que el SARIMAX obtingués prediccions decents ja que li he dedicat gran part del temps del projecte.

En quant a millores a llarg termini, crec molt interessant aprofundir més sobre les xarxes neuronals i el *Deep Learning*. Hi ha moltes coses a aprendre i millorar on hi ha un gran ventall de possibilitats a implementar.

Considero que és una eina que pot funcionar perfectament en un servidor connectat a qualsevol tipus de base de dades, i una altra possible millora seria col·locar-lo en un servidor i que cada dia s'entrin dades al model i vagi funcionant de forma totalment autònoma.

13. Bibliografia

[1] Towards data science, “The Complete Guide to Time Series Analysis and Forecasting”. Consulta: 31 d’agost del 2021. Disponible a:

https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

[2] Aptech, “Introduction to the Fundamentals of Time Series Data and Analysis”. Consulta: 31 d’agost del 2021. Disponible a:

https://www.aptech.com/blog/introduction-to-the-fundamentals-of-time-series-data-and-analysis/

[3] Wikipèdia, “Artificial intelligence”. Consulta: 31 d’agost del 2021. Disponible a:

https://en.wikipedia.org/wiki/Artificial_intelligence

[4] Wikipedia, “Machine Learning”. Consulta: 31 d’agost del 2021. Disponible a:

https://en.wikipedia.org/wiki/Machine_learning/

[5] IBM, “What is machine learning?”. Consulta: 31 d’agost del 2021. Disponible a:

https://www.ibm.com/cloud/learn/machine-learning

[6] Wikipèdia, “Autoregressive integrated moving average”. Consulta: 31 d’agost del 2021. Disponible a:

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

[7] Otexts, “Forecasting: Principles”. Consulta: 31 d’agost del 2021. Disponible a:

https://otexts.com/fpp2/index.html

[8] Analytics Vidhya, “Understanding Support Vector Machine” . Consulta: 31 d’agost del 2021. Disponible a:

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[9] Documentació pandas. Consulta: 31 d'agost del 2021. Disponible a:

https://pandas.pydata.org/

[10] Statsmodels, "SARIMAX: Introduction". Consulta: 31 d'agost del 2021. Disponible a:

https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html

[11] Kaggle, "How to use SARIMAX". Consulta: 31 d'agost del 2021. Disponible a:

https://www.kaggle.com/poiupoiu/how-to-use-sarimax

[12] Pypi, "Auto arima". Consulta: 31 d'agost del 2021. Disponible a:

https://pypi.org/project/pmdarima/

[13] Scikit-learn, "Machine Learning in Python". Consulta: 31 d'agost del 2021. Disponible a:

https://scikit-learn.org/stable/

[14] Bug SARIMAX, "ValueError: array must not contain infs or NaNs". Consulta: 31 d'agost del 2021. Disponible a tres pàgines:

https://github.com/statsmodels/statsmodels/issues/6907

https://github.com/statsmodels/statsmodels/issues/6542

https://stackoverflow.com/questions/63021311/varmax-results-extend-causes-valueerror-array-must-not-contain-infs-or-nans

[15] jiwidi, "Time series forecasting with Python". Consulta: 31 d'agost del 2021. Disponible a:

https://github.com/jiwidi/time-series-forecasting-with-python/blob/master/time-series-forecasting-tutorial.ipynb

[16] colah, "Understanding LSTM Networks". Consulta: 31 d'agost del 2021. Disponible a:

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[17] Wikipèdia, “Long short-term memory”. Consulta: 31 d’agost del 2021. Disponible a:

https://en.wikipedia.org/wiki/Long_short-term_memory

[18] Xataka, “Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial”. Consulta: 31 d’agost del 2021. Disponible a:

https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial

[19] Saul Dobilas, “Support Vector Regression (SVR) — One of the Most Flexible Yet Robust Prediction Algorithms”. Consulta: 31 d’agost del 2021. Disponible a:

https://towardsdatascience.com/support-vector-regression-svr-one-of-the-most-flexible-yet-robust-prediction-algorithms-4d25fbdaca60

[20] Tharun Peddisetty, “Baby Steps Towards Data Science: Support Vector Regression in Python”. Consulta: 31 d’agost del 2021. Disponible a:

https://towardsdatascience.com/baby-steps-towards-data-science-support-vector-regression-in-python-d6f5231f3be2

[21] Jason Brownlee, “Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras”. Consulta: 31 d’agost del 2021. Disponible a:

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

[22] Wikipèdia, “Artificial neural network”. Consulta: 31 d’agost del 2021. Disponible a:

https://en.wikipedia.org/wiki/Artificial_neural_network

14. Annexos

Classe perfils.py (auditor)

```
import numpy as np
from scipy.stats import ttest_ind
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.svm import SVR
from sklearn.preprocessing import MinMaxScaler
import math as math


class Perfils:

    # Dataframe
    dfPerfils = []
    dfMeans = []

    prediccio = []
    tolerancia = 30

    dadesColumna = 'p_Edificio_1'


    # Constructor
    def __init__(self):
        for i in range(0, 7):
            self.dfPerfils.append(pd.DataFrame(columns=range(0, 24)))
            self.dfMeans.append([0 for i in range(24)])

    # Train
    def train(self, dfDia, dia):
        if 0 <= dia <= 6:
            self.dfPerfils[dia] = self.appendRow(self.dfPerfils[dia], dfDia.values)
        return None

    # Afegeix i retorna el dataframe entrat amb la fila entrada
    def appendRow(self, df, row):
        if len(df.columns) != len(row):
            return df
        else:
            dictOfValues = {i: row[i] for i in range(0, len(row))}
            return df.append(dictOfValues, ignore_index=True)

    def getPerfils(self):
        return self.dfPerfils

    def getMeans(self):
        return self.dfMeans

    def fillnan(self, X):
        return X.fillna(X.mean(), inplace=True)

    # Calcula MAPE entre dos datasets
    def mape(self, actual, pred):
        actual, pred = np.array(actual), np.array(pred)
        return np.mean(np.abs((actual - pred) / actual)) * 100
        #return mean_absolute_error(actual, pred)

    def calRecalcular(self, dfNou, dia):
        mape = self.mape(dfNou, self.prediccio[(24 * dia - 1):(24 * dia)])

        if self.mape(dfNou, self.prediccio[(24*dia-1):(24*dia)]) > self.tolerancia:
            return mape, True
        return mape, False

    # Actualitza totes les mitjanes
    def updateMeans(self):
        for i in range(0, 24):
            for j in range(0, 7):
                self.dfMeans[j][i] = np.nanmean(self.dfPerfils[j][i])


    def fitTotal(self, df_training, dia, df_test):
        max = 0
        #Executem tots els algorismes
        accM, predM = self.fitMitjanes(df_training[self.dadesColumna], dia,
df_test[self.dadesColumna])
```

```
        accSAR, predSAR = self.fitSARIMAX(df_training[self.dadesColumna], dia,
df_test[self.dadesColumna])
        accSVR, predSVR = self.fitSVR(df_training, df_test)
        accLSTM, predLSTM = self.fitLSTM(df_training[self.dadesColumna].values,
df_test[self.dadesColumna].values)

        print('ACCURACY Mitjanes: %.f' % accM)
        print('ACCURACY SARIMAX: %.f' % accSAR)
        print('ACCURACY SVR: %.f' % accSVR)
        print('ACCURACY LSTM: %.f' % accLSTM)
        # EM GUARDO LA PREDICCIÓ
        if accSVR > max:
            self.prediccio = predSVR
            print('HA SIGUT SVR')
        elif accSAR > max:
            self.prediccio = predSAR
            print('HA SIGUT SARIMAX')
        elif accLSTM > max:
            self.prediccio = predLSTM
            print('HA SIGUT LSTM')
        else:
            self.prediccio = predM
            print('HA SIGUT PER MITJANES')

        plt.plot(df_test[self.dadesColumna].values)
        plt.plot(self.prediccio)
        plt.title('Predicció final')
        plt.xlabel('Dia')
        plt.ylabel('Energia')
        plt.legend(loc='lower right')
        plt.show()
        return None

    def fitLSTMTotal(self, df_training, dia, df_test):
        pred = []
        for i in range(7):
            a = self.fitLSTM(df_training, df_test)
            pred.append(a)
        return pred

    def fitLSTM(self, df_training, df_test):

        print('--> LSTM Initialize')
        raw_values = df_training
        diff_values = self.difference(raw_values, 1)
        raw_values_test = df_test
        diff_values_test = self.difference(raw_values_test, 1)
        print('Scaling values')
        supervised = self.timeseries_to_supervised(diff_values, 1)
        supervised_values = supervised.values
        supervised = self.timeseries_to_supervised(diff_values_test, 1)
        supervised_values_test = supervised.values

        train, test = supervised_values, supervised_values_test

        # provar de ferlo amb l'altre transformador
        scaler, train_scaled, test_scaled = self.scale(train, test)

        print('LSTM Executing...')
        print('Fitting...')
        lstm_model, model_loss_acc = self.fit_lstm(train_scaled, 1, 150, 4)
        train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)

        print('Predicting...')
        lstm_model.predict(train_reshaped, batch_size=1)

        predictions = []
        for i in range(len(test_scaled)):
            X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
            yhat = self.forecast_lstm(lstm_model, 1, X)
            yhat = self.invert_scale(scaler, X, yhat)
            yhat = self.inverse_difference(raw_values, yhat, len(test_scaled) + 1 - i)
            predictions.append(yhat)

        predictions = np.array(predictions)

        plt.plot(raw_values_test[:len(test_scaled)])
        plt.plot(predictions)
        plt.title('LSTM')
        plt.xlabel('Dia')
        plt.ylabel('Energia')
        plt.legend(loc='lower right')
        plt.show()

        self.summarize_results(model_loss_acc.history)

        return 100 - self.mape(raw_values_test[:len(test_scaled)], predictions), predictions
```

```
    def fitMitjanes(self, df_training, dia, df_test):
        # Actualitzem les mitjanes per poder fer el càlcul actualitzat
        self.updateMeans()

        print('ACURACY:  ')
        print(100 - self.mape(df_test[:24], self.dfMeans[dia]))

        return 100 - self.mape(df_test[:24], self.dfMeans[dia]), self.dfMeans[dia]


    def fitSVR(self, df_training, df_test):

        print('--> SVR Initialize')
        X = df_training.reset_index(drop=True)[['index', self.dadesColumna]]
        X[['index']] = [math.fmod(i, 24).__int__() for i in range(len(X.index))]
        X_test = df_test.reset_index(drop=True)[['index', self.dadesColumna]]
        X_test[['index']] = [math.fmod(i, 24).__int__() for i in range(len(X_test.index))]

        y = df_training.reset_index(drop=True)[[self.dadesColumna]]
        y_test = df_test.reset_index(drop=True)[[self.dadesColumna]]
        print('Dataframes created succesfully')

        print('Deleting NANs')
        X.fillna(X.mean(), inplace=True)
        X_test.fillna(X.mean(), inplace=True)
        y.fillna(y.mean(), inplace=True)

        print('SVR Executing...')
        svr = SVR(kernel='rbf', C=1, gamma='scale', epsilon=.1)

        # For SVR we need Feature Scaling
        from sklearn.preprocessing import StandardScaler
        sc_x = StandardScaler()
        sc_y = StandardScaler()

        # Scale x and y (two scale objects)
        X = sc_x.fit_transform(X)
        X_test_t = sc_x.fit_transform(X_test)
        y = sc_y.fit_transform(y.values.reshape(-1, 1))

        print('Predicting...')
        svr.fit(X, y.ravel())
        predict = svr.predict(X_test_t)


        print('Plot generated')

        # Graph
        fig, ax = plt.subplots(figsize=(9, 4))

        npre = 4
        ax.set(title='SVR', xlabel='Dia', ylabel='Energia')

        # Plot data points
        y_test = pd.DataFrame(y_test)
        y_test.plot(ax=ax, style='o', label='Observat')

        # Plot predictions
        predict = pd.DataFrame(sc_y.inverse_transform(predict)).rename(columns={0: 'Forecast'})
        predict.plot(ax=ax, style='r--', label='SVR')

        legend = ax.legend(loc='lower right')

        plt.show()

        print('MAPE:  %.5f' % self.mape(y_test, predict.values))

        return 100 - self.mape(y_test, predict.values), predict.values


    def fitSARIMAXTotal(self, df_training, dia, df_test):
        pred = []
        for i in range(7):
            a = self.fitSARIMAX(df_training, ((dia + i) % 7), df_test)
            pred.append(a)
        return pred

    def fitSARIMAX(self, dfDia, dia, test):
        print('--> SARIMAX Initialize')

        print('Generate regresion')

        dfDia = dfDia.values
        df = []
        for i in range(0, len(self.dfPerfils[dia])):
            if len(df) == 0:
                df = self.dfPerfils[dia].iloc[[i]].values[0]
```

```
            # Concatting 2 arrays
            df = [*df,  *self.dfPerfils[dia].iloc[[i]].values[0]]

        # df = pd.DataFrame(df)

        '''

        AUTO ARIMA
        from pmdarima.arima import auto_arima
        stepwise_model = auto_arima(df,start_p=0,d=1,start_q=0,
                                    max_p=5,max_d=5,max_q=5, start_P=0,
                                    D=1, start_Q=0, max_P=5,max_D=5,
                                    max_Q=5, m=24, seasonal=True,
                                    error_action='warn',trace=True,
                                    supress_warnings=True,stepwise=True,
                                    random_state=20,n_fits=50)

        print(stepwise_model.aic())
        '''

        '''
        # GRID SEARCH MANUAL

        p = d = q = range(0, 2)
        pdq = list(itertools.product(p, d, q))

        print(self.sarimax_gridsearch(dfDia[self.dadesColumna].values, pdq, pdqs, freq='D'))

        '''
        print('Fitting...')
        sarimax = sm.tsa.statespace.SARIMAX(endog=dfDia,
                                            trend='n',
                                            order=(2, 1, 1),
                                            seasonal_order=(0, 1, 1, 24))


        res = sarimax.fit(maxiter=50, disp=False)

        predict = res.forecast(steps=168, exog=test[:168])

        # Graph
        fig, ax = plt.subplots(figsize=(9, 4))
        npre = 4
        ax.set(title='SARIMAX', xlabel='Dia', ylabel='Energia')

        # Plot data points
        test = test.reset_index(drop=True)
        test.plot(ax=ax, style='o', label='Observat')


        # Plot predictions
        predict = pd.DataFrame(predict)
        predict = predict.reset_index(drop=True)
        predict.plot(ax=ax, style='r--', label='SARIMAX')

        plt.show()

        print('Plot generated')

        print('MAPE:  %.5f' % self.mape(test, predict.values))

        return 100 - self.mape(test, predict.values), predict.values



    ##########  UTILS   ###########
    def summarize_results(self, history):
        f = plt.figure()
        # pltot loss
        plt.plot(history['loss'], label='train')
        plt.ylabel("Loss")
        plt.xlabel("Epoch")
        plt.title("Loss over epoch")
        plt.show()

    # frame a sequence as a supervised learning problem
    def timeseries_to_supervised(self, data, lag=1):
        df = pd.DataFrame(data)
        columns = [df.shift(i) for i in range(1, lag + 1)]
        columns.append(df)
        df = pd.concat(columns, axis=1)
        df.fillna(0, inplace=True)
        return df

    # create a differenced series
    def difference(self, dataset, interval=1):
        diff = list()
        for i in range(interval, len(dataset)):
```

```
            value = dataset[i] - dataset[i - interval]
            diff.append(value)
        return pd.Series(diff)

    # invert differenced value
    def inverse_difference(self, history, yhat, interval=1):
        return yhat + history[-interval]

    # scale train and test data to [-1, 1]
    def scale(self, train, test):
        # fit scaler
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(train)
        # transform train
        train = train.reshape(train.shape[0], train.shape[1])
        train_scaled = scaler.transform(train)
        # transform test
        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)
        return scaler, train_scaled, test_scaled

    # inverse scaling for a forecasted value
    def invert_scale(self, scaler, X, value):
        new_row = [x for x in X] + [value]
        array = np.array(new_row)
        array = array.reshape(1, len(array))
        inverted = scaler.inverse_transform(array)
        return inverted[0, -1]

    # fit an LSTM network to training data
    def fit_lstm(self, train, batch_size, nb_epoch, neurons):
        X, y = train[:, 0:-1], train[:, -1]
        X = X.reshape(X.shape[0], 1, X.shape[1])
        model = Sequential()
        model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]),
stateful=True))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        model_loss_acc = model.fit(X, y, epochs=nb_epoch, batch_size=batch_size, verbose=1,
shuffle=False)
        return model, model_loss_acc

    # make a one-step forecast
    def forecast_lstm(self, model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        y = model.predict(X, batch_size=batch_size)
        return y[0, 0]

    ##########  SARIMAX UTILS   ##############

    # Define function
    def sarimax_gridsearch(self, df, pdq, pdqs, maxiter=50):

        # Run a grid search with pdq and seasonal pdq parameters and get the best BIC value
        ans = []
        for combinacio in pdq:
            for combinacioS in pdqs:
                mod = sm.tsa.statespace.SARIMAX(df,
                                                order=combinacio,
                                                seasonal_order=combinacioS,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                res = mod.fit(maxiter=50)
                ans.append([combinacio, combinacioS, res.bic])

        res = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'bic'])
        res = res.sort_values(by=['bic'], ascending=True)[0:20]
        return res
```