

Pràctica 2: Classificació

Aprenentatge Computacional

1. Objectius

L'objectiu d'aquesta segona pràctica és analitzar el nostre dataset *Mobile Price Classification* i classificar cadascuna de les entrades segons la seva gama de preu mitjançant la creació de diferents models de classificació que estudien les nostres dades i a partir de les seves prediccions permeten generar noves conclusions.

2. Anàlisi del dataset

Partim d'una base de dades que consta de 21 atributs de caràcter numèric (la gran majoria de tipus int64). Aquesta consisteix en un total de 2000 mostres amb diferents característiques de telèfons mòbils.

Informació de cada atribut:

ÍNDEX	VARIABLE	
1	id	
2	battery_power	Energia total que la bateria pot emmagatzemar en un sol temps mesurada en mAh
3	blue	Si té bluetooth o no
4	clock_speed	Velocitat a la que el micorprocessador executa les instruccions
5	dual_sim	Si té suport de la dual sim o no
6	fc	Megapixels de la càmara frontal
7	four_g	Si té 4G o no
8	int_memory	Memòria interna en Gigabytes
9	m_dep	Porfunditat del mòbil en cm
10	mobile_wt	Pes del mòbil
11	n_cores	Número de cores del processador
12	pc	Megapixels de la càmara principal
13	px_height	Alçada de la resolució de pixel
14	px_width	Amplada de la resolució de pixel

15	ram	Memòria d'accés aleatori en megabytes
16	sc_h	Alçada de la pantalla del mòbil en cm
17	sc_w	Amplada de la pantalla del mòbil en cm
18	talk_time	Temps més llarg que durarà una única càrrega de la bateria quan esteu parlant
19	three_g	Si té 3G o no
20	touch_screen	Si té pantalla tàctil o no
21	wifi	Si té wifi o no

Finalment tenim el nostre atribut **price_range** que està dividit en 4 possibles categories (0,1,2,3) essent 0 la gama de mòbils més barata i 3 la més cara.

El nostre dataset ve donat per dos arxius anomenats *train.csv* i *test.csv*, però per a l'objectiu de classificació de la nostra pràctica només farem servir *train.csv* ja que és l'únic que inclou l'atribut de **price_range** que és el nostre interès principal per a la classificació, per tant tractarem aquest datatset com a un únic, no comal conjunt de dades d'entrenament.

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.200	0	1	0	7	0.600	188	2	...	20	756	2549	9	7	19
1	1021	1	0.500	1	0	1	53	0.700	136	3	...	905	1988	2631	17	3	7
2	563	1	0.500	1	2	1	41	0.900	145	5	...	1263	1716	2603	11	2	9
3	615	1	2.500	0	0	0	10	0.800	131	6	...	1216	1786	2769	16	8	11
4	1821	1	1.200	0	13	1	44	0.600	141	2	...	1208	1212	1411	8	2	15

5 rows × 21 columns

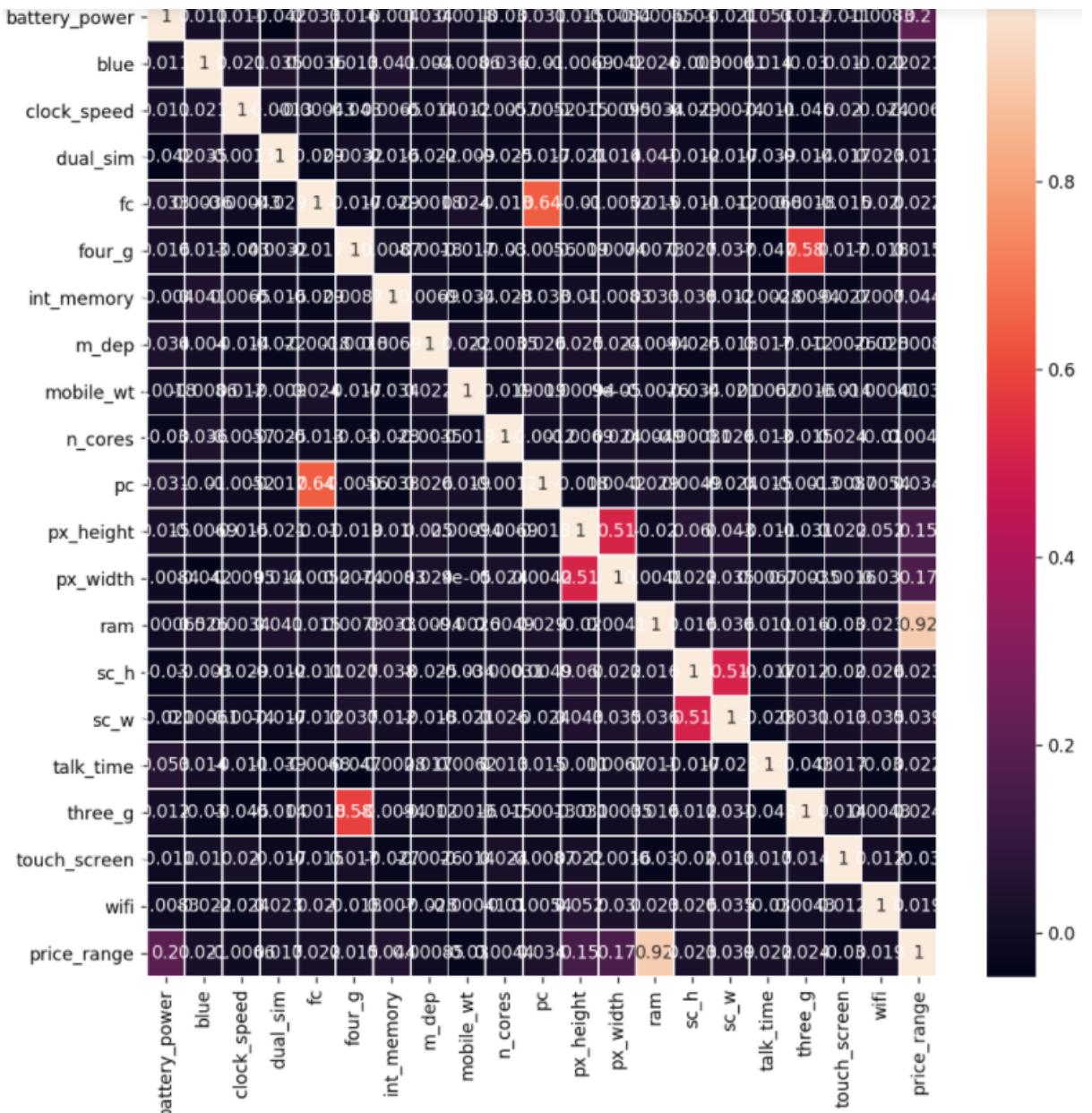
	three_g	touch_screen	wifi	price_range
	0	0	1	1
	1	1	0	2
	1	1	0	2
	1	0	0	2
	1	1	0	1

3. Disseny de la solució

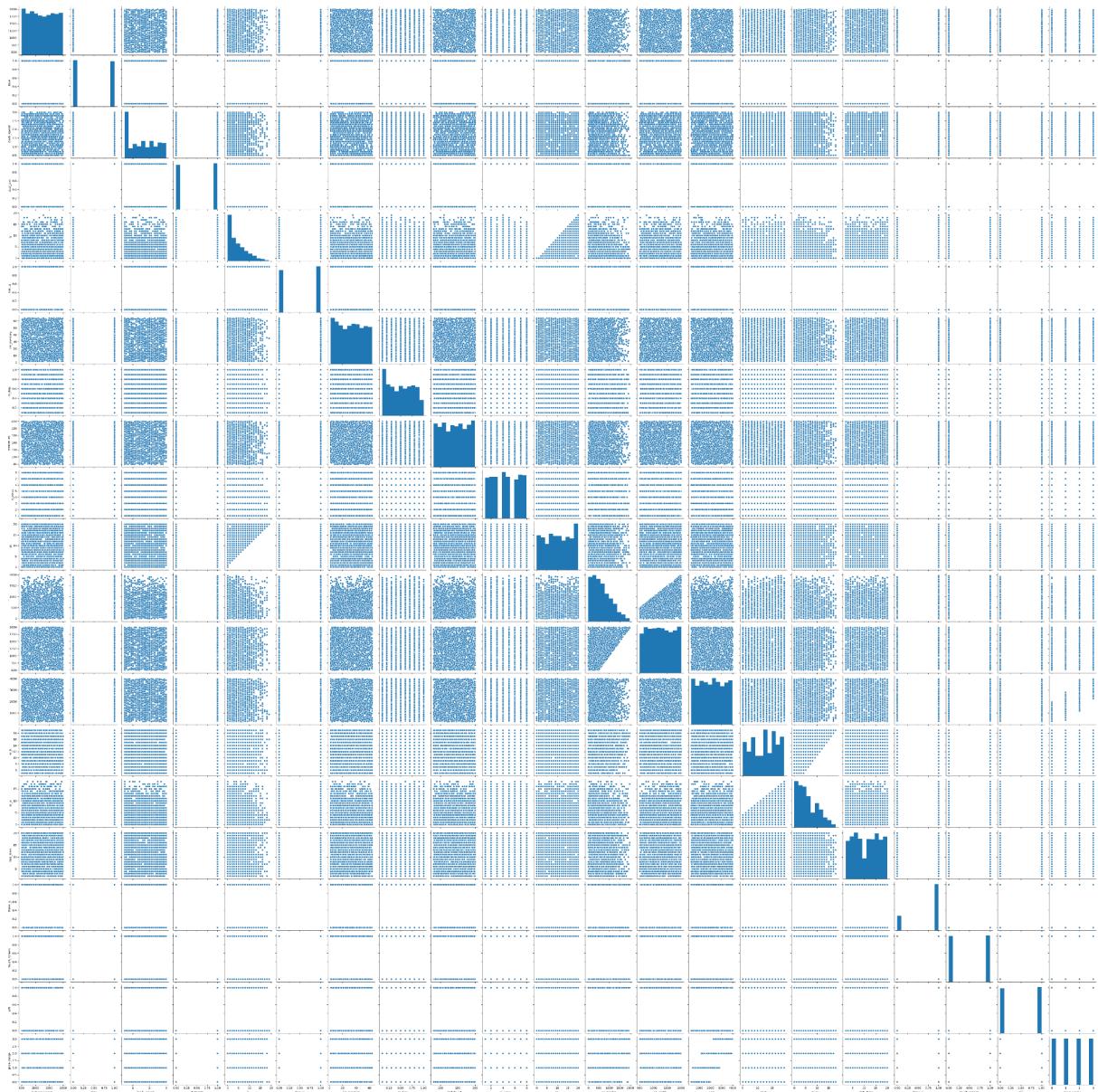
El primer que fem amb el nostre dataset és tractar de netejar-lo. Comencem per veure si trobem atributs nuls o poques dades i observem que no hi ha cap. Veiem com estan representades les dades.

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram
count	2000.000	2000.000	2000.000	2000.000	2000.000	2000.000	2000.000	2000.000	2000.000	2000.000	...	2000.000	2000.000	2000.000
mean	1238.518	0.495	1.522	0.509	4.309	0.521	32.047	0.502	140.249	4.521	...	645.108	1251.515	2124.213
std	439.418	0.500	0.816	0.500	4.341	0.500	18.146	0.288	35.400	2.288	...	443.781	432.199	1084.732
min	501.000	0.000	0.500	0.000	0.000	0.000	2.000	0.100	80.000	1.000	...	0.000	500.000	256.000
25%	851.750	0.000	0.700	0.000	1.000	0.000	16.000	0.200	109.000	3.000	...	282.750	874.750	1207.500
50%	1226.000	0.000	1.500	1.000	3.000	1.000	32.000	0.500	141.000	4.000	...	564.000	1247.000	2146.500
75%	1615.250	1.000	2.200	1.000	7.000	1.000	48.000	0.800	170.000	7.000	...	947.250	1633.000	3064.500
max	1998.000	1.000	3.000	1.000	19.000	1.000	64.000	1.000	200.000	8.000	...	1960.000	1998.000	3998.000
	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range							
2000.000	2000.000	2000.000	2000.000		2000.000	2000.000	2000.000							
12.306	5.767	11.011	0.761		0.503	0.507	1.500							
4.213	4.356	5.464	0.426		0.500	0.500	1.118							
5.000	0.000	2.000	0.000		0.000	0.000	0.000							
9.000	2.000	6.000	1.000		0.000	0.000	0.750							
12.000	5.000	11.000	1.000		1.000	1.000	1.500							
16.000	9.000	16.000	1.000		1.000	1.000	2.250							
19.000	18.000	20.000	1.000		1.000	1.000	3.000							

A continuació visualitzem la correlació dels nostres atributs.



És bastant evident veure que la gran majoria dels atributs del nostre dataset són despreciables, tenint en compte la correlació que tenen amb el nostre atribut objectiu. Observem una correlació lineal baixa per a qüasi totes les variables, moderada per al battery_power, px_height i px_width i alta per a la ram.



La distribució entre les categories no sembla ser gaire similar (les tiquetes no estan gaire balancejades), podem observar que hi ha categories que segueixen una distribució binària, d'altres una exponencial negativa, altres els valors semblen ser molt similars, etc. Això podríà generar problemes especialment si tenim un conjunt de training amb unes distribucions molt diferents al conjunt de test.

Un cop vistes les dades de les que es disposa, per tal de tenir un aprenentatge més eficient, passem a normalitzar les dades i treure outliers. Les nostres dades no estan normalitzades i seria important fer-ho ja que necessitem que es trobin dins d'un rang similar per a tenir una bona contribució dins l'anàlisi i que no generin biaix.

Procedim a fer la normalització més adient per a les nostres dades que en aquest cas sembla ser la normalització estàndard on estandarditzen les característiques eliminant la mitjana i escalant a la variància de la unitat.

	battery_power	ram	px_height	px_width
count	2000.000	2000.000	2000.000	2000.000
mean	0.000	-0.000	0.000	0.000
std	1.000	1.000	1.000	1.000
min	-1.679	-1.723	-1.454	-1.739
25%	-0.880	-0.845	-0.817	-0.872
50%	-0.028	0.021	-0.183	-0.010
75%	0.858	0.867	0.681	0.883
max	1.729	1.728	2.964	1.728

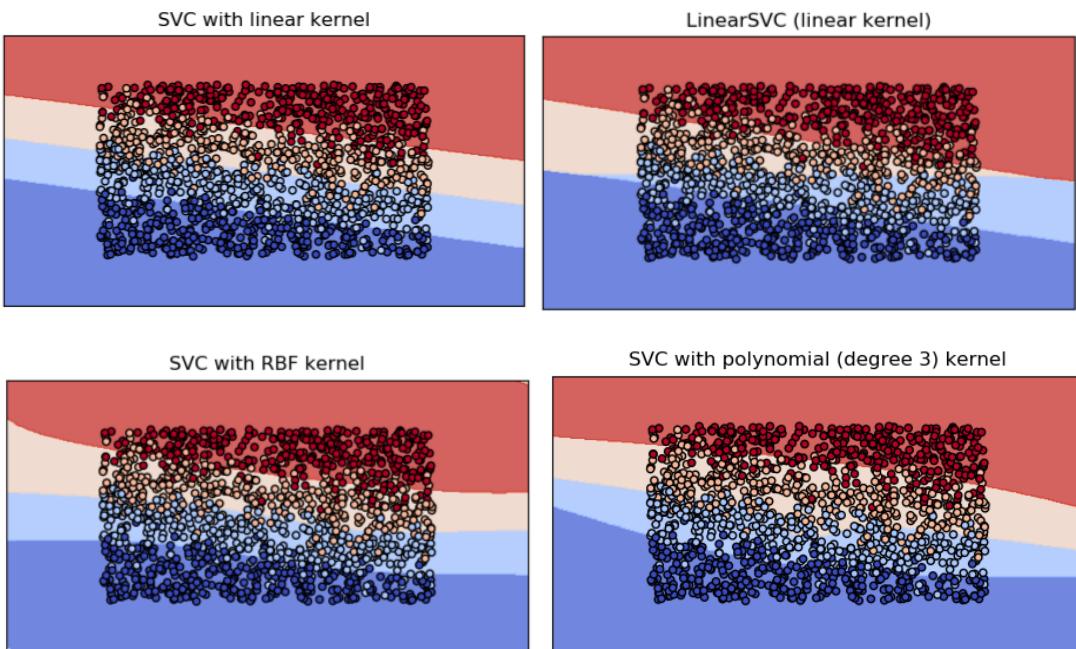
Observeu que les nostres dades han estat normalitzades i han millorat considerablement.

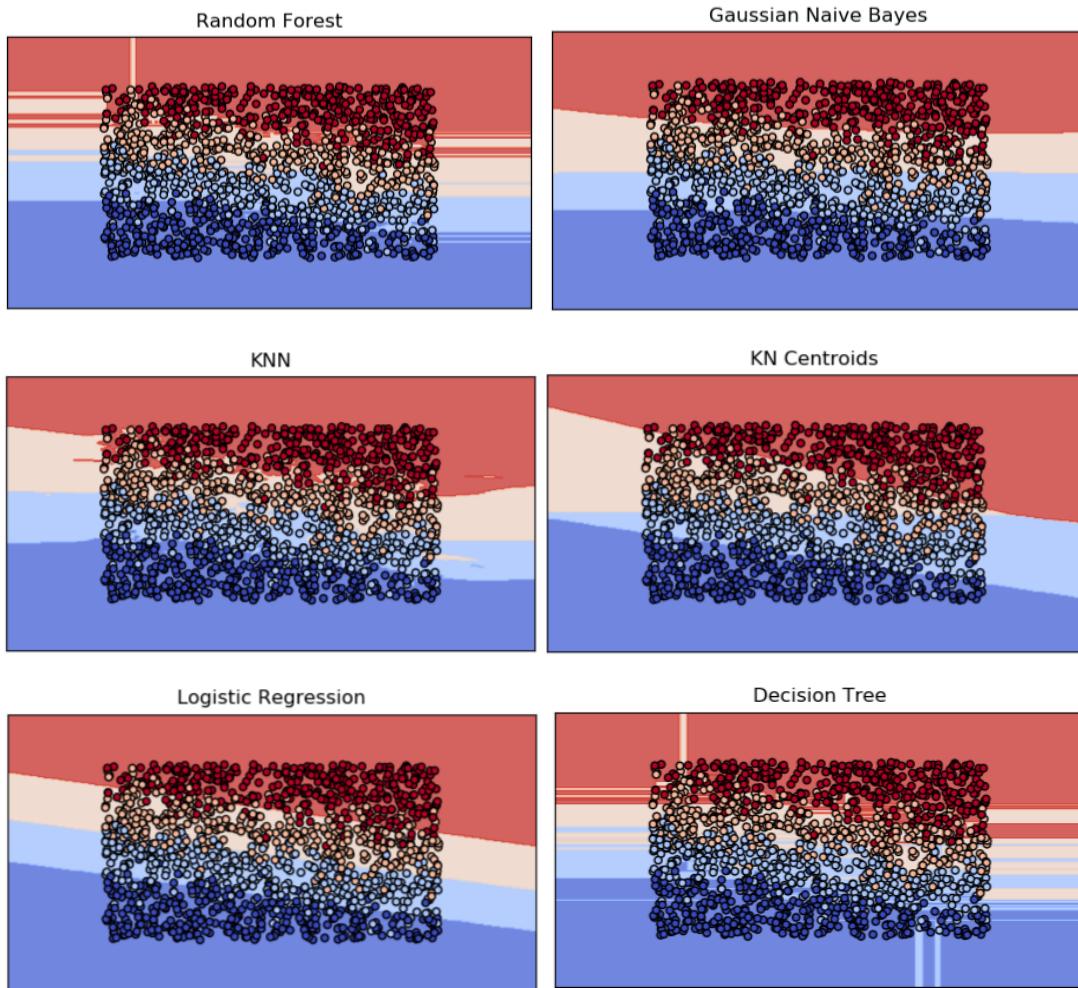
La tasca d'aquesta pràctica s'enmarca dins l'aprenentatge computacional **supervisat**. A continuació hem triat les tècniques que volem fer servir, en el nostre cas mirarem SVC with linear kernel', LinearSVC (linear kernel), SVC with RBF kernel, SVC with polynomial (degree 3) kernel, Random Forest, Gaussian Naive Bayes, KNN, KN Centroids, Decision Tree i Logistic Regression. La decisió per a triar aquests models ve donada perquè es tracta dels models més utilitzats a l'hora de fer classificació supervisada i ens aporten unes accuràcies molt bones amb el nostre dataset. Frem servir dos mètodes d'ensemble (Random Forest i Decision Tree) ja que aquests són molt bons a l'hora de classificar (molt bona accuracy), però molt costosos en temps.

Una vegada hem fet la selecció dels models els avaluem tots amb tres particions del dataset possibles, en el nostre cas hem seleccionat particions de 0.5, 0.7 i 0.8. Si avaluem tots els nostres models amb cadascuna de les particions, obtenim les millors classificacions amb la partició de 0.7, per tant procedim a dividir el nostre dataset entre train i test amb aquest rang de 0.7-0.3.

Correct classification Logistic	0.5 % of the data: 0.966
Correct classification SVM rbf	0.5 % of the data: 0.905
Correct classification SVM linear	0.5 % of the data: 0.963
Correct classification SVM poly	0.5 % of the data: 0.952
Correct classification Random Forest	0.5 % of the data: 0.775
Correct classification Gaussian Naive Bayes	0.5 % of the data: 0.808
Correct classification KNN	0.5 % of the data: 0.861
Correct classification KN Centroids	0.5 % of the data: 0.779
Correct classification Logistic	0.7 % of the data: 0.963333333333334
Correct classification SVM rbf	0.7 % of the data: 0.9266666666666666
Correct classification SVM linear	0.7 % of the data: 0.9616666666666667
Correct classification SVM poly	0.7 % of the data: 0.9566666666666667
Correct classification Random Forest	0.7 % of the data: 0.815
Correct classification Gaussian Naive Bayes	0.7 % of the data: 0.8066666666666666
Correct classification KNN	0.7 % of the data: 0.8666666666666667
Correct classification KN Centroids	0.7 % of the data: 0.74
Correct classification Logistic	0.8 % of the data: 0.955
Correct classification SVM rbf	0.8 % of the data: 0.895
Correct classification SVM linear	0.8 % of the data: 0.9575
Correct classification SVM poly	0.8 % of the data: 0.955
Correct classification Random Forest	0.8 % of the data: 0.78
Correct classification Gaussian Naive Bayes	0.8 % of the data: 0.7975
Correct classification KNN	0.8 % of the data: 0.845
Correct classification KN Centroids	0.8 % of the data: 0.77

Agafem com a valor de la X dos atributs, *battery_power* i *ram* ja que es tracten dels dos amb una correlació més alta en relació atribut objectiu. Un cop feta la selecció, passem a avaluar la classificació de cadascun dels models i les seves accuracy.





```

Accuracy of SVC with linear kernel : 0.8257142857142857
Accuracy of LinearSVC (linear kernel) : 0.7607142857142857
Accuracy of SVC with RBF kernel : 0.8214285714285714
Accuracy of SVC with polynomial (degree 3) kernel : 0.7864285714285715
Accuracy of Random Forest : 1.0
Accuracy of Gaussian Naive Bayes : 0.7664285714285715
Accuracy of KNN : 0.8721428571428571
Accuracy of KN Centroids : 0.7907142857142857
Accuracy of Logistic Regression : 0.8242857142857143
Accuracy of Decision Tree : 1.0

```

Com no hem indicat una profunditat màxima per al Random Forest o el Decision Tree ens arriba a classificar correctament totes les mostres, però això pot arribar a ser molt costós en temps. El KNN sembla ser el següent classificador que millor ho fa amb un accuracy de 0.87, per tant seleccionem aquest classificador per a realitzar els càlculs restants.

Un cop seleccionats quins models es volen testejar sobre les dades, s'han de poder avaluar correctament. Per aquests motius, haurem d'aprendre a cros-validar els resultats. Es extremadament important fer servir la cross-validation per avaluar l'efectivitat del model, especialment en els casos on trobem overfitting. També és útil per determinar els hiperparàmetres del model (quins paràmetres donaran lloc a un error de test més baix).

Dins de la cross-validation hem de tenir el nostre conjunt de dades ben separat ja que si tenim moltes dades d'entrenament ens podem trobar amb un overfitting i amb poques underfitting. Nosaltres ja l'hem dividit en dos conjunts fent servir el rang 70-30 que ens evita aquests problemes.

L'estrategia que hem seleccionar per a fer la cross.validation és el mètode de KFold. El mètode K-Fold Cross-Validation és també un procés iteratiu. Consisteix a dividir les dades de forma aleatòria en k grups d'aproximadament la mateixa mida, $k-1$ grups s'utilitzen per entrenar el model i un dels grups s'empra com a validació. Aquest procés es repeteix k vegades utilitzant un grup diferent com a validació a cada iteració. El procés genera k estimacions de l'error la mitjana del qual s'utilitza com a estimació final.

Avaluem les nostres dades amb el classificador de KNN i deixem que prohi amb diferents k (número de plecs). Observem que obtenim l'accuracy més gran (0.79437) amb $k = 4$, així que tindrem 4 grups. Amb els altres valors de la k veiem que el nostre accuracy puja i baixa però no gaire (una diferència de 10^{-3}).

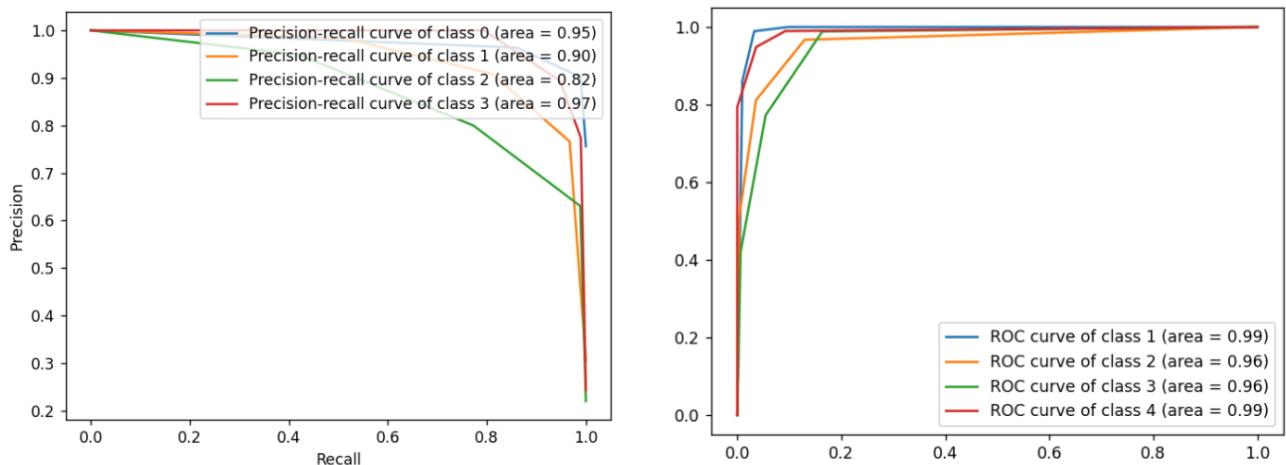
Hi ha altres estratègies per a fer la corss-validation com ara *LeaveOneOut*, però no considerem que si convenient aplicar-la ja que es tracta només d'una especialització de KFold que agafa $k=n$ grups i tenint en compte que el nostre dataset té 2000 mostres, triigaría moltíssim en executar-se. La millor estratègia que hem trobat per a fer aquesta cross-validation és el KFold en temps i en resultats.

A l'hora d'avaluar com de bé estan funcionant els models que estem utilitzant podem utilitzar diferents mètriques de classificació, però no totes són correctes, en funció del tipus de dades que utilitzem serà més útil utilitzar-ne una o altra. Hem de partir de la base que les nostres dades estan força balancejades donat que cada tenim més o

menys la mateixa quantitat a cada classe possible. És per això que la millor mètrica serà l'accuracy_score, ja que tant tant l'f1_score com l'average_precision_score es centren més en els valors ben classificats que en els mal classificats i això podria acabar produint una falsa impressió de que un model està funcionant molt bé quan no és veritat. Una altra manera de decidir quina és la millor mètrica és utilitzar el classification report. Aquest ens dóna la precisió, el recall i l'f1-score de cada classe. Podem observar que en el nostre cas funciona molt bé amb precisions molt elevades.

	precision	recall	f1-score	support
0	0.91	1.00	0.95	93
1	0.92	0.86	0.89	122
2	0.85	0.82	0.83	88
3	0.93	0.95	0.94	97
accuracy			0.91	400
macro avg	0.90	0.91	0.90	400
weighted avg	0.90	0.91	0.90	400

Una altra cosa interessant a mirar del nostre dataset seria la representació de les corbes Precisió-Recall i ROC (a esquerra i dreta respectivament).



L'objectiu per a totes dues corbes és tenir una àrea sota aquestes de 1, de manera que es veu que els resultats són molt bons. Tot i això, com que el nostre dataset té les classes força balançejades el millor és utilitzar la corba ROC.

Fins ara hem estat evaluant els nostres models de classificació o bé amb els paràmetres que ja tenen predefinits o bé amb valors a semialeatoris que nosaltres els donàvem, però hi ha dos mètodes que ens permeten trobar quins valors hauríem de donar als paràmetres per obtenir els millors resultats possibles, i ho fan provant diferents valors d'un conjunt que l'usuari passa com a argument. Aquests dos mètodes són el Randomized Parameter Optimization i el Exhaustive Grid Search. Si bé els dos són perfectament aptes, si disposem de temps i recursos limitats per a dur a terme aquest procés el millor és utilitzar el Randomized, donat que a part de que l'Exhaustive és més costós computacionalment també pot ser que no arribi a trobar mai el valor que més ens interessa i en canvi el Randomized té moltes més probabilitats de trobar-lo. També existeixen altres mètodes com podria ser per exemple el BayesSearchCV però segueix essent menys eficaç.

Després d'aquest procés, hem trobat que els millor valors eren els següents:

- Gaussian Naive Bayes: {'var_smoothing': 0.43287612810830584}
- KNN: {'leaf_size': 1, 'n_neighbors': 38, 'p': 1}
- Logistic Regression: {'C': 0.0001, 'penalty': 'none'}
- Decision Tree: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 5}

4. Apartat B

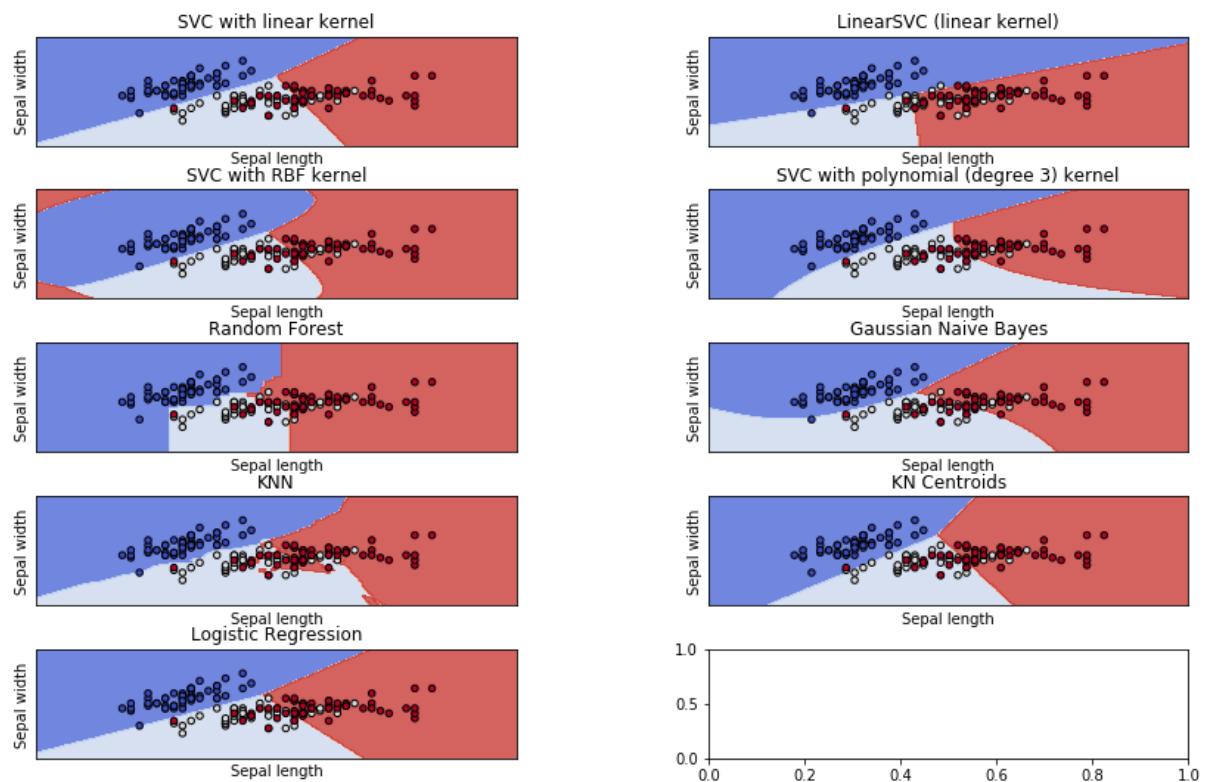
L'apartat B es tractava d'una mena d'inicialització en aquesta pràctica. A partir del dataset de *iris* havíem de fer una comparativa de diferents models de classificació per a observar quin feia la classificació més precisa.

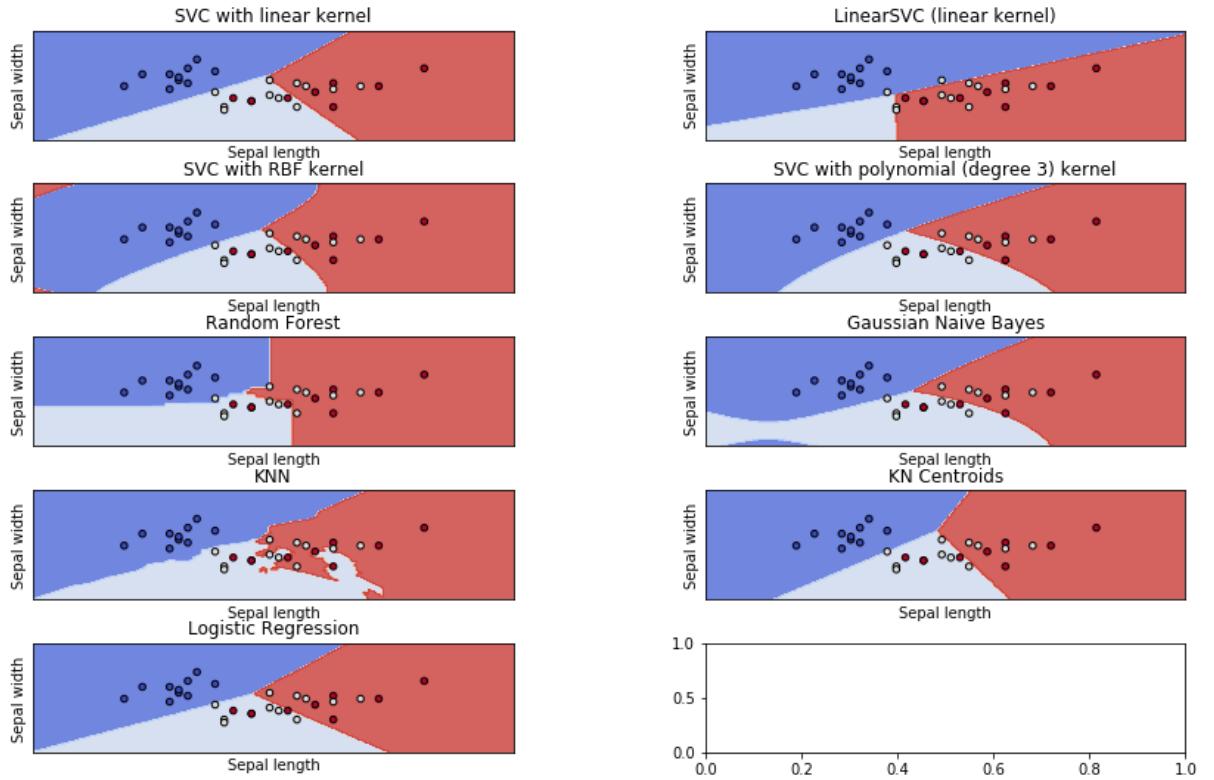
Hem aplicat els models SVC amb kernel rbf, Random Forest, Gaussian Naive Bayes, KNN, KN Centroids i Logistic Regression

Una vegada hem fet la selecció dels models els avaluem tots amb tres particions del dataset possibles, en el nostre cas hem seleccionat particions de 0.5,0.7 i 0.8. Si avaluem tots els nostres models amb cadascuna de les particions, obtenim les millors classificacions amb la partició de 0.5, per tant procedim a dividir el nostre dataset entre train i test amb aquest rang de 0.5-0.5.

Correct classification Logistic	0.5 % of the data:	0.8133333333333334
Correct classification SVM	0.5 % of the data:	0.7866666666666666
Correct classification Random Forest	0.5 % of the data:	0.7866666666666666
Correct classification Gaussian Naive Bayes	0.5 % of the data:	0.8133333333333334
Correct classification KNN	0.5 % of the data:	0.7733333333333333
Correct classification KN Centroids	0.5 % of the data:	0.84
Correct classification Logistic	0.7 % of the data:	0.8222222222222222
Correct classification SVM	0.7 % of the data:	0.7555555555555555
Correct classification Random Forest	0.7 % of the data:	0.8
Correct classification Gaussian Naive Bayes	0.7 % of the data:	0.8222222222222222
Correct classification KNN	0.7 % of the data:	0.7777777777777778
Correct classification KN Centroids	0.7 % of the data:	0.8222222222222222
Correct classification Logistic	0.8 % of the data:	0.7
Correct classification SVM	0.8 % of the data:	0.6666666666666666
Correct classification Random Forest	0.8 % of the data:	0.6333333333333333
Correct classification Gaussian Naive Bayes	0.8 % of the data:	0.7
Correct classification KNN	0.8 % of the data:	0.6666666666666666
Correct classification KN Centroids	0.8 % of the data:	0.7

Agafem com a valor de la X els dos últims atributs. Un cop feta la selecció, passem a avaluar la classificació de cadascun dels models i les seves accuracy. Provarem primer a fer-ho a partir del model amb totes les dades i després farem una classificació només amb les dades d'entrenament.





5. Conclusions

Com a conclusió diríem que el millor model de classificació seria el Random Forest. Veiem que això és força lògic ja que utilitza la tècnica d'ensemble que consisteix bàsicament en fer diferents arbres de decisió (tants com se li indiquin per paràmetre) i acabi fent una mitja dels valors obtinguts per tal de trobar el valor, i de fet trobem que ja en el primer moment d'utilitzar-lo, sense haver de buscar els millor valors per als paràmetres fent servir l'Exhaustive Grid Search ja fa una classificació que arriba a l'accuracy perfecte.

6. Problemes trobats

El problema més gran que ens hem trobat al fer la pràctica ha sigut a l'hora de buscar quin era el millor valor per els hiperparàmetres dels models de regressió. El problema ens venia bàsicament perquè no sabíem ben bé quins paràmetres considerar en alguns casos i sobretot alhora de passar el conjunt sobre el que fer les proves, ja que cada execució trigava molt i moltes vegades havíem de parar l'execució després de mitja hora sense haver obtingut resultats perquè el rang de

valors era massa gros. Al final ho vam poder solucionar executant model a model i anant incrementant el conjunt de valors molt a poc a poc per poder veure fins a quin punt era òptim deixar que l'algoritme arribés al final i trobés la solució o si era millor deixar-ho a mitges.

A part d'això, també vam tenir una mica de problemes a l'hora de triar quina era la millor mètrica de classificació perquè no acabavem de veure clarament la diferència entre elles, de manera que vam haver de destinar força temps a buscar les avantatges i inconvenients de cadascuna per acabar seleccionant quina era la que més ens convenia per el nostre cas.

Líink al repositori de GitHub:

<https://github.com/carlota Castro/APC-Pr-ctica2.git>