Universitat Autònoma de Barcelona

Facultat de Ciències

Pràctica 2

Aprenentatge Computacional

MatCAD

Pablo Ruiz, Clara Sorolla i Maria Pallejà

1565555, 1569191, 1570129

Índex

1 Introducció **3**

2 Apartat B **4**
2.1 Dataset . . . 4
2.2 Models . . . 5
2.3 Resultats . . . 6
2.4 Interpretació dels resultats . . . 7

3 Apartat A **8**
3.1 EDA . . . 8
3.1.1 Atributs . . . 8
3.1.2 Distribució dels atributs . . . 9
3.1.3 Correlació entre atributs . . . 10
3.2 Preprocessing . . . 11
3.2.1 Estudi dels Nans i errors en el conjunt de dades . . . 11
3.2.2 Tractament de les dades categòriques . . . 11
3.2.3 Normalització i escalatge de les dades . . . 11
3.2.4 Datasets escollits per a l'estudi . . . 11
3.2.5 PCA i PolinomyalFeatures . . . 12
3.3 Model selection . . . 13
3.3.1 Testejar models . . . 13
3.3.2 Matrius de confusió . . . 14
3.3.3 Mètodes d'Ensemble . . . 18
3.4 Crossvalidation . . . 19
3.4.1 LeaveOneOut . . . 21
3.5 Metric Analysis . . . 22
3.5.1 ROC i PR Curve . . . 22
3.5.2 Mètrica per optimitzar la classificació . . . 23
3.6 Hyperparameter Search . . . 24
3.6.1 Formes de buscar el millor paràmetre . . . 24
3.6.2 Mètodes més eficients . . . 24
3.6.3 Execució dels 2 mètodes estudiats . . . 24

4 Conclusions **25**

Índex de figures

1 Precisió de les diferents combinacions partició/model. 6
2 Histogrames dels atributs 9
3 *Heatmap* de les correlacions entre atributs. 10
8 Matrius de confusió del `dataset1` 15
13 Matrius de confusió del `dataset2` 17
14 Fit_time dels dos datasets 19
15 Score_time dels dos datasets 20
16 Test_precision_macro dels dos datasets 20
17 Test_recall_macro dels dos datasets 20
18 *Precision-Recall Curve* del `dataset` amb dades desequil·librades 22
19 *ROC Curve* del `dataset` amb dades equil·librades 22

Índex de taules

1 Dades del *dataset* Iris 4
2 Atributs del conjunt de dades inicial 8
3 Quantitat de cada categoria de l'atribut objectiu 9
4 Atributs que tenen més correlació amb l'atribut objectiu 10
5 Quantitat de Nans en el *dataset* inicial 11
6 Precisions dels diferents models aplicats als dos datasets 13
7 Resultats dels models provats amb el `Dataset1`. 15
8 Resultats dels models provats amb el `Dataset2`. 17
9 Resultats del *Cross-validation* fent servir el primer `Dataset`. 19
10 Resultats del *Cross-validation* fent servir el segon `Dataset`. 19
11 Resultats del LeaveOneOut fent servir els dos datasets. 21
12 Classification report 23
13 Resultats dels mètodes estudiats per buscar els millors hiperparametres 24

1 Introducció

La classificació és una tasca fonamental dins del camp de l'aprenentatge computacional que consisteix en saber dir a quina classe pertany una mostra a partir dels seus atributs.

L'objectiu d'aquesta pràctica és aplicar diversos classificadors, avaluar correctament l'error dels models i visualitzar-ne les dades i el rendiment.

En l'apartat B aplicarem diversos models a un conjunt de dades senzill. Ens servirà per experimentar i provar noves eines d'avaluació, optimització i visualització.

En l'apartat A aplicarem aquestes mateixes tècniques en un cas real. Classificarem les mostres d'un conjunt de dades[1] sobre *Pokemons*. Cada mostra és un *Pokemon* amb diversos atributs i cada mostra pot pertànyer a una de les següents classes:

- Comú
- Llegendari

Ens trobem davant d'un problema de classificació binària. A més, com veurem més endavant la classe comú està més representada que la llegendària, és a dir, hi ha una distribució desigual. Aquest és un problema anomenat *Imbalanced binary classification*. La llibreria *sklearn* compta amb moltes eines per abordar aquest problema, per això en farem un ús extensiu.

[1] https://www.kaggle.com/rounakbanik/pokemon

2 Apartat B

Aquesta part de la pràctica ens permet experimentar amb un conjunt de dades senzill abans d'endinsar-nos en un problema major. És més útil i entenedor provar els classificador amb poques dades i simples.

2.1 Dataset

La llibreria *sklearn* compta amb alguns conjunts de dades elementals per provar tota la maquinaria implementada en la seva llibreria. Hem escollit el famós *dataset* Iris que compta amb mostres de les diferents varietats de la flor iris. Cada mostra té els següents atributs:

- Sepal length (cm)
- Sepal width (cm)
- Petal length (cm)
- Petal width (cm)

i pot pertànyer a una de les següents classes:

- Setosa
- Versicolor
- Virginica

amb identificadors 0, 1, i 2, respectivament.

Mostrem una part del conjunt de dades per fer-nos una idea:

sepal length	**sepal width**	**petal length**	**petal width**	**class**
5.2	3.5	1.5	0.2	0
6.1	2.8	4.7	1.2	1
5.7	2.5	5.0	2.0	2
6.3	2.5	4.9	1.5	1
6.7	3.1	4.7	1.5	1

Taula 1: Dades del *dataset* Iris

Veiem que totes les dades són de tipus numèric i decimal. Això permetria implementar de manera senzilla una regressió clàssica.

Quant a la classificació, si sols considerem tres dels quatre atributs, podem imaginar-nos cada mostra com un punt en l'espai, en $\mathbb{R}^3$. Doncs, podem deduir que les mostres d'una mateixa classe estaran agrupades en regions petites de l'espai. Aquesta idea ens fa pensar en l'algorisme del KNN o en plans que separin les diferents classes (Màquines de vectors de suport).

2.2 Models

La llibreria *sklearn* compta amb molts algorismes de classificació. Hem provat tots els que coneixem i algun altre més. Els models que hem provat sobre el `dataset` Iris són:

- Regressió logística
- Màquines de vectors de suport
  - Lineal
  - Polinòmic
  - Funció de base radial (rbf)
- Descens del gradient
- KNN
- Arbre de decisió
- Gaussian Naive Bayes

Com que les màquines de vectors de suport poden fer servir diferents tipus de *kernels*[2], diferenciem diversos models d'aquest tipus.

Quan entrenem un model li passem el parell (X, y). Però aquestes dades no són tot el conjunt de dades original, sinó una partició aleatòria anomenada conjunt d'entrenament. Ens reservem unes quantes mostres per validar posteriorment el rendiment del model, ja que si el validéssim amb les mateixes dades ens estaríem aprofitant de el *over-fitting* i ens estaríem enganyant.

Ara bé, sí que podem decidir quina proporció de mostres assignem a cada conjunt. Aquesta proporció és un hyper-paràmetre molt important que afecta directament a la classificació.

Entrenem cadascun dels models mencionats amb tres particions diferents:

- 50% d'entrenament
- 70% d'entrenament
- 80% d'entrenament

Per tant, al final, tenim diverses combinacions partició/model que podem avaluar.

[2]La classe `LinearSVC` i la classe `SVC` amb *kernel* lineal són similars quant al concepte però són diferents quant a la implementació.

2.3 Resultats

El mètode `score()` de les classes dels models de *sklearn* retorna la precisió de classificar la X i la y del conjunt de dades de test. La formula de la precisió és:

$$\frac{\text{true positive}}{\text{true positive} + \text{false positive}}.$$

De cada combinació partició/model obtenim una les següents precisions:

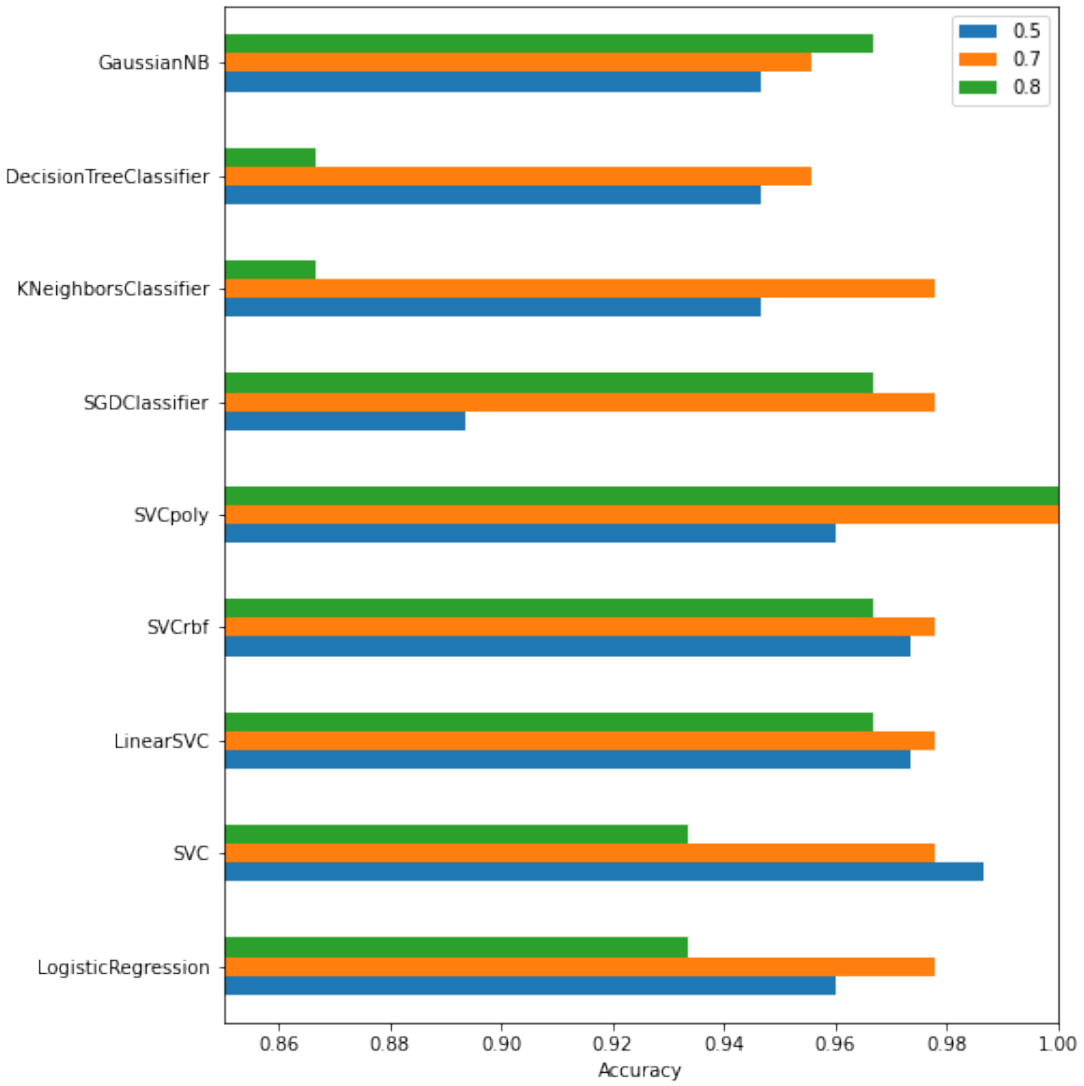


Figura 1: Precisió de les diferents combinacions partició/model.

2.4 Interpretació dels resultats

L'eix *Accuracy* del gràfic de barres horitzontal va des de 0.85 fins a 1. Per tant, de primeres podem veure que tots els classificadors amb totes les particions tenen un bon rendiment. Ara bé, en aquesta petita escala $[0.85, 1]$ hi ha diferencies significatives entre classificadors.

Veiem que els conjunts de dades amb un 70% de dades d'entrenament acostumen a resultar en una millor precisió. Podem deduir que aquesta proporció d'entrenament i test és la que condueix a un equilibri quant al *under-fitting* i *over-fitting*.

En la partició del 50% es pot produir *under-fitting* perquè la quantitat de dades d'entrenament no és prou com per modelitzar la complexitat de les dades de test. En la partició del 80%, encara hi ser de les més comuns, pot sobre-entrenar el model i afegir una complexitat excessiva.

Destaca el baix rendiment de l'arbre de decisió i el KNN amb el 80% d'entrenament, així com el perfecte rendiment de la maquina de vectors de suport amb *kernel* polinòmic amb el 70% i 80% d'entrenament.

En general, les màquines de vectors de suport lideren els resultats, ja que són estables per a qualsevol partició i tenen un rendiment significatiu en totes les seves versions.

3 Apartat A

3.1 EDA

3.1.1 Atributs

Atribut	Tipus
Abilities	object
Against_bug	float64
Against_dark	float64
Against_dragon	float64
Against_electric	float64
Against_fairy	float64
Against_fight	float64
Against_fire	float64
Against_flying	float64
Against_ghost	float64
Against_grass	float64
Against_ground	float64
Against_ice	float64
Against_normal	float64
Against_poison	float64
Against_psychic	float64
Against_rock	float64
Against_steel	float64
Against_water	float64
Attack	int64
Base_egg_steps	int64
Base_happiness	int64
Base_total	int64
Capture_rate	object
Classification	object
Defense	int64
Experience_growth	int64
Height_m	float64
Hp	int64
Japanese_name	object
Name	object
Percentage_male	float64
Pokedex_number	int64
Sp_attack	int64
Sp_defense	int64
Speed	int64
Type1	object
Type2	object
Weight_kg	float64
Generation	int64
Is_legendary	int64

Taula 2: Atributs del conjunt de dades inicial

En la taula anterior es mostren tots els atributs que té el nostre conjunt de dades. Podem observar que hi ha 41 atributs diferents que inclouen les diverses característiques d'un *Pokemon*. La taula mostra el nom de l'atribut i el tipus de dada de la qual es tracta. Veiem que tenim atributs de tipus categòric, numèric i binari.

3.1.2 Distribució dels atributs

L'atribut objectiu és *Is_legendary* ja que el propòsit és determinar si un *Pokemon* és llegendari o no. Es tracta d'un atribut binari que indica 1 en cas de ser-ho i 0 en cas contrari. És important tenir en compte que les etiquetes no estan balancejades ja que tenim molt més *Pokemons* no llegendaris que llegendaris. Això ens portarà complicacions més endavant. En la següent taula es mostra la quantitat que tenim de cada categoria.

Is_legendary	**Quantitat**
0	731
1	70

Taula 3: Quantitat de cada categoria de l'atribut objectiu

En la figura 2 visualitzem els histogrames dels atributs per veure la distribució de les seves dades i veure si estan balancejades. Observem que hi ha diversos atributs amb dades no balancejades, com *Is_legendary*, els atributs de *against*, *base_happiness*... També podem observar quins atributs sembla que segueixen una distribució gaussiana (*speed, attack, sp_defense*...).

Figura 2: Histogrames dels atributs

3.1.3 Correlació entre atributs

A continuació es mostra una taula amb els atributs que tenen una correlació més alta amb l'atribut objectiu *Is_legendary*.

Atribut	Correlació
Base_egg_steps	0.873488
Base_total	0.485440
Base_happiness	0.413108
Sp_attack	0.406281
Weight_kg	0.393023
Experince_growth	0.361038
Sp_defense	0.343241
Height_m	0.322155
Speed	0.311639
Hp	0.308405
Attack	0.303295
Defense	0.265587

Taula 4: Atributs que tenen més correlació amb l'atribut objectiu

En la figura 3 es mostra la correlació entre totes les parelles d'atributs. Observem que hi ha molt poques parelles amb alta correlació.

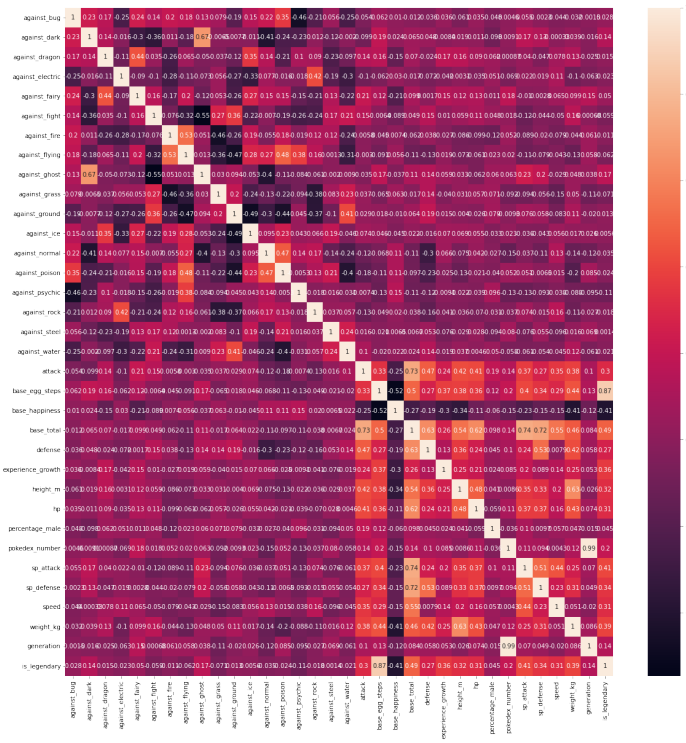


Figura 3: *Heatmap* de les correlacions entre atributs.

3.2 Preprocessing

3.2.1 Estudi dels Nans i errors en el conjunt de dades

El primer que vam fer per començar el següent pas del nostre estudi va ser estudiar la quantitat de `Nans` que teníem al nostre *dataset*. Vam trobar el següent:

Atribut	Tipus	Quantitat de Nans	% Nans
height_m	float64	20	2.497
percentatge_male	float64	98	12.23
type2	object	384	85.39
weight_kg	float64	20	2.497

Taula 5: Quantitat de Nans en el *dataset* inicial

L'atribut que té un percentatge molt elevat de `Nans` en el nostre *dataset* inicial és *type2*, un atribut de text. En aquesta secció no el tractarem ja que farem modificacions més endavant al aplicar tècniques per codificar dades categòriques. En els altres atributs que contenen valors numèrics controlarem els valors de `Nan` utilitzant la mitjana, una tècnica que ja hem fet servir en anteriors projectes i que ens ha donat bons resultats.

Un error que hem trobat al nostre *dataset* mentre l'hem estudiat ha sigut que a l'atribut *capture_rate* hi havia un valor d'una de les files que era de tipus `object` el qual no tenia sentit. Després de buscar per Internet informació sobre el *Pokemon* en qüestió li hem donat el valor de 70, ja que aquest era el seu valor real de *capture_rate*.

3.2.2 Tractament de les dades categòriques

Un cop tractats els `Nans` i arreglats els errors del `dataset` ens hem fixat en les dades categòriques i com podríem tractar-les. Els atributs amb dades de text són: *classification, type1, type2, name, japanese_name i abilities*. Hem vist tres casos diferents:

El primer, l'atribut *abilities* consisteix en una llista de `strings` que descriu les habilitats de cada *Pokemon*. Per tal de poder treballar amb aquestes dades hem modificat manualment el `dataset` creant una columna nova per a cada abilitat diferent que hi ha en el *dataset* i li hem donat un valor de 1 o 0 depenent de si el *Pokemon* en qüestió la té o no. En conseqüència, el nombre d'atributs del nostre `dataset` ha crescut de manera considerada.

El segon cas que hem trobat ha sigut en els atributs *classification, type1, type2* que són de tipus text i per tal de poder-los tractar hem fet servir la funció `get_dummies` de la llibreria `pandas`, la qual ens permet crear columnes per a cada un dels valors únics d'aquests atributs i donar-li un valor binàri a cada fila depenent de si el *Pokemon* en qüestió el té o no, molt similar al cas anterior i per tant també ens va fer créixer el número d'atributs.

Finalment, hem decidit descartar els atributs *name i japanese_name* del nostre estudi ja que no ens aportaven informació necessària per poder predir el nostre atribut objectiu.

3.2.3 Normalització i escalatge de les dades

Un cop hem tingut totes les dades amb valors numèrics hem estudiat si les dades dels atributs estaven normalitzades per poder decidir si cal o no normalitzar-les en cas de que no ho estiguessin. Un cop estudiades, hem observat que no ho estaven però com que tenen escales molt variades i són atributs categòrics hem decidit escalar-les utilitzant la tècnica `MinMaxScaler`.

3.2.4 Datasets escollits per a l'estudi

Finalment, un cop acabat el *preprocessing* ens ha quedat un `dataset` amb moltes columnes, és a dir, molts atributs. Per poder fer un millor estudi hem decidit filtrar aquests atributs respecte

la correlació que tenien amb el nostre atribut objectiu *is_legendary*. Hem eliminat aquells que ens han donat un valor menor a un llindar de |0.1|. A més, hem creat un segon dataset on hem decidit equil·librar les dades de *is_legendary*, agafant una mostra aleatòria de dades negatives de la mateixa longitud de dades que teniem positives, per tal d'aconseguir un *dataset* amb dades balancejades. Per tant, d'aqui en endavant s'han estudiat els següents datasets:

- **Dataset 1**: *Dataset* original amb els canvis realitzats en les dades categòriques i sense els atributs que tenen un llindar menor a |0.1|.
- **Dataset 2**: Com el Dataset1 però amb dades de *is_legendary* balancejades.

Estudiar aquests dos datasets ens permet veure si és millor tenir dades balancejades o no per tal de trobar un bon model per predir el nostre atribut objectiu.

3.2.5 PCA i PolinomyalFeatures

Hem decidit no aplicar un `PCA` degut a que la majoria dels nostres atributs són categòrics. A més, també hem decidit no aplicar `PolinomyalFeatures` degut a que tenim masses atributs i sabem que el nombre d'atributs de la matriu de sortida s'escala polinomialment en el nombre de característiques de la matriu d'entrada i exponencialment en el grau.

3.3 Model selection

3.3.1 Testejar models

La llibreria *sklearn* compta amb molts algorismes diferents de classificació. Hem provat tots els que coneixem i algun altre més que hem pensat que ens podria donar bons resultats. Els models que hem provat són:

- Regressió logística
- Maquines de vectors de suport
  - Lineal
  - Polinòmic
  - Funció de base radial (rbf)
- Descens del gradient
- KNN
- Arbre de decisió
- Gaussian Naive Bayes

Com que les màquines de vectors de suport poden fer servir diferent *kernels*[3], diferenciem diversos models d'aquest tipus.

Hem entrenat tots els models tant amb ambdós *datasets*. De cada combinació model/*dataset* obtenim les següents precisions.

	Imbalanced	**Balanced**
LogisticRegression	0.983	0.979
SVC	0.990	0.975
LinearSVC	0.986	0.979
SVCrbf	0.990	0.996
SVCpoly	0.899	0.439
SGDC	0.964	0.975
KNN	0.904	0.643
DecisionTree	0.993	0.943
GaussianNB	0.920	0.771

Taula 6: Precisions dels diferents models aplicats als dos datasets

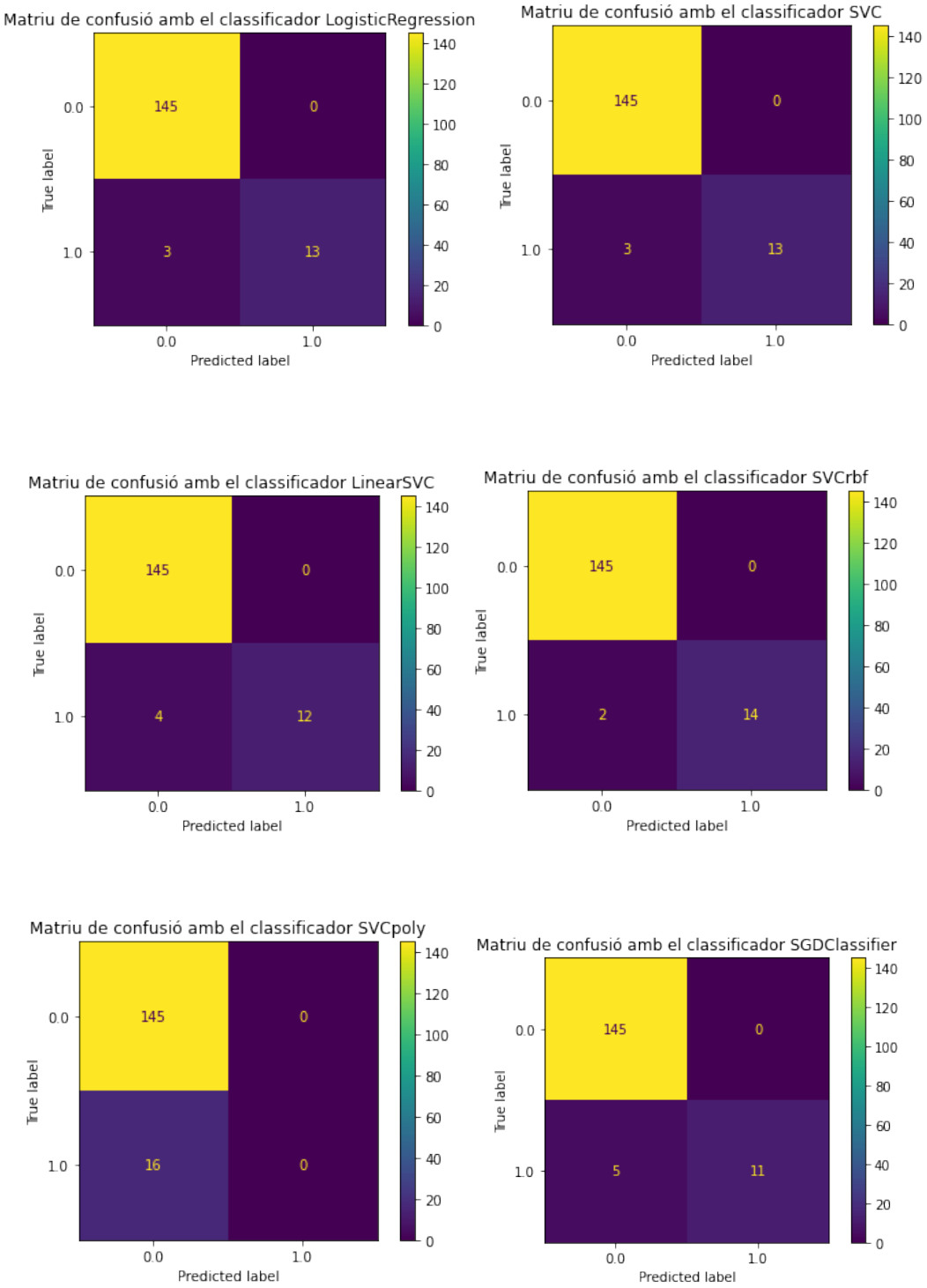
En la taula anterior observem que totes les combinacions model-*imbalanced* tenen una molt bona precisió. Ara bé, quan mirem la columna *balanced* observem quins models s'estan aprofitant de la mètrica de la precisió juntament amb la classe objectiu desbalancejada. Més endavant veurem una millor mètrica per a aquest conjunt de dades. Doncs, els models SVM amb *kernel* polinòmic, el *KNN* i el *Gaussian Naive Bayes* no responen satisfactòriament a aquest problema de classificació.

[3]La classe `LinearSVC` i la classe `SVC` amb *kernel* lineal són similars quant al concepte però són diferents quant a la implementació.

3.3.2 Matrius de confusió

Per tal de poder decidir quin és el model amb millors resultats ens hem fixat en les matrius de confusió que hem obtingut amb cada un dels `datasets`:

- **Dataset 1:**



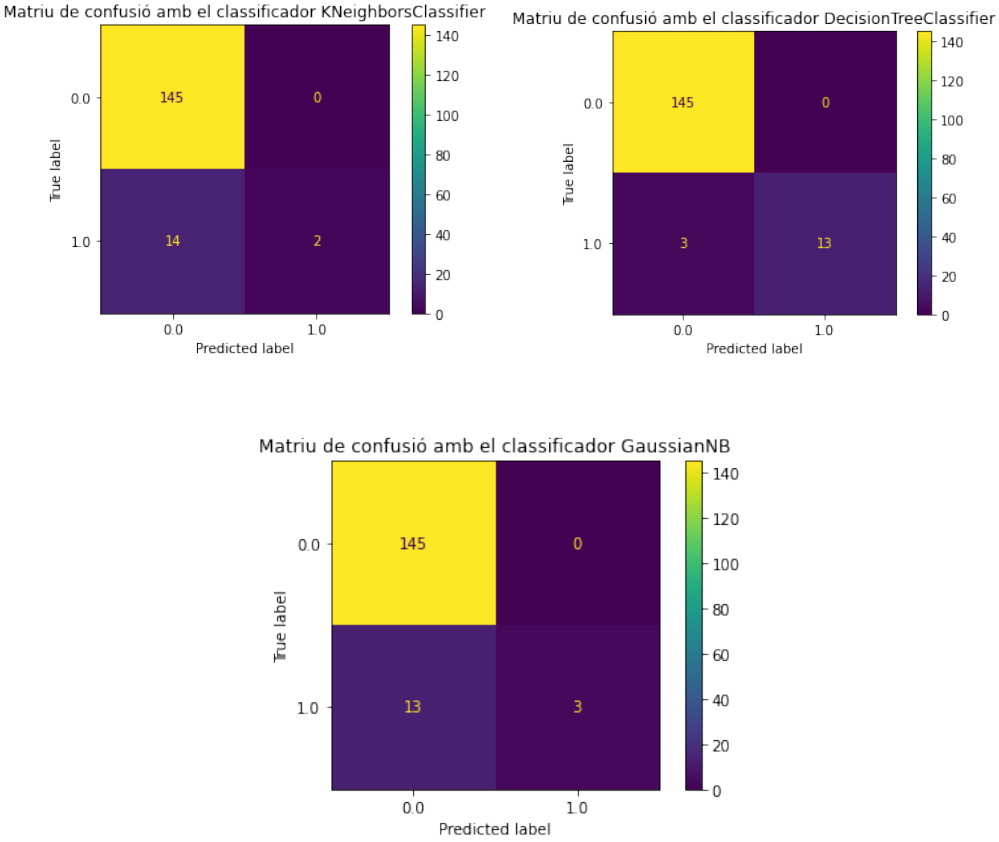


Figura 8: Matrius de confusió del `dataset1`

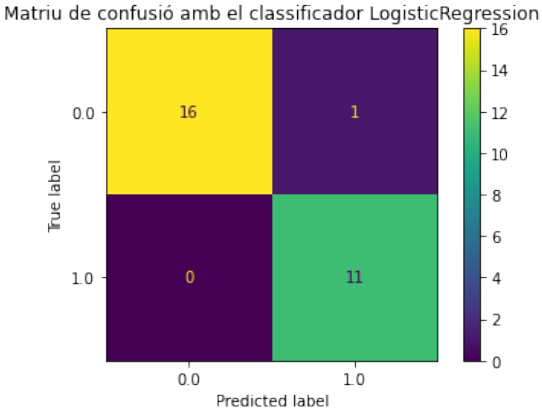
A partir de les matrius hem pogut extreure la informació de la següent taula:

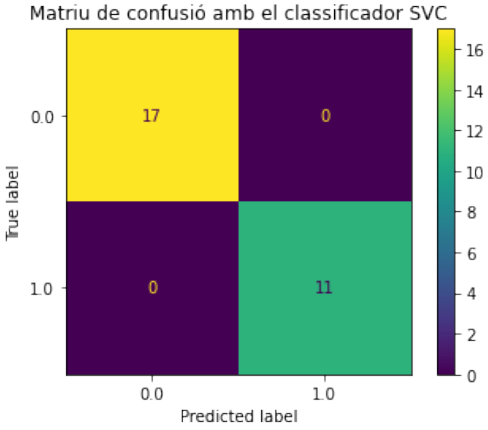
Model	T-negatives	F-positives	F-negatives	T-positives
`Logistic Regression`	145	0	3	13
`SVC: Linear kernel`	145	0	3	13
`LinearSVC`	145	0	4	12
`SVC: rbf kernel`	145	0	2	14
`SVC: poly kernel`	145	0	16	0
`SGDC Classifier`	145	0	5	11
`KNeighbors Classifier`	145	0	14	2
`Decision Tree Classifier`	145	0	3	13
`GaussianNB`	145	0	13	3

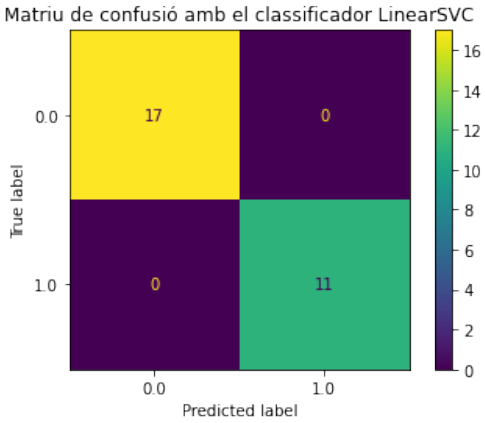
Taula 7: Resultats dels models provats amb el `Dataset1`.

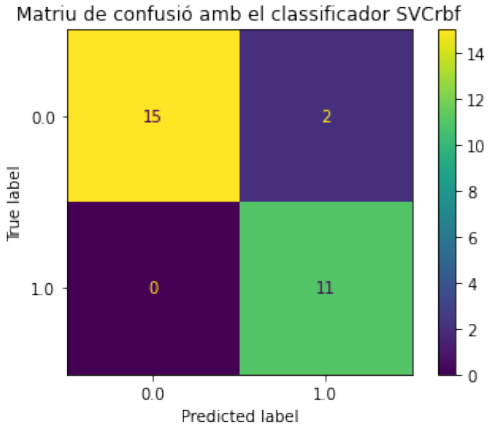
Podem veure clarament com el model amb millors resultats és el `SVC` amb kernel `rbf`. També ens podem fixar en com tots els nostres models classifiquen bé la classe negativa, en el nostre cas que no sigui llegendari el *Pokemon* i en canvi, la classe positiva (que sigui llegendari) la classifica pitjor. Això passa degut a la distribució de les dades del nostre atribut objectiu: No tenim equil·libri entre les dues classes, hi ha una clara majoria de la classe negativa i això es reflexa en els nostres resultats.

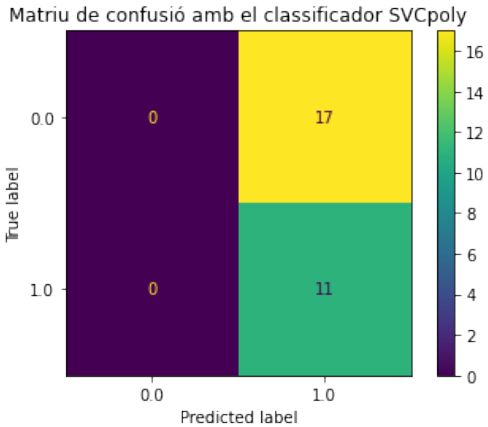
- **Dataset 2:**

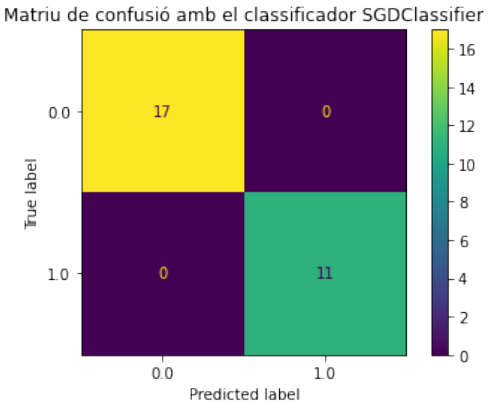

Matriu de confusió amb el classificador LogisticRegression
16
1
0
11
0.0
1.0
True label
Predicted label


Matriu de confusió amb el classificador SVC
17
0
0
11
0.0
1.0
True label
Predicted label


Matriu de confusió amb el classificador LinearSVC
17
0
0
11
0.0
1.0
True label
Predicted label


Matriu de confusió amb el classificador SVCrbf
15
2
0
11
0.0
1.0
True label
Predicted label


Matriu de confusió amb el classificador SVCpoly
0
17
0
11
0.0
1.0
True label
Predicted label


Matriu de confusió amb el classificador SGDClassifier
17
0
0
11
0.0
1.0
True label
Predicted label

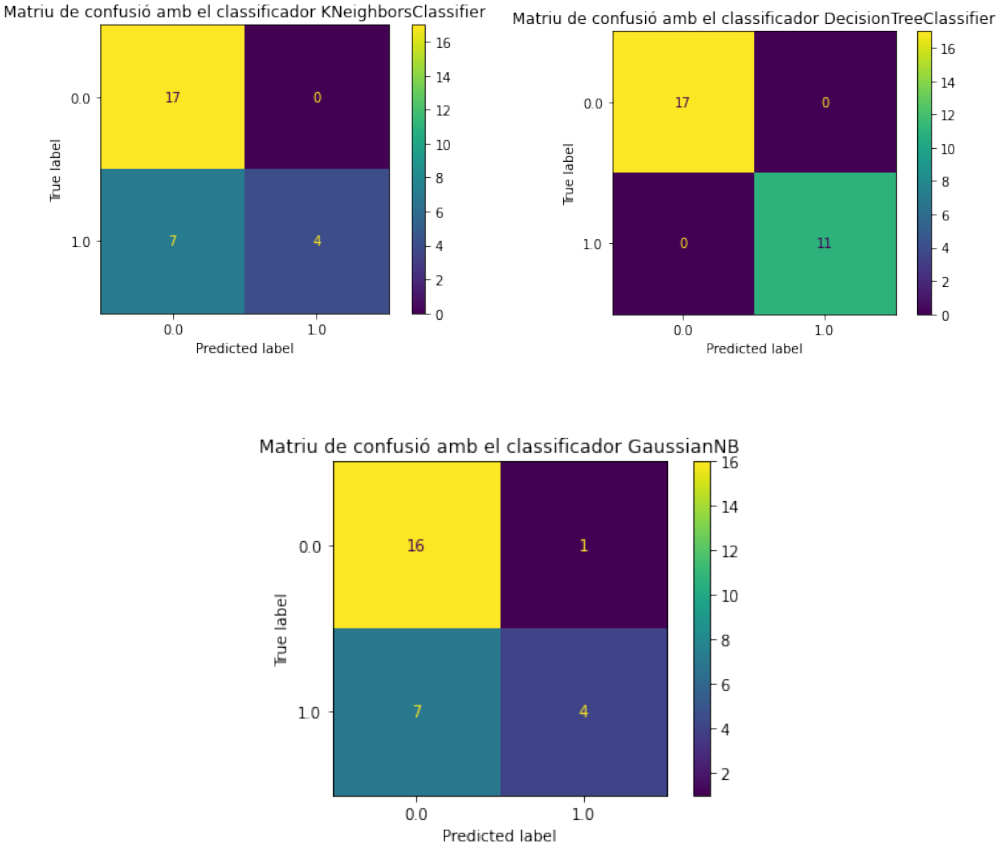


Figura 13: Matrius de confusió del `dataset2`

A partir de les matrius hem pogut extreure la informació de la següent taula:

Model	T-negatives	F-positives	F-negatives	T-positives
`Logistic Regression`	16	1	0	11
`SVC: Linear kernel`	17	0	0	11
`LinearSVC`	17	0	0	11
`SVC: rbf kernel`	15	2	0	11
`SVC: poly kernel`	0	17	0	11
`SGDC Classifier`	17	0	0	11
`KNeighbors Classifier`	17	0	7	4
`Decision Tree Classifier`	17	0	0	11
`GaussianNB`	16	1	7	4

Taula 8: Resultats dels models provats amb el `Dataset2`.

Veient els resultats podem concloure que en aquest cas els millors models serien el *SVC* amb kernel `Linear`, el *LinearSVC* o el *DecisionTree Classifier*. En aquest cas tenim menys mostres a classificar però no es veu una diferencia evident entre la classificació de les dades negatives i positives ja que en el `dataset2` la distribució de l'atribut objectiu està equil·librada.

3.3.3 Mètodes d'Ensemble

En el nostre estudi hem decidit provar diferents mètodes d'*Ensemble*, aquests ens permeten crear diferents models amb el mateix algorisme d'aprenentatge i combinar-los per a crear-ne un de millor. Per fer-ho hem utilitzar el primer `dataset`, ja que tot i no tenir les dades de l'atribut objectiu distribuïdes equitativament és més gran que no pas el segon `dataset` que hem creat per equil·librar-les. Hem fet proves amb tres mètodes diferents.

El primer és el *Random Forest*, un model que crea classificadors d'arbres de decisió en diverses submostres del conjunt de dades i utilitza la mitjana per millorar la precisió de la nostra predicció i controlar l'`overfitting`. Aquest ens ha donat uns resultats molt bons, arribant a obtenir una precisió del 100% diverses vegades que l'hem executat.

El segon mètode que hem utilitzat ha sigut el `Bagging` que crea models independents als altres de forma paral·lela i després els combina. Per fer-ho modifica el conjunt d'aprenentatge seleccionant aleatòriament elements d'aquest conjunt i repartint diferents dades per a cada classificador que crea. Per a decidir com es fa la classificació del model conjunt es mira quina classe genera cada model i es consideren com a vot per a aquella classe. La classe més votada serà la que retornarà el model conjunt. Aquest mètode també ens ha donat molt bons resultats, la precisió al executar-ho ha variat en un rang entre el 95-100%.

Finalment, l'últim mètode que hem provat és el *Boosting*, que consisteix en crear models de forma seqüencial de manera que els models són dependents entre ells i finalment es combinen. Per tal de crear cada model el que es fa és mirar com es comporten els models creats anteriorment i donar més importància les classes que classifiquen malament; és a dir, es concentra en reduir el biaix. Puja doncs els pesos dels patrons que estaven mal classificats i baixa els que estaven ben classificats. Els classificadors són agregats per votació ponderada, és a dir la classe més votada per tots els models serà la que retornarà el classificador conjunt. Els resultats amb aquest mètode també han resultat ser bons, la seva precisió ha variat en un rang del 96-100%.

3.4 Crossvalidation

Cross-validar els resultats és important per evitar que una partició concreta del `dataset` aporti un *bias* a les mètriques de validació. Com més dades d'entrenament tinguem, més fiables seràn els nostres resultats. Per fer la cross-validació hem utilitzat el *Stratified method*. Els plecs del `dataset` es fan conservant el percentatge de mostres per a cada classe. Els valors de la `k` faràn que canvi el número de plecs que es fa en el nostre conjunt de dades. Com que no tenim tantes dades hem de posar un número petit per tal de que no falli. Hem escollit un valor de `k`: 5.

Després de fer *cross-validació* amb diferents models utilitzant els dos `datasets` que tenim hem utilitzat els *scores* següents com a estratègia per avaluar els resultats: `precision` i `recall`. A més, per tal de poder comparar no només els seus *scores* sinó també el seu rendiment, hem avaluat el temps d'entrenament.

Model	Precision	Recall
Logistic Regression	0.965167330879203	0.8550880626223092
SVC: Linear kernel	0.9634910972916396	0.8901174168297455
LinearSVC	0.9680524213175022	0.8908023483365948
SVC: rbf kernel	0.9643204018305198	0.9465753424657535
SVC: poly kernel	0.456304347826087	0.5
KNeighbors Classifier	0.5568782471971561	0.5071428571428571
SGDC Classifier	0.9686414170969193	0.8129158512720156
Decision Tree Classifier	0.8907826067489932	0.8953033268101762
GaussianNB	0.760373054078039	0.55

Taula 9: Resultats del *Cross-validation* fent servir el primer `Dataset`.

Model	Precision	Recall
Logistic Regression	0.9723809523809525	0.9714285714285715
SVC: Linear kernel	0.9473809523809524	0.9428571428571428
LinearSVC	0.9598809523809525	0.9571428571428573
SVC: rbf kernel	0.9664743589743591	0.9642857142857142
SVC: poly kernel	0.7902576489533011	0.6357142857142857
KNeighbors Classifier	0.778025641025641	0.5999999999999999
SGDC Classifier	0.9301839826839826	0.9071428571428571
Decision Tree Classifier	0.9598076923076924	0.9571428571428571
GaussianNB	0.6059494872446818	0.5714285714285714

Taula 10: Resultats del *Cross-validation* fent servir el segon `Dataset`.

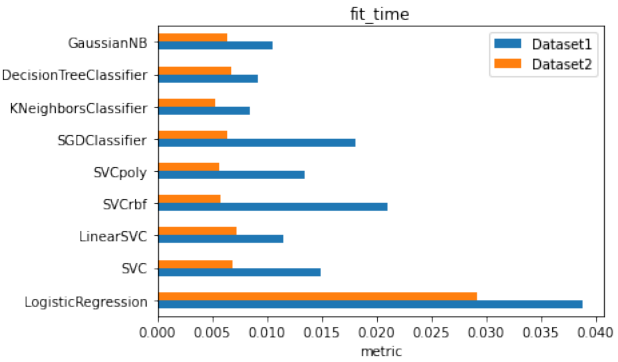


Figura 14: Fit_time dels dos datasets

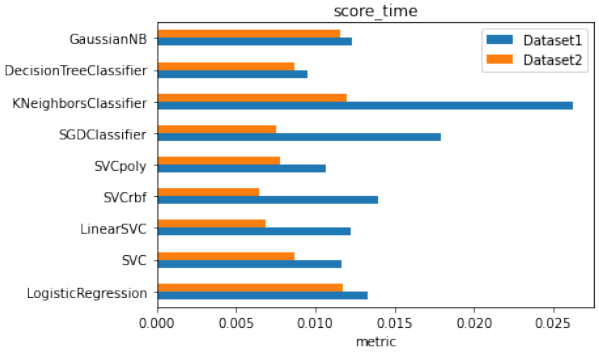


Figura 15: Score_time dels dos datasets

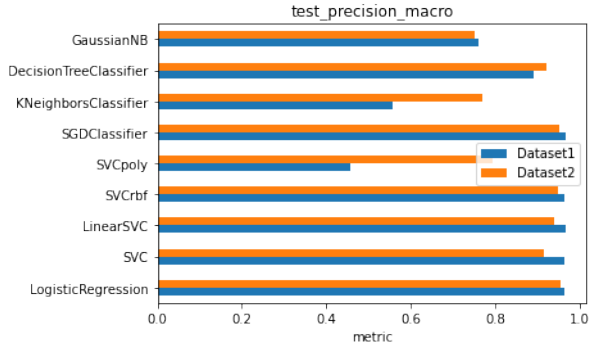


Figura 16: Test_precision_macro dels dos datasets

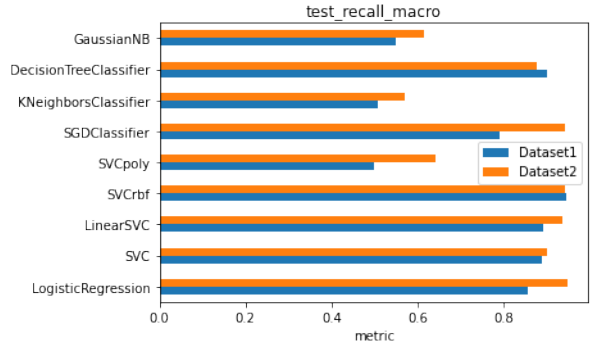


Figura 17: Test_recall_macro dels dos datasets

Veient els resultats obtinguts, en el cas del primer `dataset` podem concloure que el model que ens dona millors resultats com ja haviem vist abans és *SVC* amb un kernel `rbf`, el seu temps d'entrenament però és més alt que altres models que hem vist. Per altra banda, el segon `dataset` ens dona millors resultats amb el model *Logistic* però també ens trobem amb un temps d'entrenament molt alt, comparat amb els altres models.

3.4.1 LeaveOneOut

El *LeaveOneOut* consisteix en utilitzar només una mostra com a test mentre les altres mostres es fan servir per entrenar el model. Al fer servir una única observació per calcular l'error, aquest variarà molt segons quina observació es faci servir. Per evitar-ho, el procés s'ha de repetir tants cops com observacions tenim i a cada iteració deixem fora una iteració diferent, ajustant el model amb la resta i calculant l'error. Al final farem la mitja de tots els errors.
Aquest procés ens permet reduir la variabilitat quan dividim aleatòriament les observacions en dos grups. Per tant, els resultats obtinguts en principi seran més exactes però té un cost computacional molt més alt.

Com sabem que té un cost computacional molt alt, hem descartat alguns models estudiats anteriorment que hem vist que donaven pitjors resultats, per així poder reduir el temps d'execució. A continuació es mostra una taula amb els resultats obtinguts aplicant `LeaveOneOut` als dos datasets estudiats.

Model	**Dataset 1**	**Dataset 2**
`Logistic Regression`	0.986267	0.971429
`SVC: Linear kernel`	0.992509	0.957143
`LinearSVC`	0.991261	0.964286
`SVC: rbf kernel`	0.990012	0.964286
`KNeighbors Classifier`	0.920099	0.6
`SGDC Classifier`	0.966292	0.971429
`Decision Tree Classifier`	0.995006	0.964286
`GaussianNB`	0.932584	0.6

Taula 11: Resultats del LeaveOneOut fent servir els dos datasets.

En la taula anterior observem que pel primer `dataset` tots els models donen molt bons resultats. En el segon `dataset` segueixen donant resultats similars als que hem obtingut abans sense fer servir *LeaveOneOut*, inclús alguns donen pitjors. Per tant, podríem concloure que no surt a compte fer servir el *LeaveOneOut* ja que els resultats sense fer-lo servir eren molt similars.

3.5 Metric Analysis

3.5.1 ROC i PR Curve

La *Precision-Recall Curve* i la *ROC Curve* són dues eines que ens permeten saber com seràn les probabilitats de que surti cada una de les opcions amb un model que intenta predir un atribut binàri. Depenent de com sigui la nostra base de dades amb la que entrenem el nostre model per predir la classe utilitzarem una o l'altra.

En primer lloc, hem utilitzat el nostre primer `dataset` on hi ha les dades de l'atribut objectiu *is_legendary* distribuïdes de forma desequil·librada. Per a aquests casos és millor utilitzar *Precision-Recall Curve* i així ho hem fet utilitzant el model *SVC* amb kernel `rbf`.

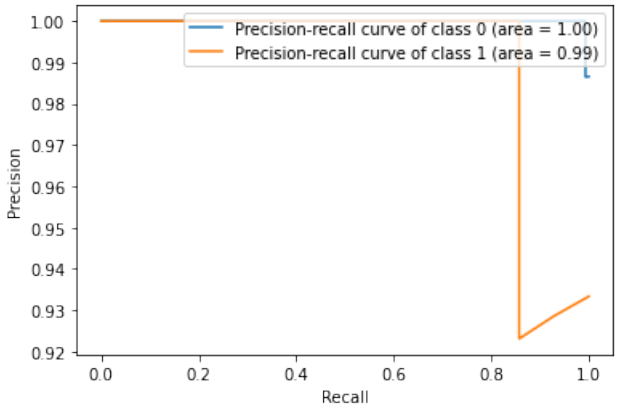


Figura 18: *Precision-Recall Curve* del `dataset` amb dades desequil·librades

Com podem veure i coincidint amb els resultats que hem obtingut en estudiar els models, la classe negativa la classifica sense errors i en canvi, la classe positiva tot i tenir una àrea molt alta, fa errors. Aquests resultats tenen sentit tenint en compte que el nostre `dataset` té una distribució de les dades desigual.

En segon lloc, hem utilitzat el segon `dataset` on hi havia les dades de l'atribut objectiu distribuïdes de manera equitativa. Per a aquests casos és millor utilitzar el *ROC Curve* i ho hem fet utilitzant el model *SVC* amb un kernel `Linear`, ja que en el nostre estudi dels models ens ha donat bons resultats.

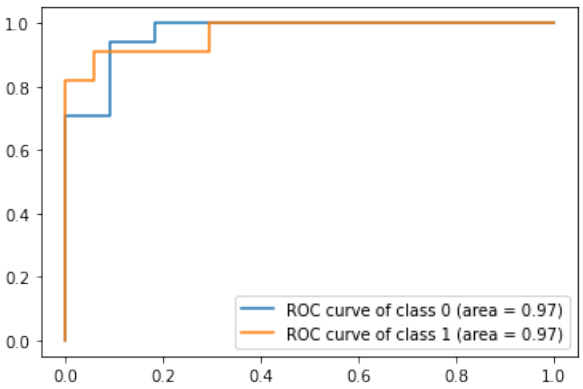


Figura 19: *ROC Curve* del `dataset` amb dades equil·librades

Els resultats també tenen sentit amb el que hem pogut observar al estudiar els diferents models. Les àrees d'ambdues classes, tant negativa com positiva, són molt grans això ens diu que el model tindrà pocs errors en la classificació.

3.5.2 Mètrica per optimitzar la classificació

Usant la funció `classification_report()` de la llibreria *sklearn* podem obtenir, de manera ràpida i breu, un resum de les mètriques de la classificació.

Hem entrenat un regressor logístic i hem usat aquesta funció que rep la y del conjunt de dades de test i la y predita pel classificador.

	common	**legendary**	**macro avg**	**weighted avg**
precision	0.979592	0.928571	0.954082	0.974521
recall	0.993103	0.812500	0.902802	0.975155
f1-score	0.986301	0.866667	0.926484	0.974412
support	145	16	161	161

Taula 12: Classification report

Veiem que la mètrica `precision` és la que dona millor resultat quant a la classificació de llegendaris. Ara bé, no podem guiar-nos per aquesta mètrica perquè les classes no estan balancejades. Hi ha molts més exemples de *pokemons* comuns que de llegendaris.

Doncs, ens queda el `recall` i el `f1-score`. Sens dubte, el `recall` ens servirà per avaluar com de bé classifica el model, però usarem el `f1-score` per comparar models. El `f1-score` és la mitjana harmònica del `precision` i el `recall`, per això permet la comparativa entre classificadors en igualtat de condicions.

3.6 Hyperparameter Search

3.6.1 Formes de buscar el millor paràmetre

En seccions anteriors hem fet *cross-validació* durant l'entrenement per tal de conèixer quin serà el resultat esperat del nostre model fent servir dades mai vistes abans. Haver-ho fet també ens permet optimitzar quins són els hiperparàmetres dels models que millor funcionaran en tests que es realitzaran en un futur. Hem trobat dues formes de buscar el millor parametre. Aquestes són les següents:

- **GridSearchCV**: Considera totes les combinacions possibles que hi ha entre els paràmetres.
- **RandomizedSearchCV**: Mostreja un nombre determinat de candidats d'un espai de paràmetres amb una distribució especificada.

GridSearchCV pot garantir que els paràmetres més precisos es trobin dins del rang de paramàtres especificat, però aquest també pot arribar a ser un defecte degut a que requereix recòrrer totes les combinacions de paràmetres possibles. Fa una cerca exhaustiva. Quan tenim gran quantitat de conjunt de dades i múltiples paràmetres, aquest triga molt, és la seva desavantatge principal. Per aquesta raó, aquest mètode és adequat per tres o quatre hiperparàmetres, però quan aquest creix, la complexitat computacional creix de manera exponencial i tindrà un cost computacional massa alt. En aquest cas, és quan es més recomanable fer servir la cerca aleatòria (*Randomized-SearchCV*).
Si dispossesim de recursos limitats (un PC durant 1 hora), el mètode de *RandomizedSearchCV* ens permetria obtenir millor resultats ja que prova una mostra aleatòria dels paràmetres i és molt més ràpid. En canvi, el *GreadSearchCV* al provar totes les combinacions possibles i anar tant lent, és possible que no arribi a provar les millors combinacions i ens retorni pitjors resultats.

3.6.2 Mètodes més eficients

Existeixen models que poden ajustar dades per a un rang de valors d'algun paràmetre gairebé tan eficientment com ajustar l'estimador per a un valor únic del paràmetre. Aquesta característica es pot aprofitar per dur a terme una validació creuada més eficient utilitzada per a la selecció del model d'aquest paràmetre.

El paràmetre més comú susceptible d'aquesta estratègia és el paràmetre que codifica la força del regularitzador. En aquest cas diem que calculem el camí de regularització de l'estimador.

3.6.3 Execució dels 2 mètodes estudiats

Per tal de provar els diferents mètodes, hem estudiat el model *SVC* en el `dataset` amb les dades de l'atribut objectiu distribuïdes de forma no equitativa ja que hem vist anteriorment que és un del que ens donava millors resultats. A més, li hem passat els diferents kernels, valors de gamma i valors de C que volíem estudiar per tal de que cadascun dels mètodes ens retornés el que aquest creia que era la millor combinació. Els resultats obtinguts han estat els següents:

Mètode	**Kernel**	**Gamma**	**C**	**Temps/training**	**T. execució**
GridCV	rbf	0.2857	0.3127	0.0109	40 min
RandomizedCV	rbf	0.2653	0.4136	0.0117	2 min

Taula 13: Resultats dels mètodes estudiats per buscar els millors hiperparametres

En la taula anterior podem observar que els resultats obtinguts són bastant similars amb una única diferència, el temps d'execució total. Així, demostrem el que s'ha mencionat abans, que el mètode *GridSearchCV* és molt més costos computacionalment. Observem que el temps que triguen per training és bastant similar, per tant per arribar a una combinació de paràmetres bastant similar, el primer mètode ha necessitat molt més temps i no sortiria a compte encara que sigui molt bo.

4 Conclusions

D'entre totes les aplicacions de l'aprenentatge computacional, la classificació és una de les més aplicades avui en dia a la vida real. Per exemple, diferenciar entre un tumor maligne i un de benigne, evitar els correus de *spam* o classificar els diferents objectes que rep la càmera d'un cotxe autònom. Els avantatges que poden suposar aquesta capacitat de classificar pot ser diferencial per a una empresa.

En aquesta pràctica hem pogut experimentar amb les eines de classificació més elementals, així com amb les diverses maneres d'avaluar el rendiment dels classificadors i visualitzar el seu comportament.

En l'apartat B hem provat aquestes eines en un conjunt de dades senzill com és l'Iris. Aquesta simplicitat ens permet entendre de manera profunda el funcionament dels instruments algorísmics més fonamentals.

En l'apartat A ens hem ocupat d'un conjunt de dades amb més complexitat. Amb 41 atributs diferents per cada mostra. Les mostres són *Pokemons* i hem tractat de classificar-los en comuns i llegendaris. Per tant, hem treballat dins de l'àmbit de la classificació binària.

En l'exploració inicial de les dades hem vist que predominen els atributs categòrics. Els atributs *against* en concret, presentaven una correlació amb l'atribut objectiu menor que la resta. És comprensible ja que aquest conjunt d'atributs aglutina un tipus d'informació concreta en molta quantitat d'atributs, per tant, cadascun d'ells no pot rivalitzar quant a discriminació amb altres atributs com, per exemple, l'atribut *speed*.

Encara que era deduïble, les mostres no estan balancejades. Hi ha 731 mostres de *Pokemon* comuns contra només 70 de llegendaris. Aquest fet el tenim en compte durant la pràctica.

Un cop hem donat un primer cop d'ull, hem arreglat les deficiències del conjunt de dades original per posar-lo a punt per als classificadors. D'una banda, els atributs *height_m*, *percentatge_male*, *type2* i *weight_kg* presentaven dades sense valor les quals hem optat per omplir-les amb la mitja. D'altra banda, com hem dit abans, hi ha molts atributs categòrics els quals hem canviat usant la funció `get_dummies()`. Encara que aquest mètode estén la quantitat d'atributs significativament, ens ha semblat que havíem de mantenir la informació discriminant que aportaven aquests atributs.

Després, hem entrenat tots els models de la llibreria *sklearn* que coneixem i alguns més. Hem provat diversos *kernels* per les màquines de vectors de suport i hem provat mètodes d'*Ensemble* (*Random Forest*, *boosting* i *bagging*) que han donat uns resultats molt bons.

A continuació, hem fet un dels passos més importants: la *cross*-validació. Hem fet servir tant el *dataset* balancejat com l'original. Els resultats han estat reveladors perquè el conjunt de dades balancejat s'ha imposat per sobre de l'altre.

Com veiem, l'equilibri en la quantitat de mostres comunes i llegendàries és determinant a l'hora de classificar el conjunt de dades, però també ho és a l'hora d'avaluar-lo amb una mètrica concreta. Ens hem adonat ràpidament que l'*accuracy* no és viable davant el cas no balancejat. En aquest cas, el *recall* i el *f1-score* han estat la solució.

Per acabar, hem tractat d'optimitzar els hyper-paràmetres com si anéssim a enviar el model a producció. L'estratègia *GridSearch* (mirar totes les combinacions possibles d'hyper-paràmetres) ens ha sorprès, ja que hem vist que escollint els paràmetres adients es pot potenciar molt el model. Ara bé, aquesta tasca requereix un esforç computacional significatiu.

Creiem que la recerca dels millors hyper-paràmetres l'hauríem d'haver fet just abans del *cross-validation*.