Pràctica 2: Classificació

Joel Guevara Lopez, 1564581
Jordi Morales Casas, 1564921
Oriol Benítez Bravo, 1566931

Aprenentatge Computacional, III MatCAD

Desembre 2021

Índex

1 Models de classificació aplicats al *dataset* IRIS

La nostra base de dades consisteix en un conjunt d'atributs que ens donen les característiques d'una flor i ens diu a què espècie de flor pertany. Les variables són *Sepal Length*, *Sepal Width*, *Petal Length* i *Petal Width* i les especies a classificar seran *Setosa*, *Versicolour* i *Virginica*.

En aquesta pràctica el que farem serà utilitzar diferents tipus de models de classificació per poder trobar una predicció que ens decideixi l'espècie de la flor a partir de les seves característiques.

1.1 Anàlisis de les dades inicial

Primerament, serà necessari observar les correlacions inicials de les variables que tenim, per poder predir quines variables seran les més dependents de l'espècie de flor.

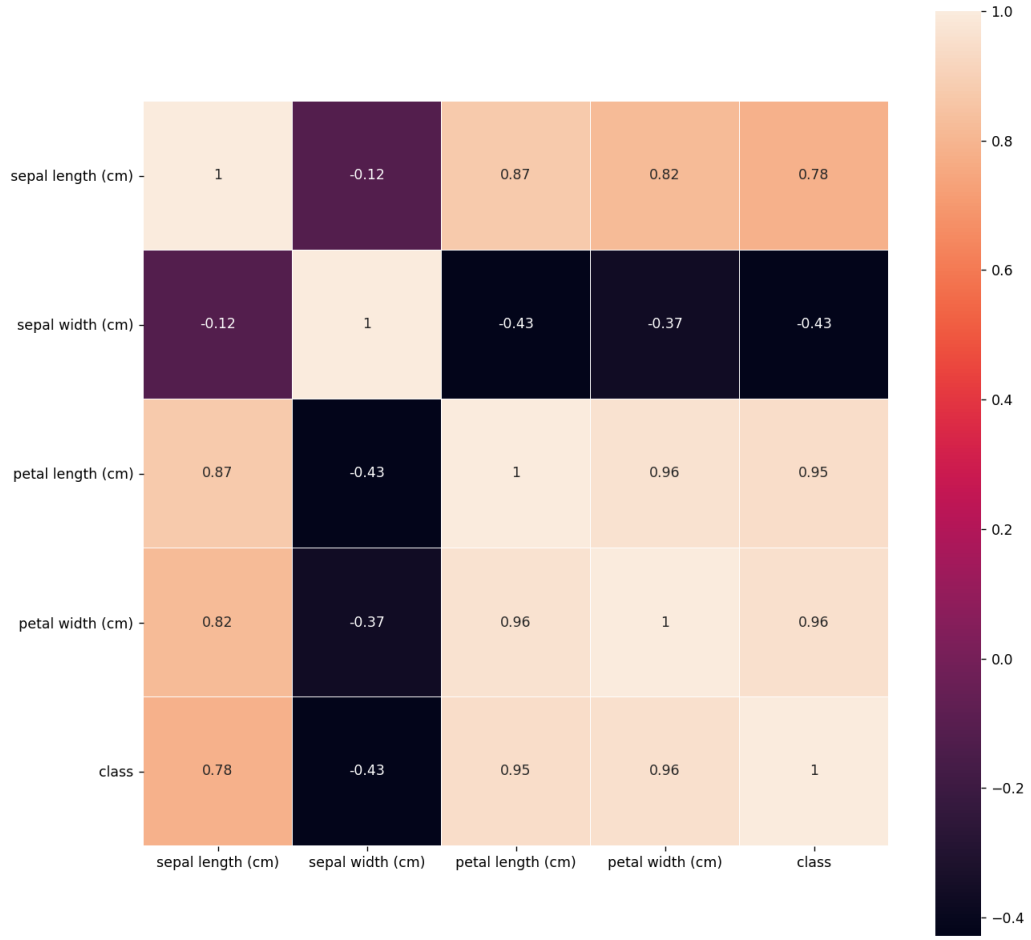


Figura 1: Correlacions dels atributs amb la variable a classificar del dataset Iris.

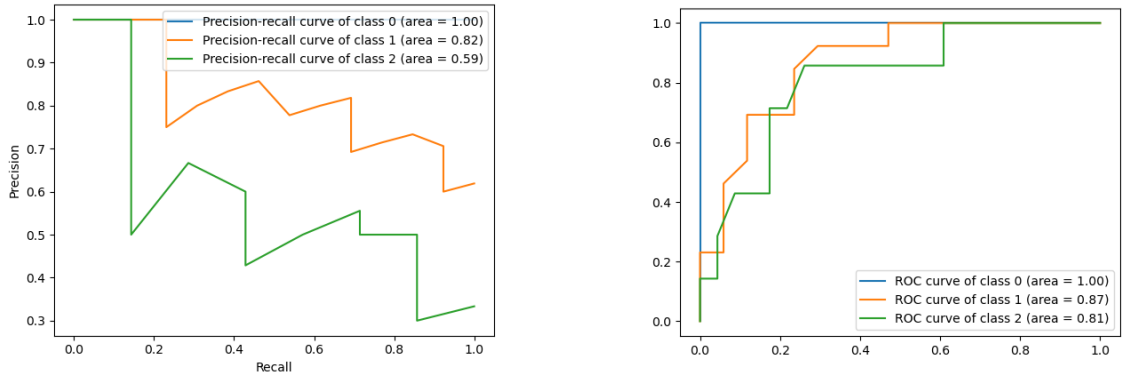
Observem que les variables *petal length* i *petal width* tenen una alta correlació amb la classe, per tant, s'utilitzaran només aquests dos atributs per fer la classificació. Això ho fem per tal de simplificar els càlculs com perquè es pot predir que les altres variables no aportaran informació.

1.2 Models de classificació

Per cada model hem generat les corbes *PR* i *ROC* per poder avaluar el rendiment la classificació de les diferents classes. La *PR* mostra la precisió, entesa com la fracció d'instàncies recuperades que són rellevants, contra el *recall*, que és la fracció d'instàncies rellevants que han sigut recuperades. La *ROC* és una representació gràfica de la sensibilitat en enfront de l'especificitat d'un sistema de classificació binari.

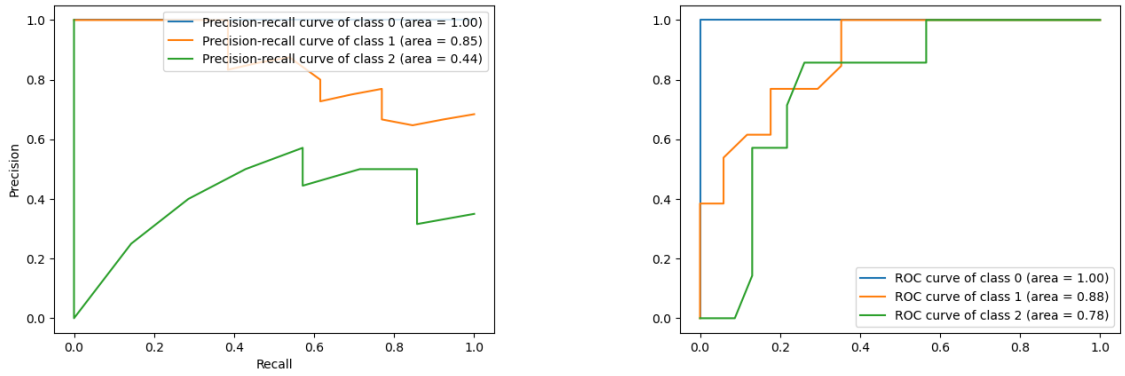
A continuació es pot observar els resultats:

- **Regressió logística:**



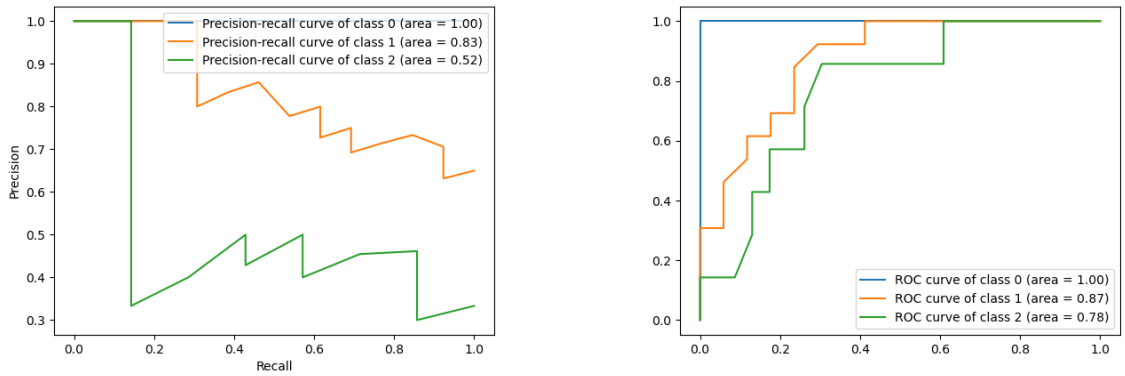
Regressió logística: Corba PR (dreta) i ROC (esquerra)

- **SVM amb kernel RBF:**



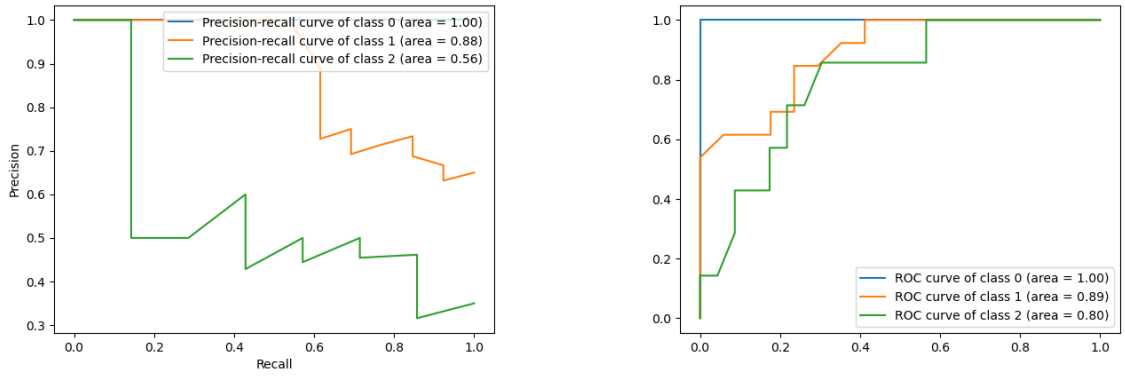
SVM amb kernel RBF: Corba PR (dreta) i ROC (esquerra)

- **SVM amb kernel lineal:**



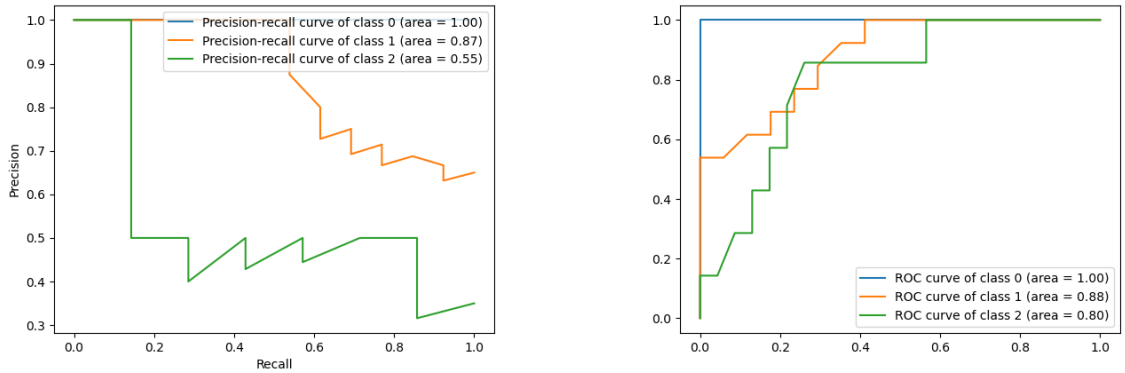
SVM amb kernel lineal: Corba PR (dreta) i ROC (esquerra)

- **SVM amb kernel polinomial de grau 2:**



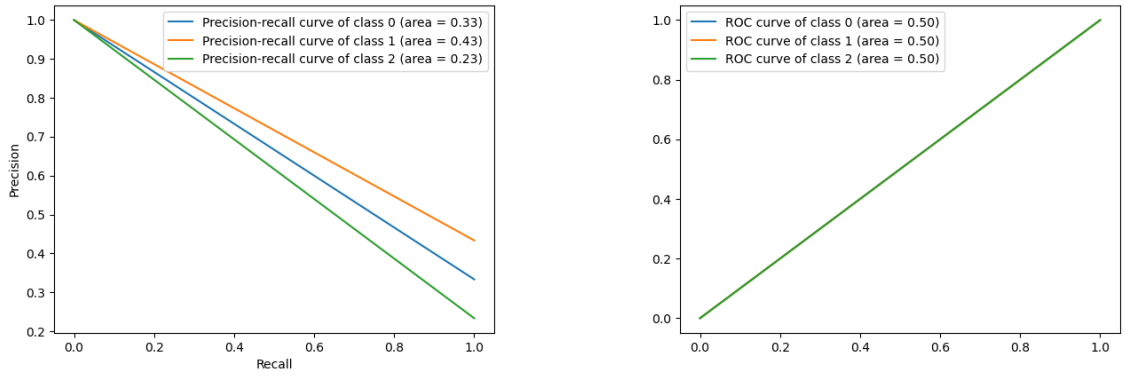
SVM amb kernel polinomial de grau 2: Corba PR (dreta) i ROC (esquerra)

- **SVM amb kernel polinomial de grau 3:**



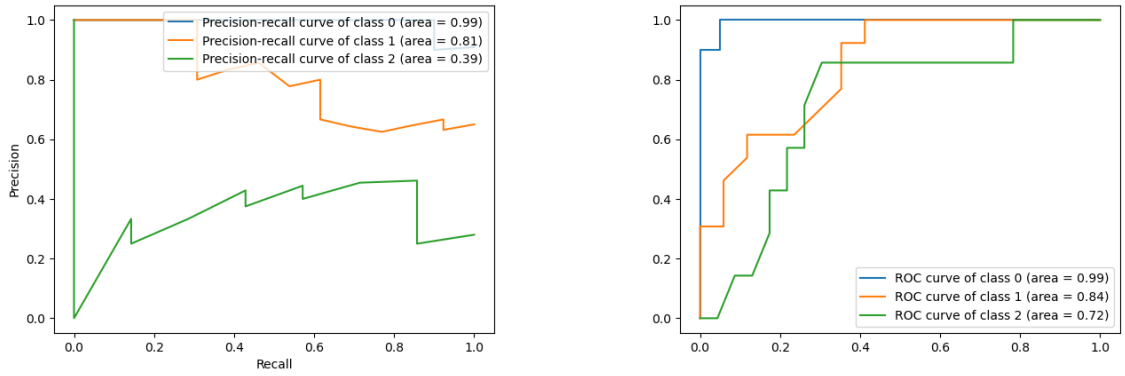
SVM amb kernel polinomial de grau 3: Corba PR (dreta) i ROC (esquerra)

- **SVM amb kernel Sigmoidal:**



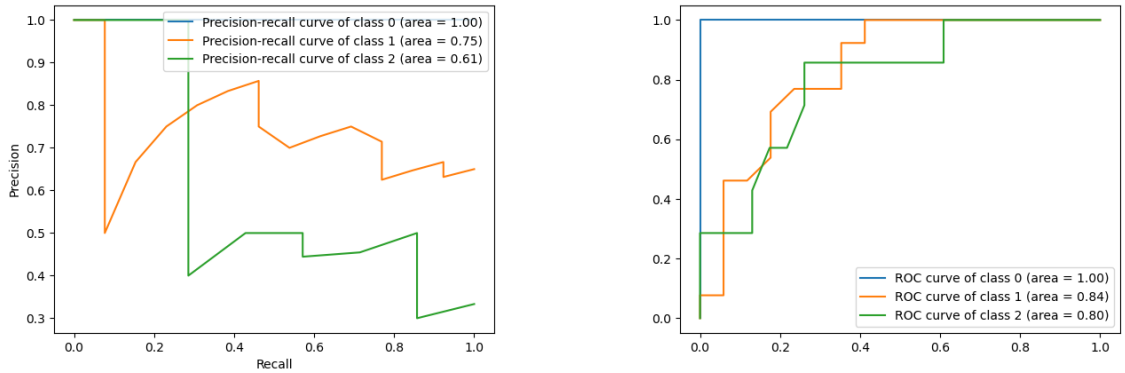
SVM amb kernel Sigmoidal: Corba PR (dreta) i ROC (esquerra)

- **Random Forest amb criteri *GINI*:**



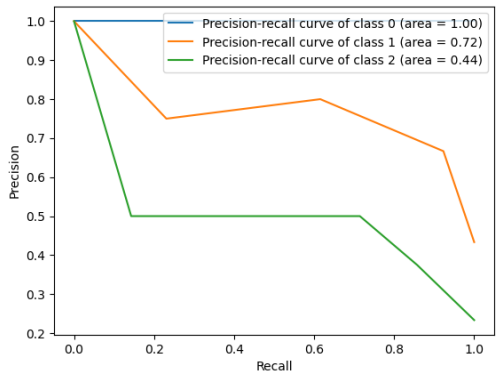
Random Forest amb criteri *GINI*: Corba PR (dreta) i ROC (esquerra)

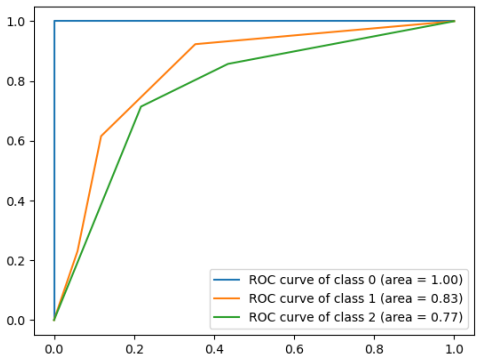
- **Random Forest amb criteri *entropy*:**



Random Forest amb criteri *entropy*: Corba PR (dreta) i ROC (esquerra)

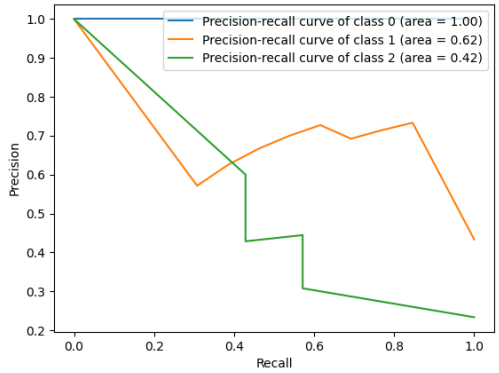
- **KNN amb l'algoritme de *ball tree*:**

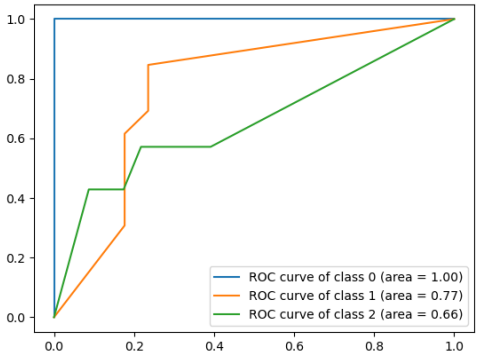




KNN amb l'algoritme de *ball tree*: Corba PR (dreta) i ROC (esquerra)

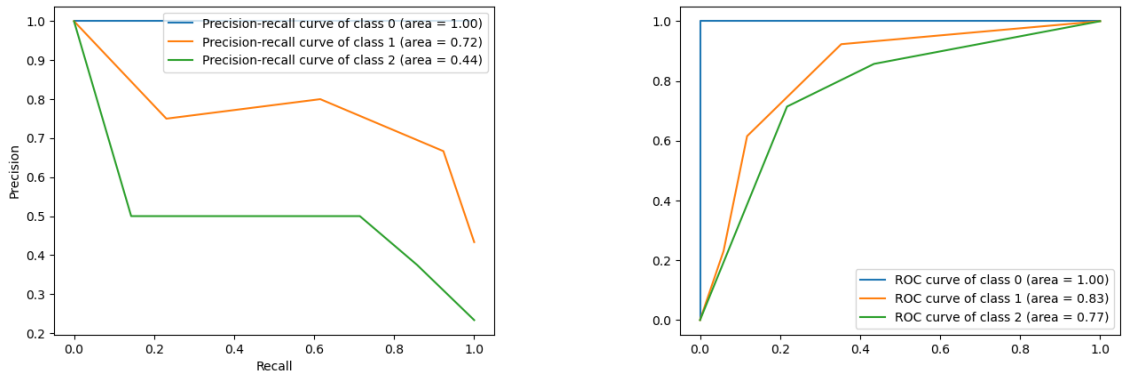
- **KNN amb l'algoritme de *ball tree* amb els pesos ajustats segons la distància:**





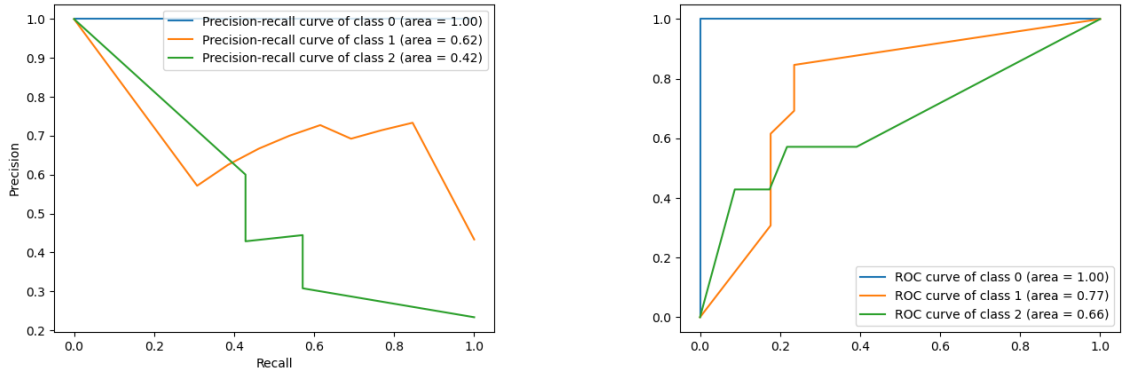
KNN amb l'algoritme de *ball tree* amb els pesos ajustats segons la distància: Corba PR (dreta) i ROC (esquerra)

- **KNN amb l'algoritme de *kd tree*:**



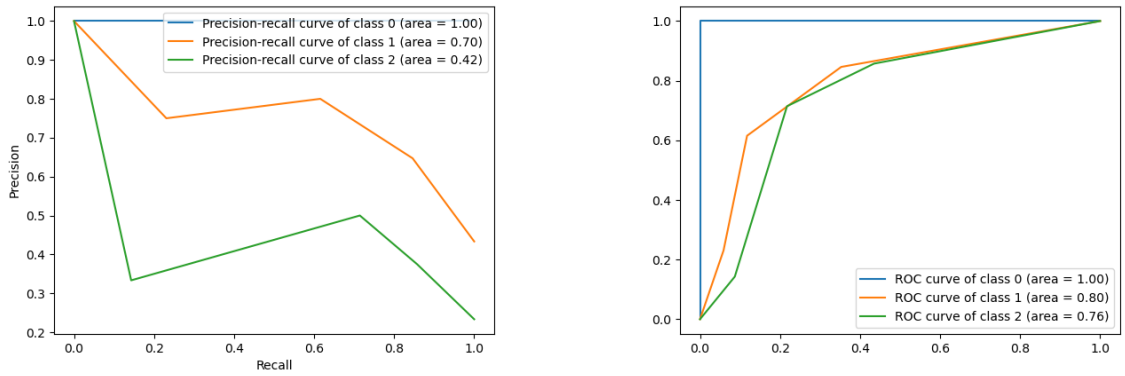
KNN amb l'algoritme de *kd tree*: Corba PR (dreta) i ROC (esquerra)

- **KNN amb l'algoritme de *kd tree* amb els pesos ajustats segons la distància:**



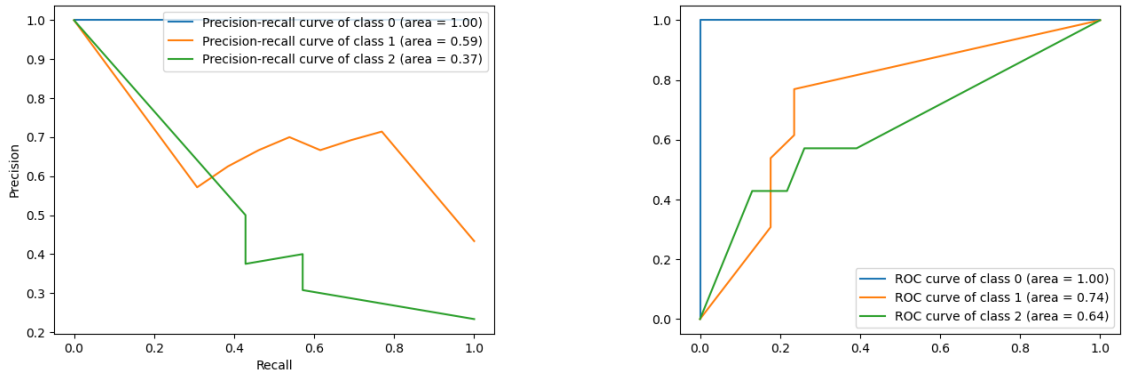
KNN amb l'algoritme de *kd tree* amb els pesos ajustats segons la distància: Corba PR (dreta) i ROC (esquerra)

- **KNN complet (força bruta):**



KNN complet: Corba PR (dreta) i ROC (esquerra)

- **KNN complet amb els pesos ajustats segons la distància:**



KNN complet amb els pesos ajustats segons la distància: Corba PR (dreta) i ROC (esquerra)

Si estudiem classe per classe, veiem com la classe 0 (Setosa) es classifica perfectament (Arriba a (1,1) seguint una línia horitzontal a la gràfica *PR* i a l'àrea màxima a la corba ROC), la qual cosa indica que les seves característiques la diferencien de la resta.

No succeeix el mateix amb la resta de classes: Tant la classe 1 (Versicolor) com la 2 (Virginica) tenen un comportament força similar amb tots els models. Si ens fixem en la corba *PR* observem com no és possible assolir un *recall* alt sense perdre molta precisió. Tot i això, si mirem l'àrea sota la corba ROC, podem concloure que és bastant probable que distingeixi positivament aquestes classes (Les corbes es troben per sobre de la diagonal). D'entre tots podem destacar el model logístic, l'*SVM* amb *kernel RBF* i el *Random Forest* amb criteri *GINI*. Aquests semblen obtenir el millor balanç entre precisió i *recall* i classifiquen correctament les classes de manera consistent.

L'únic cas on no succeeix el descrit anteriorment és quan es fa servir un *SVM* amb *kernel*

Sigmoidal. En un cas ideal les corbes *PR* haurien de tendir a un *recall* d'1 amb el màxim de precisió possible, però el que veiem és el contrari, la qual cosa ens indica clarament que aquestes dades no són res apropiades pel model. També s'ha perdut la propietat que classificava perfectament la classe 0.

Per comprovar aquestes reflexions, es pot observar la precisió d'aquests models en el classificador de les classes.

Métode de selecció	**Precisió**	**Especificacions**
Logistic	0.8	
SVM	0.7666666666666667	kernel='rbf'
SVM	0.7333333333333333	kernel='lineal'
SVM	0.7333333333333333	kernel='poly',degree=2
SVM	0.7333333333333333	kernel='poly',degree=3
SVM	0.2333333333333334	kernel='sigmoid'
Random Forest	0.8	max_depth=2
Random Forest	0.7666666666666667	n_estimators=1000, max_depth=5,criterion="entropy"
KNN	0.6666666666666666	algorithm="ball_tree"
KNN	0.6666666666666666	algorithm="kd_tree"
KNN	0.7333333333333333	algorithm="brute"
KNN	0.6666666666666666	algorithm="ball_tree",weights="distance"
KNN	0.6666666666666666	algorithm="kd_tree",weights="distance"
KNN	0.7	algorithm="brute",weights="distance"

Podem observar que les millors classificacions les fan el model logístic i el *Random Forest* amb *max_depth=2*, tot i això, en alguns casos les diferències són molt petites i depenen de com es distribueixen les dades.

Les que fan una pitjor classificació són el *SVM* amb el *kernel* de la funció sigmoide i el *KNN* amb els algoritmes *ball tree* i *kd tree* tant amb pesos com sense.

2 Classificació numèrica aplicada al *dataset* POKEMON

La nostra base de dades consisteix en un conjunt de dades sobre els 802 Pokemon de les set generacions.

De cada Pokemon tenim el nom, el nom en japonès, el numero en la Pokedex, el percentatge dels Pokemon de l'espècie que són mascles, els dos tipus del Pokemon, la descripció del Pokemon, altura i pes del Pokemon, la ràtio de captura, els passos per obrir l'ou de l'espècie, les habilitats del Pokemon, la quantitat d'experiència que pot arribar a obtenir, la felicitat base del Pokemon, la vida base, l'atac base, la defensa base, l'atac especial base, la defensa especial base, la velocitat base, la generació a la qual pertany i si és llegendari o no.

2.1 Anàlisis i tractament inicial de les dades

El primer pas que hem de donar abans de poder treballar amb les dades és solucionar la presència de valors *NaN* a dues de les columnes del dataset. Per sort una d'aquestes dues (*percentage_male*) hem considerat que no era útil per l'estudi i l'hem tret sencera. L'altra columna, *type2*, sí que era més interessant mantenir-la perquè indica el segon tipus del Pokemon. La raó per la qual molts valors d'aquest atribut estaven buits era degut al fet que alguns Pokemon només tenen un tipus (*type1*, totes les mostres tenen aquest atribut). La solució ha sigut omplir els *type2* buits amb el mateix valor que hi ha a *type1*.

Si ens fixem en les dades categòriques veiem com alguns atributs que fan referència a noms (*name*, *japanese_name* o *abilities*) i que podrien considerar-se que són categories són prescindibles a l'hora de classificar. Altres columnes amb dades categòriques que sí que volem són les de *type1* i *type2*. Per codificar-les fem servir la rutina *LabelEncoder* perquè les etiquetes prenguin valors numèrics enters.

Una altra consideració ha sigut com normalitzar les dades, ja que fer això podria facilitar el procés de classificació. En el nostre cas farem una normalització estàndard sobre tota la base de dades restant la mitjana i dividint entre la desviació estàndard.

2.2 Estudi de les correlacions

A continuació es presenten les correlacions entre les variables dependents i si un Pokemon és llegendari o no:

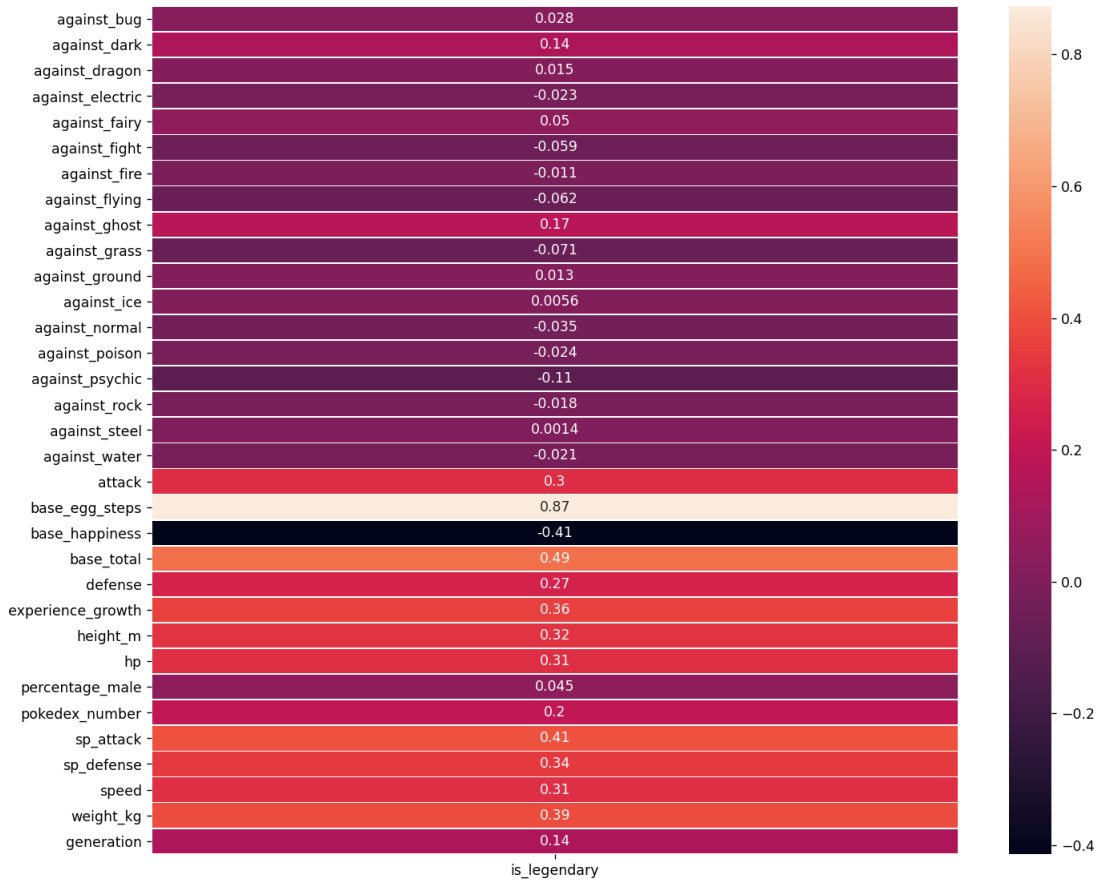


Figura 2: Correlacions dels atributs amb la variable a classificar del dataset Pokemon.

La conclusió més directa que podem treure és que la variable *base_egg_steps* sembla que pot descriure molt bé el comportament de la variable objectiu, ja que té una correlació positiva del 0.87. Altres variables que també tindran un pes important en els models són *attack*, *base_happiness*, *base_total*, *experience_growth*, *height*, *hp*, *sp_attack*, *sp_defense*, *speed* i *weight_kg*.

De les anteriors totes tenen una correlació positiva (afavoreixen que el Pokemon sigui llegendari), excepte *base_happiness*.

2.3 Aplicació de models

Per decidir quin model és el millor per classificar els atributs hem aplicat uns models i hem observat les seves *accuracy*.

Per fer-los hem utilitzat els atributs reduïts que ens ha donat la *Features selection*, que dona com a resultat que s'utilitzan 6 dels atributs de la base de dades inicials. Les features seleccionades seràn *agains_dark, agains_ghost, base_egg_steps, base_total, capture_rate i experience_growth*

En la següent taula es poden observar els resultats.

Mètode de selecció	**Precisió**	**Especificacions**
Logistic	0.9813664596273292	
SVM	0.968944099378882	kernel='rbf'
SVM	0.9813664596273292	kernel='lineal'
SVM	0.9937888198757764	kernel='poly',degree=2
SVM	0.9937888198757764	kernel='poly',degree=3
SVM	0.8571428571428571	kernel='sigmoid'
Random Forest	0.9813664596273292	max_depth=2
Random Forest	1.0	n_estimators=1000, max_depth=5,criterion="entropy"
KNN	0.9751552795031055	algorithm="ball_tree"
KNN	0.9751552795031055	algorithm="kd_tree"
KNN	0.9751552795031055	algorithm="brute"
KNN	0.9813664596273292	algorithm="ball_tree",weights="distance"
KNN	0.9813664596273292	algorithm="kd_tree",weights="distance"
KNN	0.9813664596273292	algorithm="brute",weights="distance"

Observem que el *Random forest* amb el nombre d'estimadors amb 1000, el màxim de profunditat amb 5 i que el criteri de classificació sigui l'entropia ens classifica de forma perfecta el model, tot i que ens trobem moltes opcions amb un 0.99 de *accuracy*, les quals podrien també ser una bona opció per fer la classificació.

En aquest cas hem aplicat *ensamble* en els *random_forest*, ja que ens millorava el classificador, però augmenta el cost de recursos per fer la classificació, podent provocar problemes a l'aplicar-ho a bases de dades molt grans.

Per provar si el *PCA* millorava la classificació hem aplicat els mateixos models sobre les dades modificades i els resultats es poden observar en la següent taula.

Mètode de selecció	**Precisió**	**Especificacions**
Logistic	0.9627329192546584	
SVM	0.968944099378882	kernel='rbf'
SVM	0.9751552795031055	kernel='lineal'
SVM	0.937888198757764	kernel='poly',degree=2
SVM	0.968944099378882	kernel='poly',degree=3
SVM	0.9503105590062112	kernel='sigmoid'
Random Forest	0.937888198757764	max_depth=2
Random Forest	0.968944099378882	n_estimators=1000, max_depth=5,criterion="entropy"
KNN	0.9751552795031055	algorithm="ball_tree"
KNN	0.9751552795031055	algorithm="kd_tree"
KNN	0.9751552795031055	algorithm="brute"
KNN	0.968944099378882	algorithm="ball_tree",weights="distance"
KNN	0.968944099378882	algorithm="kd_tree",weights="distance"
KNN	0.968944099378882	algorithm="brute",weights="distance"

També hem aplicat a els *polinomial features* els diferents models per veure si causen alguna millora.

Taula de dades elevades al quadrat:

Mètode de selecció	**Precisió**	**Especificacions**
Logistic	0.9627329192546584	
SVM	0.9565217391304348	kernel='rbf'
SVM	0.9565217391304348	kernel='lineal'
SVM	0.9503105590062112	kernel='poly',degree=2
SVM	0.9627329192546584	kernel='poly',degree=3
SVM	0.9192546583850931	kernel='sigmoid'
Random Forest	0.9440993788819876	max_depth=2
Random Forest	0.9503105590062112	n_estimators=1000, max_depth=5,criterion="entropy"
KNN	0.9627329192546584	algorithm="ball_tree"
KNN	0.9627329192546584	algorithm="kd_tree"
KNN	0.9627329192546584	algorithm="brute"
KNN	0.9565217391304348	algorithm="ball_tree",weights="distance"
KNN	0.9565217391304348	algorithm="kd_tree",weights="distance"
KNN	0.9565217391304348	algorithm="brute",weights="distance"

Taula de dades amb la combinació lineal de les dades:

Mètode de selecció	**Precisió**	**Especificacions**
Logistic	0.9627329192546584	
SVM	0.9627329192546584	kernel='rbf'
SVM	0.9627329192546584	kernel='lineal'
SVM	0.9627329192546584	kernel='poly',degree=2
SVM	0.9627329192546584	kernel='poly',degree=3
SVM	0.937888198757764	kernel='sigmoid'
Random Forest	0.9503105590062112	max_depth=2
Random Forest	0.968944099378882	n_estimators=1000, max_depth=5,criterion="entropy"
KNN	0.9565217391304348	algorithm="ball_tree"
KNN	0.9565217391304348	algorithm="kd_tree"
KNN	0.9565217391304348	algorithm="brute"
KNN	0.9565217391304348	algorithm="ball_tree",weights="distance"
KNN	0.9565217391304348	algorithm="kd_tree",weights="distance"
KNN	0.9565217391304348	algorithm="brute",weights="distance"

Observem que la millor forma de fer la classificació és sense aplicar cap d'aquestes opcions i treballar amb les dades en brut del *dataset*, ja que són les millors *accuracy* que hem obtingut.

2.4 *Cross-validation*

Per assegurar que un model serà efectiu quan es faci servir en una situació "real"es fan servir tècniques de validació creuada. D'aquesta manera, en teoria, s'aconsegueix que els resultats del model siguin independents de les dades utilitzades en el seu entrenament.

L'estratègia més simple és la de dividir el *dataset* en dos conjunts: Un d'entrenament per generar el model i l'altre de prova, per testejar la seva precisió. En moltes situacions aquesta tècnica és suficient, especialment si disposem de moltes dades.

Si no tenim un gran volum de mostres, l'estratègia *LeaveOneOut* pot ser més adequada. Es fan tants entrenaments com mostres hi ha al *dataset*. Cadascun d'aquests models agafa una única mostra per fer validació i la resta s'utilitzen per entrenar.

K-fold és una altra tècnica de *cross-validation* on es fan k particions de les dades, una d'aquestes es farà servir per validar el model i la resta per entrenar. Al final s'obtenen k models, i el resultat serà la mitjana dels resultats de tots els models. De fet, *LeaveOneOut* és un tipus de *K-fold* on k és igual al nombre de mostres.

Escollim el *K-fold* de *sklearn* amb $k = 50$ perquè dona força bon resultats sobre la nostra base de dades. Si el combinem amb un model Random Forest, obtenim una precisió mitjana

entre els 50 models de 0.995.

Aplicar *LeaveOneOut* amb una regressió logística dona 0.98377 de precisió. Les dimensions del dataset no són tan grans com perquè suposi un problema de temps executar aquest algorisme, però ja hem vist com altres configuracions de models i *cross-validations* donen millors resultats.

2.5 Anàlisis de mètriques

Grafiquem les corbes *PR* i *ROC* sobre les dades en brut, amb els models de *KNN* (amb *weights* = *distance*) i el *random forest* (amb *criterion* = *entropy* i 100 estimadors amb 5 de màxima profunditat). El primer representa un cas on el model es perfecte, mentre que l'altre comet errors en alguns casos.

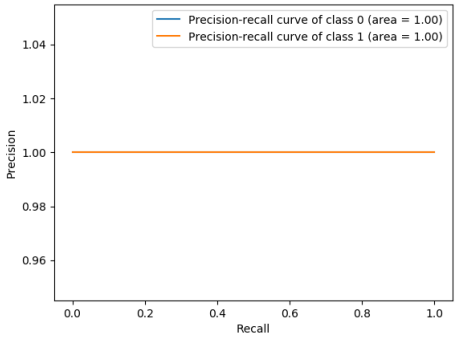


Figura 3: Corbes de PR peel *Random Forest*

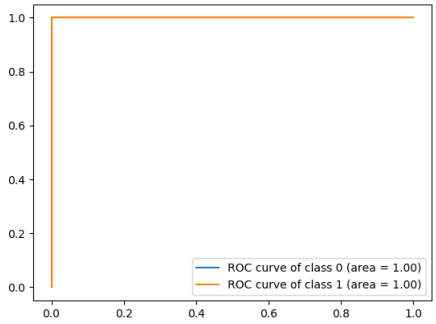


Figura 4: Corbes de ROC peel *Random Forest*

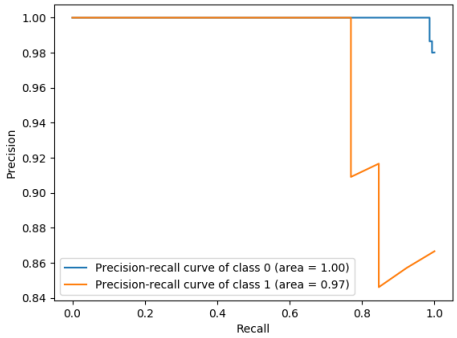


Figura 5: Corbes de PR per el *KNN*

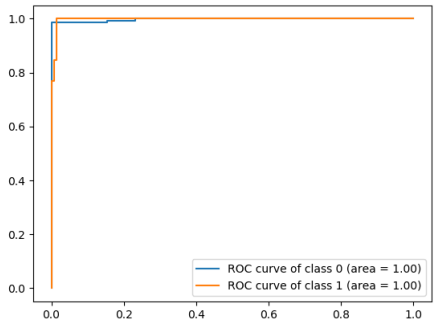


Figura 6: Corbes de ROC per el *KNN*

Observem que en el cas del *Random Forest* la precisió és 1, i, per tant, l'àrea sota la corba és màxima en tots dos casos. En canvi, pel *KNN*, per trobar una de les classes la recerca és quasi perfecte segons la *ROC*, però, en la *PR* observem pateix algunes variacions i dificultats en classificar.

Per assegurar la classificació observem el *classification_report* (que retorna la precisió, el *recall* i el *F1 score* per cada classe) dels dos models estudiats.

S'observa que les dues taules coincideixen i són la següent:

Distancia	**Precision**	**Recall**	**f1-score**	**support**
No llegendari	1.00	1.00	1.00	731
Llegendari	0.97	1.00	0.99	70
accuracy			1.00	801
macro avg	0.99	1.00	0.99	801
weighted avg	1.00	1.00	1.00	801

Per poder avaluar correctament amb un cas desfavorable en la predicció utilitzarem el model de la sigmoide per comparar.

Distancia	**Precision**	**Recall**	**f1-score**	**support**
No llegendari	0.91	0.90	0.91	731
Llegendari	0.08	0.09	0.08	70
accuracy			0.83	801
macro avg	0.49	0.49	0.49	801
weighted avg	0.84	0.83	0.84	801

De les dues taules podem concloure que, tot i que tant el *Random Forest* com el *SVM* amb *kernel sigmoide* tenen una precisió força alta, el que fa que el primer sigui superior és que classifica amb la mateixa efectivitat les dues classes mentre que el *SVM* classifica molt bé els Pokemon no llegendaris, però pràcticament no encerta els llegendaris.

2.6 Cerca d'hiperparàmetres

Els hiperparàmetres són uns valors que els models fan servir internament i que defineixen com serà el procés d'entrenament. Normalment, aquests paràmetres són independents de les dades. Per maximitzar el rendiment d'un model cal trobar la millor combinació d'hiperparàmentres. Destaquem dos mètodes:

- *Grid Search*: Consisteix en provar exhaustivament totes les combinacions possibles entre un conjunt de paràmetres donats per l'usuari (Els punts que defineixen cada combinació generen una malla). És un bon mètode per fer servir si el model que volem optimitzar no és gaire complex o si tenim una idea aproximada de quins valors haurien de prendre els hiperparàmetres.

- *Random search*: De manera similar al *Grid Search*, l'usuari genera una malla amb els hiperparàmetres que es volen provar, però ara les combinacions es trien aleatòriament. D'aquesta manera es pot explicitar quantes combinacions volem mirar. Això ve molt bé quan es treballa amb un model molt complex o es vol explorar una malla molt gran, on fer una cerca exhaustiva és inviable en termes de temps d'execució.

Si provem els dos algoritmes amb un model *Random Forest*, veiem com en aquest cas els dos troben la mateixa combinació de variables.

	n_estimators	**max_depth**	**Bootstrap**	R^2 **obtingut**	**temps (s)**
Grid Search	100	5	True	0.987577639	414.07
Random Search	100	5	True	1.0	395.863

Amb el model KNN, la *Grid Search* ha aconseguit trobar una combinació que dona un R^2 perfecte en les primeres iteracions mentre que l'opció aleatòria, com no fa una cerca exhaustiva de totes les combinacions, ha trobat una combinació pitjor, no només per tenir un R^2 més baix, sinó també per generar un model més complex.

	Leaf Size	p	**n_neighbors**	R^2 **obtingut**	**temps (s)**
Grid Search	1	1	1	1.0	82.6404
Random Search	44	1	5	0.99378882	16.075

Es pot observar el increment en el cost de recursos per trobar els valors de la primera taula en comparació a la segona.

2.7 Conclusions

S'ha pogut comprovar que el millor model de classificació per saber si un Pokemon és llegendari o no és el *Random Forest* que classifica de forma perfecta aquesta variable. Per fer el *Random Forest* de la millor forma s'utilitzarà **n_estimators=100**, **max_depth=5** i **bootstrap = TRUE**.

La resta de models que hem provat, amb diferents combinacions d'hiperparàmetres, produeixen resultats positius amb una precisió propera al 100%, el que ens indica que algunes de les variables determinen amb molta força si un Pokemon és o no llegendari.