PREDICCIÓ DEL GRUP D'EDAT

Pràctica de Mineria de Dades

EPSEVG
2024-2025
17/12/2024

Pau Ramos
Samuel Justo

1. Descripció de les dades

Propostes de datasets a utilitzar:

1. https://www.kaggle.com/datasets/abdullah0a/human-age-prediction-synthetic-dataset (predir l'edat d'una persona a partir d'estadístiques mèdiques)
2. https://www.kaggle.com/datasets/manishkc06/startup-success-prediction (no arriba a 1000 per poc, predir si una startup tindra exit o no)
3. https://www.kaggle.com/datasets/kelvinkelue/credit-card-fraud-prediction (predir si una transacció és fraudulenta)

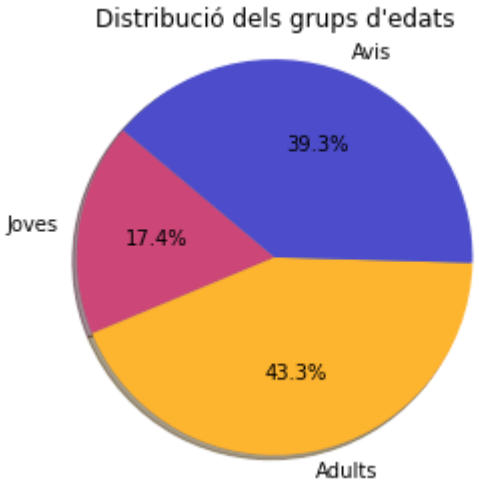
Finalment vam escollir el de dataset de Human Age Prediction Dataset que serviria per intentar predir l'edat segons majoritàriament dades mèdiques.

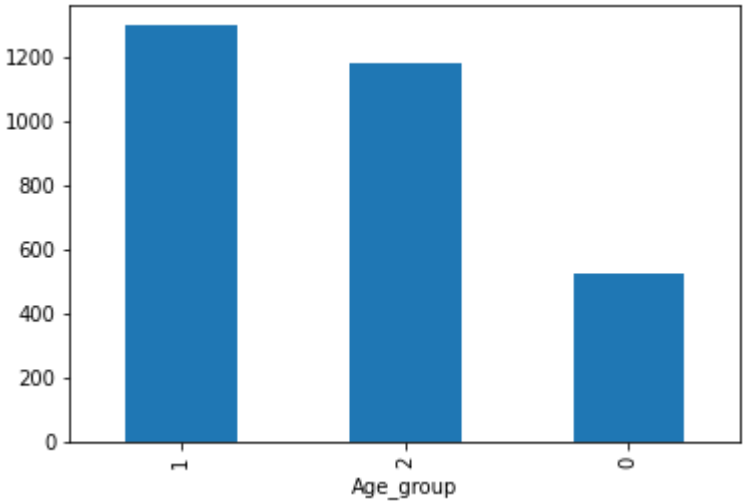
Atributs del dataset i els seus valors

1. Gender (Gènere)
2. Height (cm) (Alçada en centímetres)
3. Weight (kg) (Pes en quilograms)
4. Blood Pressure (s/d) (Pressió arterial en format sistólica/diastólica)
5. Cholesterol Level (mg/dL) (Nivell de colesterol en mil·ligrams/decilitre)
6. BMI (Índex de massa corporal)
7. Blood Glucose Level (mg/dL) (Nivell de glucosa a la sang en mil·ligrams/decilitre)
8. Bone Density (g/cm^2) (Densitat òssia en grams/centímetre quadrat)
9. Vision Sharpness (Nitidesa de la visió)
10. Hearing Ability (dB) (Habilitat auditiva en decibels)
11. Physical Activity Level (Nivell d'activitat física)
12. Smoking Status (Estat fumador)
13. Alcohol Consumption (Consum d'alcohol)
14. Diet (Dieta)
15. Chronic Diseases (Malalties cròniques)
16. Medication Use (Ús de medicaments)
17. Family History (Historial de malalties cròniques de la família)
18. Cognitive Function (Funcions cognitives)
19. Mental Health Status (Nivell de salut mental)
20. Sleep Pattern (Patró de descans)
21. Stress Levels (Nivells d'estrès)
22. Pollution Exposure (Exposició a la contaminació)
23. Sun Exposure (Exposició al sol)
24. Education Level (Nivell d'educació)
25. Income Level (Nivell econòmic)
26. Age (years) (Edat en anys)
27. Age_group (Grup d'edats al que pertany)

Representació gràfica d'atributs

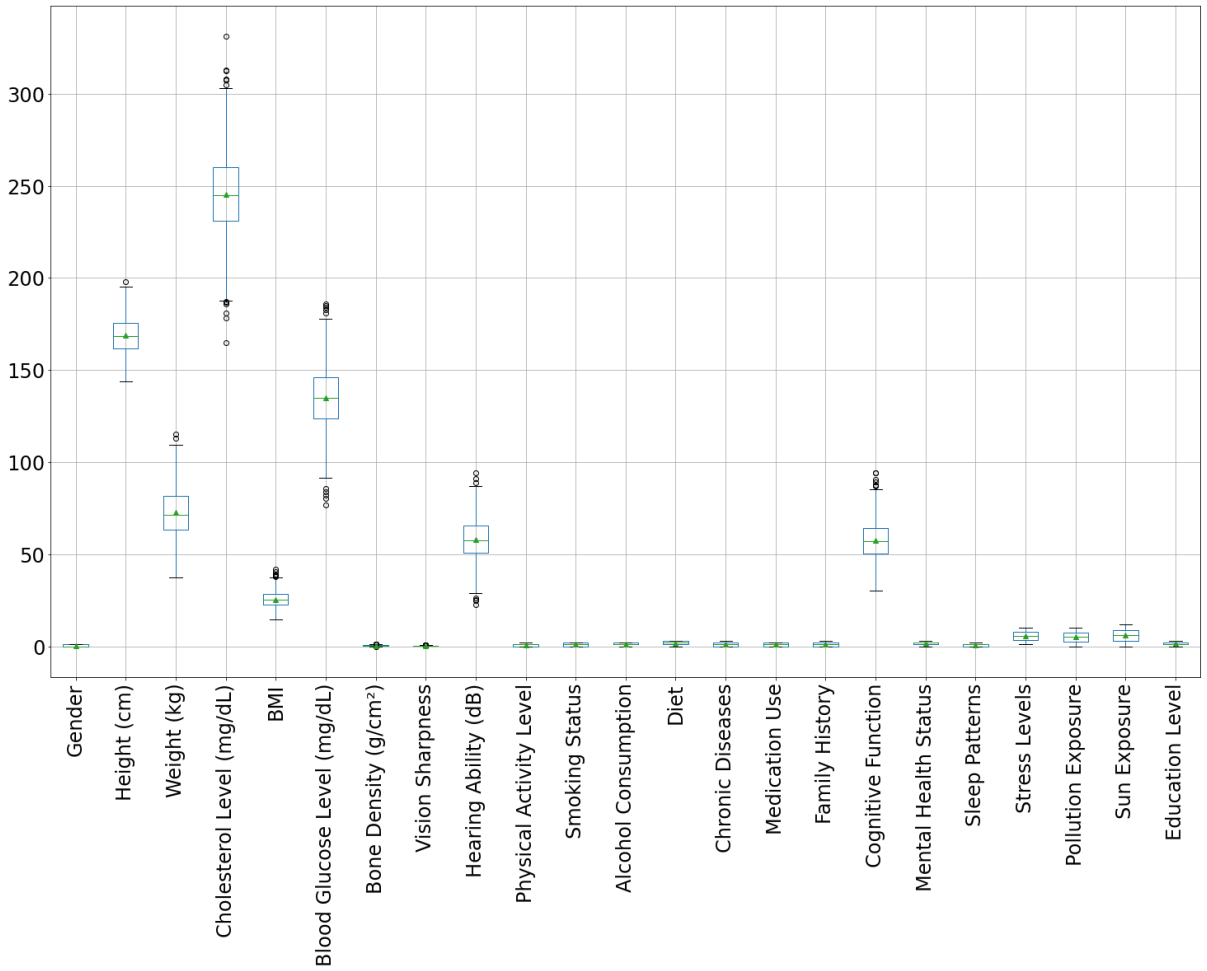
Divisió dels tres grups en els que hem repartit les dades:



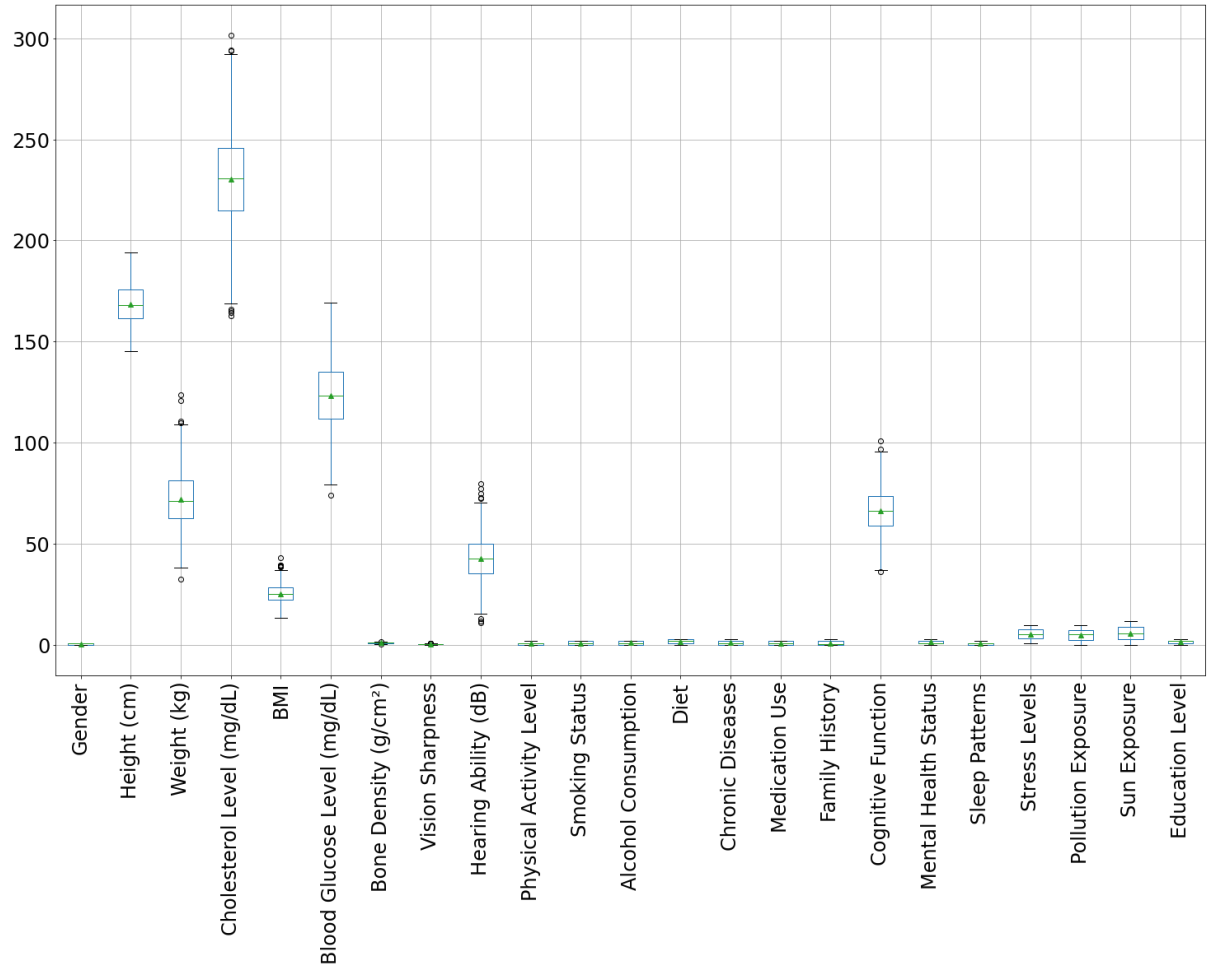


Boxplot de les dades, on es veuen els valors típics i els outliers:

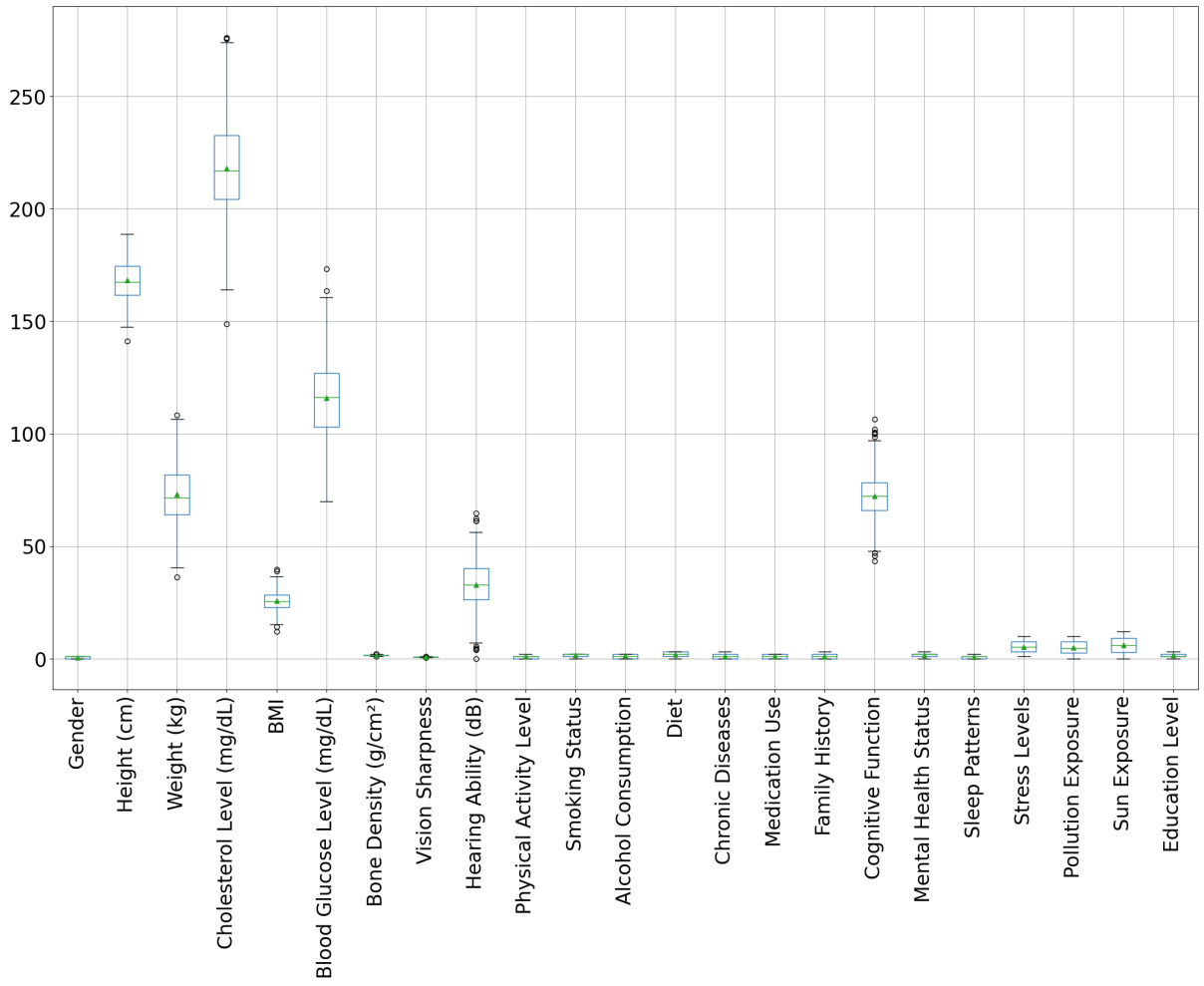
Avis:



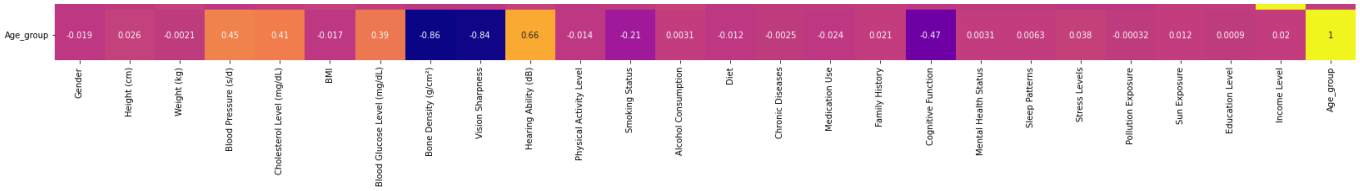
Adults:



Joves:



Relació dels atributs amb l'objectiu



Relació de cada atribut amb l'age_group (el paràmetre a predir), on -1 és molt relacionat inversament, és a dir, que quan aquest valor és més baix, més alt és el grup d'edat, i 1 és el contrari, com més alt el valor, més alt el grup d'edat. Com podem veure en la imatge, els paràmtres amb més relació amb l'objectiu són, la densitat òsea, la pressió de la sang, el colesterol, la glucosa, la nitidesa de la visió, la oïda i les funcions cognitives.

2. Pre-processament de les dades

Primer, hem afegit la columna age_group, que volem predir, a partir de la columna age, ja que aquesta conté enters que no són dinàmics i cal convertir-los a una dada que es pugui predir, en el nostre cas, grups per edat:

Age groups:

- [18-30]: Jove (0)
- [31-60]: Adult (1)
- [61-...): Jubilat (2)

També s'han de modificar els atributs de tipus categòric a tipus numeral de la següent forma:

Gender:
- Male (0)
- Female(1)

Mental Health Status:
- Poor (0)
- Fair (1)
- Good (2)
- Excellent (3)

Smoking Status:
- Current (0)
- Former (1)
- Never (2)

Alcohol Consumption:
- Frequent (0)
- Occasional (1)

- Never (2)

Medication Use:
- Regular (0)
- Occasional (1)
- No (2)

Physical Activity Level:
- Low (0)
- Moderate (1)
- High (2)

Diet:
- High-fat (0)
- Low-carb (1)
- Vegetarian (2)
- Balanced (3)

Sleep Patterns:
- Insomnia (0)
- Normal (1)
- Excessive (2)

Income Level:
- Low (0)
- Medium (1)
- High (2)

Education Level:
- No Education (0)
- Undergraduate (1)
- High School (2)
- Postgraduate (3)

Chronic Diseases:
- No Disease (0)
- Hypertension (1)
- Diabetes (2)
- Heart Disease (3)

Family History:
- No Disease (0)
- Hypertension (1)
- Diabetes (2)
- Heart Disease (3)

3. Models de mineria de dades

- **Single Fold Cross Validation**

Single Fold cross Validation és un mètode de machine learning que consisteix en dividir una sola vegada el dataset i evaluar a partir d'això el model. Les dades s'han de dividir entre entrenament del model (training) i una part per provar (test) el model entrenat. Per executar el Single Fold CV, farem servir un RandomForestClassifier, un tipus de classificador automàtic que pertany als arbres d'aprenentatge. És necessari trobar els millors paràmetres per cada algorisme per optimitzar la solució, així que es farem servir el GridSearchCV per trobar la màxima profunditat del random forest classifier. Per fer la búsqueda, RandomForestClassifier té aquests paràmetres en específic:

- estimadors = 100: arbres que es crearan al construir el random forest, número alt però no massa alt per evitar massa temps de processament.
- cv = 5: número de crossfold (divisóns del dataset) per trobar la millor profunditat per l'arbre
- random state = 20: no afecta el dataset, genera nombres aleatoris i dona l'opció de poder-los reproduir en el mateix estat

```
# Trobar millor profunditat per a randomforest
scaler = StandardScaler()

# Guardar a X tots els atributs menys el grup d'edats, a Y el grup d'edats
X = data_sf.drop('Age_group', axis=1)
y = data_sf['Age_group']

scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(scaled, y, test_size=0.2, random_state=10)

param_grid = {'max_depth': [5, 10, 15, 20, None]}


#Grid search per classificador validation
grid_search = GridSearchCV(RandomForestClassifier(n_estimators=100, random_state=20), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print('Millor profunditat trobada:', grid_search.best_params_['max_depth'])
✓ 15.4s
Millor profunditat trobada: 10
```

Una vegada trobada la millor profunditat, s'ha de crear el model, amb els següents paràmetres:

- estimadors = 50: arbres que es crearan al construir el random forest, número alt però no massa alt per evitar massa temps de processament.(mateix anterior pas)
- max_depth serà el valor trobat per la optimització de profunditat anterior, en el nostre cas, 10.
- min_sample_split= 4: nombre mínim de mostres necessàries per dividir un node de l'arbre, un node no es dividirà si conté menys de 4 mostres, per evitar dividir massa i no tenir overfitting de dades.
- min_sample_leaf = 2: nombre mínim mostres requerides per cada fulla de l'arbre, cada fulla requereix el mínim de 2 mostres, evita que les

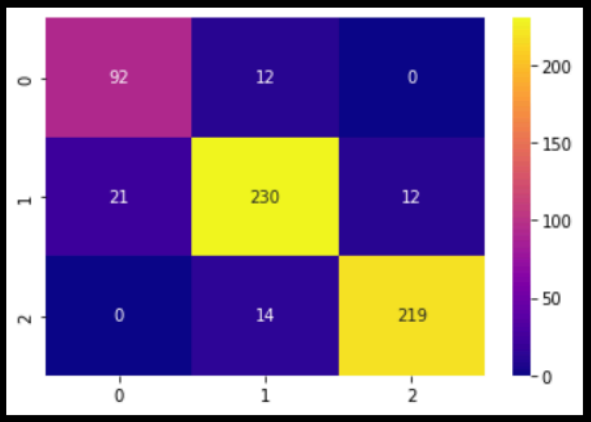
fulles tinguin poques mostres per no crear molt soroll en les prediccions

- random state = 20: no afecta el dataset, genera nombres aleatoris i dona l'opció de poder-los reproduir en el mateix estat

INFORME MODEL SINGLE FOLD CV

	precision	recall	f1-score	support
0	0.81	0.88	0.85	104
1	0.90	0.87	0.89	263
2	0.95	0.94	0.94	233
accuracy			0.90	600
macro avg	0.89	0.90	0.89	600
weighted avg	0.90	0.90	0.90	600

MATRIU DE CONFUSIÓ



RESULTATS ACCURACY
Accuracy: 0.9016666666666666
Training Accuracy (entrenat): 0.9741666666666666
Testing Accuracy (test): 0.9016666666666666

- **Precision (Precisió):** indica el percentatge de prediccions positives del model són correctes. Per la classe 0, els joves, una precisió positiva del 81% i per la classe 1, els adults, una precisió del 90% i pels jubilats, una precisió del 95%, resultats molt alts i bons.

- **Recall (Recuperació):** indica el percentatge de mostres positives reals que han pogut ser identificades correctament pel model. En aquest model hi ha un recall per la classe 0, d'un recall positiu del 88% i per la classe 1, d'un recall positiu del 87% i per la classe 2 un 94%.

- **F-score (Puntuació F1):** la mitjana harmònica entre el recall i la precisió, sent una mètrica equilibrada del rendiment del model. Per la

classe 0 és del 0’85, per la classe 1 del 0’89 i per la classe 2 d’un 0’94. Una accuracy total del 90%.

- **Support (Suport):** mostra quantes mostres de cada classe s’ha utilitzat per calcular els resultats anteriors.

- **Matriu de Confusió:** mostra el nombre de prediccions correctes (diagonal-esquerra a dreta [0, 0], [1, 1], [2, 2]) i les prediccions incorrectes, la resta de caselles. El model fa 92 prediccions correctes per la classe 0, 230 prediccions correctes per la classe 1 i 219 prediccions correctes per la classe 2, fa 12 prediccions incorrectes de la classe 0, 33 prediccions incorrectes de la classe 1 i 14 prediccions incorrectes per la classe 2.

- **Precisió model:** Ens retorna que el model té 90%, a les dades entrenades retorna una accuracy correctament el 97’4% dels casos i les del test un 90%.

- K-Fold Cross Validation

K-fold consisteix en dividir K vegades el dataset i repetir en diverses iteracions el procés d’anàlisis per aconseguir una avaluació i estimació més robusta que el single Fold CV.

Per executar K-fold CV, es fa servir KNeighborsClassifier, que classifica les noves dades en funció dels veïns més propers del conjunt de dades entrenades.

És necessari trobar els millors paràmetres per cada algorisme per optimitzar la solució, així que es fa servir GridSearchCV per trobar el nombre de veïns pel KNeighborsClassifier.
Per fer la búsqueda més optimitzada, el GridSearch i el KNeighborsClassifier, tenen aquests paràmetres:

- **range(1,30):** trobar millor valor de n_neighbors(nombre veïns) que entre el rang de 1 a 29.
- **cv=5:** número de crossfold (divisions del dataset) per trobar la millor n_neighbors.

```
#Trobar millor parametre n_neighbors pel model de kfold cross validation
param_grid = {'n_neighbors': range(1, 30)}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_full, y_train_full)

print("Millor número de veïns:",grid_search.best_params_['n_neighbors'])
```
✓ 1.5s

```
Millor número de veïns: 18
```

Una vegada trobat el millor nombre de veïns, en el nostre cas 18, crear el model amb els següents paràmetres:

- n_splits = 5: cada bucle de K Fold es dividira 5 plecs iguals fent servir StratifiedKFold
- shuffle = True: les particions de les dades siguin aleatòries.
- random state = 42

```
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

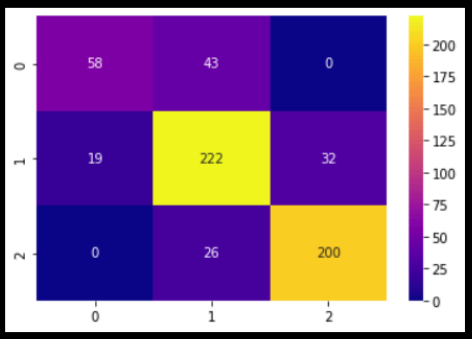
```
INFORME MODEL K FOLD CV

              precision    recall  f1-score   support

           0       0.75      0.57      0.65       101
           1       0.76      0.81      0.79       273
           2       0.86      0.88      0.87       226

    accuracy                           0.80       600
   macro avg       0.79      0.76      0.77       600
weighted avg       0.80      0.80      0.80       600

MATRIU DE CONFUSIÓ
```




```
 MITJANES DE CADA FOLD
Average Accuracy: 0.8075000000000001
Average Precision: 0.8127840626471837
Average Recall: 0.8075000000000001
Average F1 Score: 0.802333806176701
```

- **Precision (Precisió):** Per la classe 0 una precisió del 75%, per la classe 1, una precisió positiva del 76% i per la classe 2, un 86%, resultats molts més baixos que el single fold.

- **Recall (Recuperació):** En aquest model hi ha un recall per la classe 0, d'un recall del 57%, per la classe 1 un recall del 81% i per la classe 2 un recall del 88%.

- **F-score (Puntuació F1):** Per la classe 0 és d'aproximadament 0'65, per la classe 1 del 0'79 i per la classe 2 del 0'87. Una accuracy total del 80%.

- **Support (Suport):** mostra quantes mostres de cada classe s'ha utilitzat per calcular els resultats anteriors.

- **Matriu de Confusió:** El model fa 58 prediccions correctes per la classe 0, 222 prediccions correctes per la classe 1 i 200 prediccions correctes per la classe 2, fa 43 prediccions incorrectes de la classe 0, 51 prediccions incorrectes de la classe 1 i 26 prediccions incorrectes per la classe 2.

- **Mitjanes del model:** Ens retorna que per cada fold la mitjana de l'accuracy es del 80'7%, la mitjana de la precisió un 81'2%, del recall un 80'7% i F-score del 80'2%.

4. Mètodes de machine-learning

- **Naive Bayes**

L'algorisme de Naive Bayes és un mètode de classificació fundamentat en el teorema de Bayes, que assumeix la independència de les dades entre si i expressa la probabilitat condicional d'un esdeveniment aleatori A donat B en termes de distribució de probabilitat condicional de l'esdeveniment B donat A i la distribució de probabilitat marginal de l'esdeveniment A.

Sense ajustar cap paràmetre:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

cv = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

gnb = GaussianNB()
cv_scores = cross_val_score(gnb,X=X,y=y,cv=cv)
np.mean(cv_scores)
```

On

● n_splits=10: les dades es dividiran en 10 subsets i el model s'entrenara i provara 10 vegades utilitzant un test set diferent cada vegada, StratifiedKFold

manté el percentatge de les classes, molt important per datasets no balancejats.

● shuffle=True: les particions de dades siguin aleatòries.

● (random_state=42)

S'obté el següent resultat

	precision	recall	f1-score	support
0	0.80	0.89	0.84	522
1	0.88	0.83	0.86	1298
2	0.92	0.92	0.92	1180
accuracy			0.88	3000
macro avg	0.86	0.88	0.87	3000
weighted avg	0.88	0.88	0.88	3000

- **Precision (Precisió):** Per la classe 0 una precisió del 80%, per la classe 1, una precisió del 88% i per la classe 2, un 92%, resultats bastant bons, sobretot per a les classes 1 i 2.

- **Recall (Recuperació):** En aquest model hi ha un recall per la classe 0 del 89%, per la classe 1 un recall del 83% i per la classe 2 un recall del 92%. Valors molt bons novament.

- **F-score (Puntuació F1):** Per la classe 0 és de 0'84, per la classe 1 del 0'86 i per la classe 2 del 0'92. Una accuracy total del 88%. Veiem que amb els valors per defecte s'obtenen bons resultats.

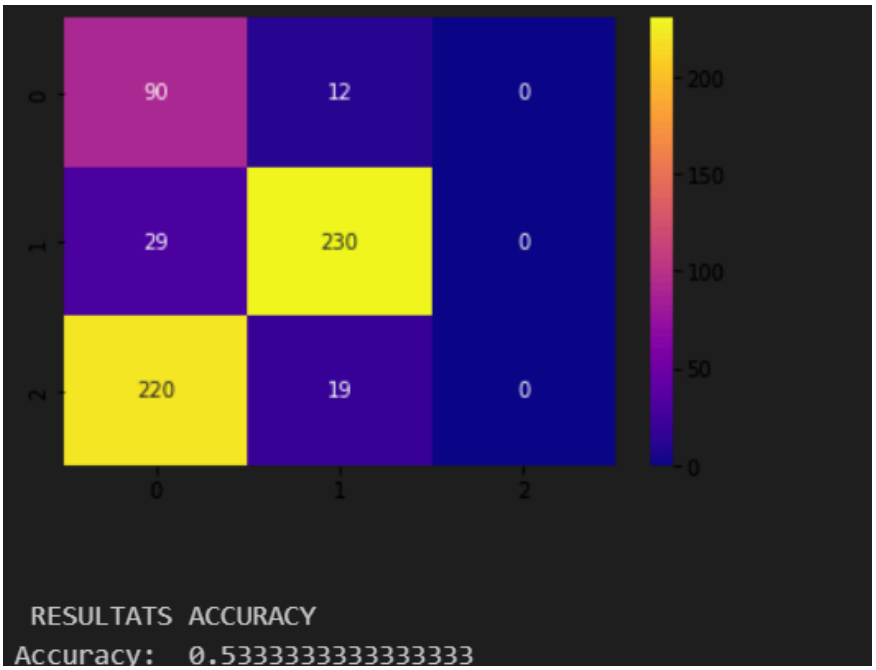
- **Matriu de Confusió:** El model fa 465 prediccions correctes per la classe 0, 1081 prediccions correctes per la classe 1 i 1090 prediccions correctes per la classe 2, fa 57 prediccions incorrectes de la classe 0, 217 prediccions incorrectes de la classe 1 i 90 prediccions incorrectes per la classe 2.

- **Accuracy del model:** Ens retorna que accuracy del model és del 87'87%.

Quan volem ajustar el threshold per optimitzar-ho, ens trobem amb un problema ja que es calcula a partir de la probabilitat d'una classe, per exemple 1, i assumeix que tota la resta són 0, obtenint els següents resultats:

	precision	recall	f1-score	support
0	0.27	0.88	0.41	102
1	0.88	0.89	0.88	259
2	0.00	0.00	0.00	239
accuracy			0.53	600
macro avg	0.38	0.59	0.43	600
weighted avg	0.43	0.53	0.45	600

Com no fa prediccions per 2, té un 0.00 en tot, 0 té una precissió molt baixa per aquest mateix motiu, i la precissió total del model:



Del 53,3%, naturalment surt baixa. Ens quedem amb la versió sense ajustar, que té una accuracy de 0.8786666666666667.

- **K-NN**

  L'algorisme K-Nearest Neighbors KNN, és un mètode utilitzat per la classificació i regressió de dades. Consisteix en un nombre K que indica quants veïns més propers a un valor es tindran en compte al fer una predicció, d'un valor gran de K, s'utilitzaran més veïns i un valor més petit menys.

  Per optimitzar el classificador, s'ha de buscar el valor de k que ens doni els millors resultats pel nostre dataset. Es pot fer donant un rang, per exemple de 1 a 15 i obtenint el resultat del accuracy que ens retorna:

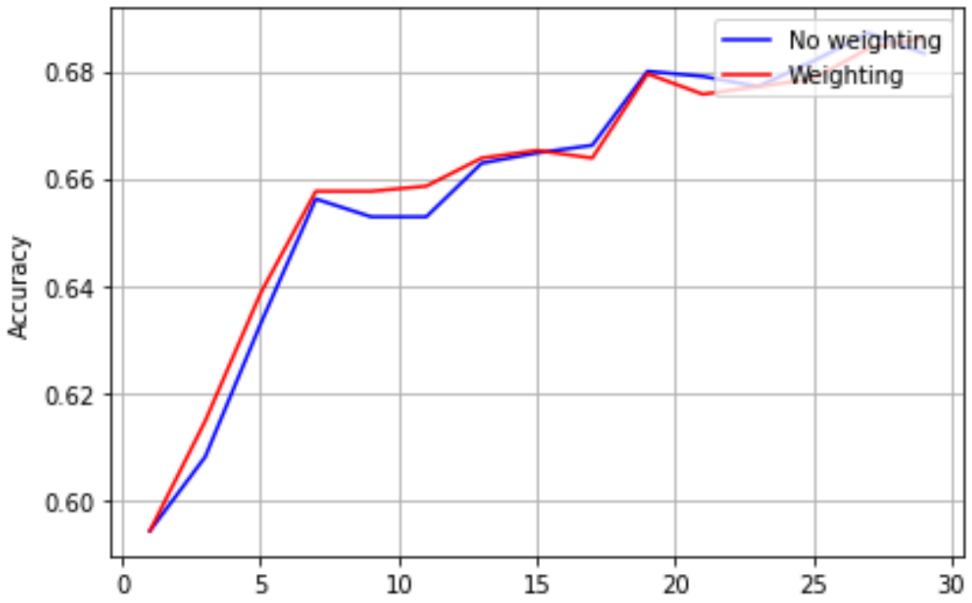
```python
accuracies = []

for k in range(1, 16):
    knn = nb.KNeighborsClassifier(n_neighbors=k)
    cv_scores = cross_val_score(knn, X=X_train, y=y_train,  cv=15)
    accuracies.append(np.mean(cv_scores))
    print(f"Accuracy {k} neighbour: {np.mean(cv_scores)}")
```

✓ 4.2s

```
Accuracy 1 neighbour: 0.5919047619047618
Accuracy 2 neighbour: 0.5695238095238094
Accuracy 3 neighbour: 0.6038095238095238
Accuracy 4 neighbour: 0.6257142857142857
Accuracy 5 neighbour: 0.6357142857142857
Accuracy 6 neighbour: 0.6442857142857144
Accuracy 7 neighbour: 0.6595238095238096
Accuracy 8 neighbour: 0.6523809523809524
Accuracy 9 neighbour: 0.6542857142857142
Accuracy 10 neighbour: 0.6576190476190477
Accuracy 11 neighbour: 0.6523809523809523
Accuracy 12 neighbour: 0.6576190476190475
Accuracy 13 neighbour: 0.6604761904761904
Accuracy 14 neighbour: 0.6657142857142857
Accuracy 15 neighbour: 0.6604761904761904
```

També es pot veure depenent del tipus de K, ja sigui amb una ponderació usant el pes (weights) dels valors amb les diferents opcions per la distància o uniforme (sense utilitzar les distàncies entre els veïns):



El gràfic ens dona una representació de l'Accuracy i les relacions amb els valors de k, siguin calculats amb distància (vermell) o no (blau). Podem observar que el millor valor de K és donat per els valors sense pes, amb una accuracy de 0'6871, un 68'7%. De la manera més senzilla que podem buscar els millors paràmetres per knn, és utilitzant GridSearchCV:

- **range(1,30):** trobar millor valor de k entre el rang de 1 a 29.

- **weights= ('distance','uniform'):** trobar si els resultats són millors calculant k amb la ponderació de la distància entre els veïns o no.

```
params = {'n_neighbors':list(range(1,30,2)), 'weights':('distance','uniform')}
```

- **param_grid:** valor trobat per la optimització de k feta abans.
- **cv= 10:** número de crossfold (divisions del dataset) per trobar el millor valor de k.
- **n_jobs= -1:** per millorar el temps d'execució i realitzar una cerca en paral·lel

```
knc = nb.KNeighborsClassifier()
clf = GridSearchCV(knc, param_grid=params,cv=10,n_jobs=-1)
clf.fit(X_train, y_train)
```

Els millors paràmetres que troba per KNN són:

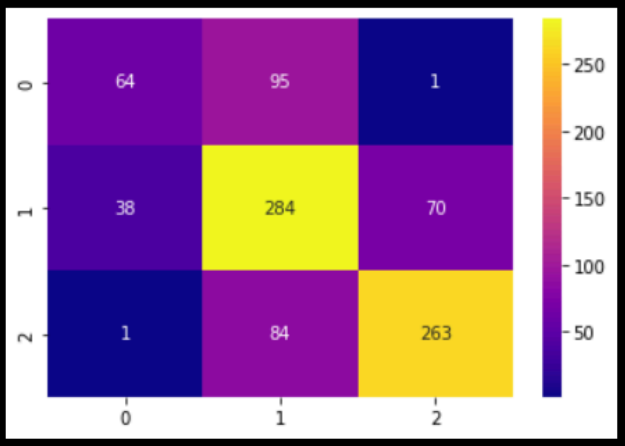
```
{'n_neighbors': 27, 'weights': 'uniform'} Accuracy= 0.6871428571428572
```

Valors que coincideixen amb el gràfic fet anteriorment, amb el millor valor de K sent 27 i amb el pes de les dades uniforme i amb una accuracy del 68'7%.

INFORME MODEL KNN

	precision	recall	f1-score	support
0	0.62	0.40	0.49	160
1	0.61	0.72	0.66	392
2	0.79	0.76	0.77	348
accuracy			0.68	900
macro avg	0.67	0.63	0.64	900
weighted avg	0.68	0.68	0.67	900

MATRIU DE CONFUSIÓ



RESULTATS ACCURACY

Accuracy 0.6788888888888889

- **Precision (Precisió):** Per la classe 0 una precisió del 62%, per la classe 1, una precisió positiva del 61% i per la classe 2, un 79%, resultats molts més baixos que el single fold.

- **Recall (Recuperació):** En aquest model hi ha un recall per la classe 0, d'un recall del 40%, per la classe 1 un recall del 72% i per la classe 2 un recall del 76%.

- **F-score (Puntuació F1):** Per la classe 0 és d'aproximadament 0'49, per la classe 1 del 0'66 i per la classe 2 del 0'77. Una accuracy total del 68%.

- **Support (Suport):** mostra quantes mostres de cada classe s'ha utilitzat per calcular els resultats anteriors.

- **Matriu de Confusió:** El model fa 64 prediccions correctes per la classe 0, 284 prediccions correctes per la classe 1 i 263 prediccions correctes per la classe 2, fa 96 prediccions incorrectes de la classe 0, 108 prediccions incorrectes de la classe 1 i 85 prediccions incorrectes per la classe 2.

- **Accuracy del model:** Ens retorna que accuracy del model és del 67'8%.

- Decision Trees

Un arbre de decisió és una representació gràfica en forma d'arbre on cada node representa una característica o pregunta d'un conjunt de dades, cada fulla representa una possible resposta a aquesta pregunta. Per optimitzar el classificador tots els possibles paràmetres que es poden especificar s'han trobat fent servir GridSearchCV:

- **criterion= ['entropy', 'gini']:** determina la funció utilitzada per mesurar la qualitat de la divisió.

- **max_depth= [None, 2, 5, 10, 20, 30]:** la profunditat màxima de l'arbre.

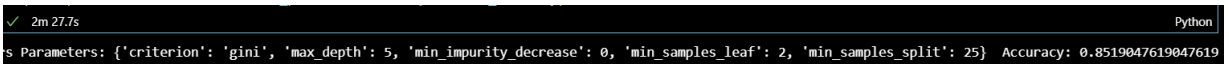
- **min_impurity_decrease= [0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]:** un node es dividirà si aquesta divisió indueix una impuresa superior o igual aquests valors, podrà ser algun valor entre els valors de la llista.

- **min_samples_split=[2, 5, 10, 15, 20, 25]:** el nombre mínim de valors per dividir un node intern.

- **min_samples_leaf= [1, 2, 4]:** el nombre mínim de valors per ser una fulla de l'arbre, podrà ser algun valor entre els valors de la llista.

```
param_grid = {
    'criterion': ['entropy', 'gini'],
    'max_depth': [None, 2, 5, 10, 20, 30],
    'min_impurity_decrease': [0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5],
    'min_samples_split': [2, 5, 10, 15, 20, 25],
    'min_samples_leaf': [1, 2, 4],
}
```

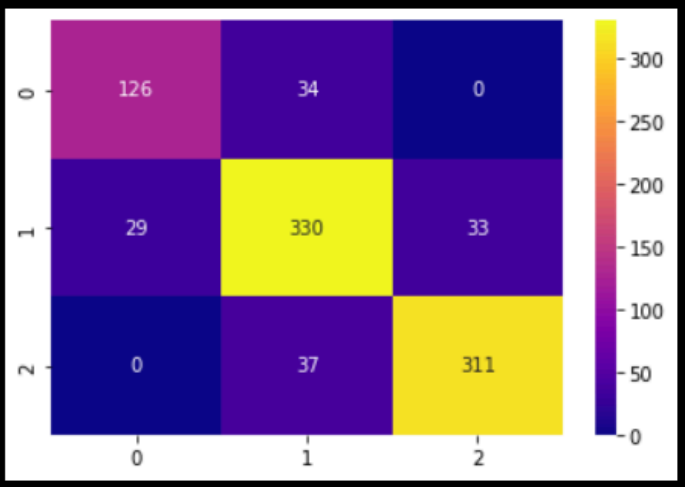
Els millors paràmetres que troba són:



INFORME DECISION TREE

	precision	recall	f1-score	support
0	0.81	0.79	0.80	160
1	0.82	0.84	0.83	392
2	0.90	0.89	0.90	348
accuracy			0.85	900
macro avg	0.85	0.84	0.84	900
weighted avg	0.85	0.85	0.85	900

MATRIU DE CONFUSIÓ



RESULTATS ACCURACY

Accuracy 0.8522222222222222

- **Precision (Precisió):** Per la classe 0 una precisió del 81%, per la classe 1, una precisió positiva del 82% i per la classe 2, un 90%.

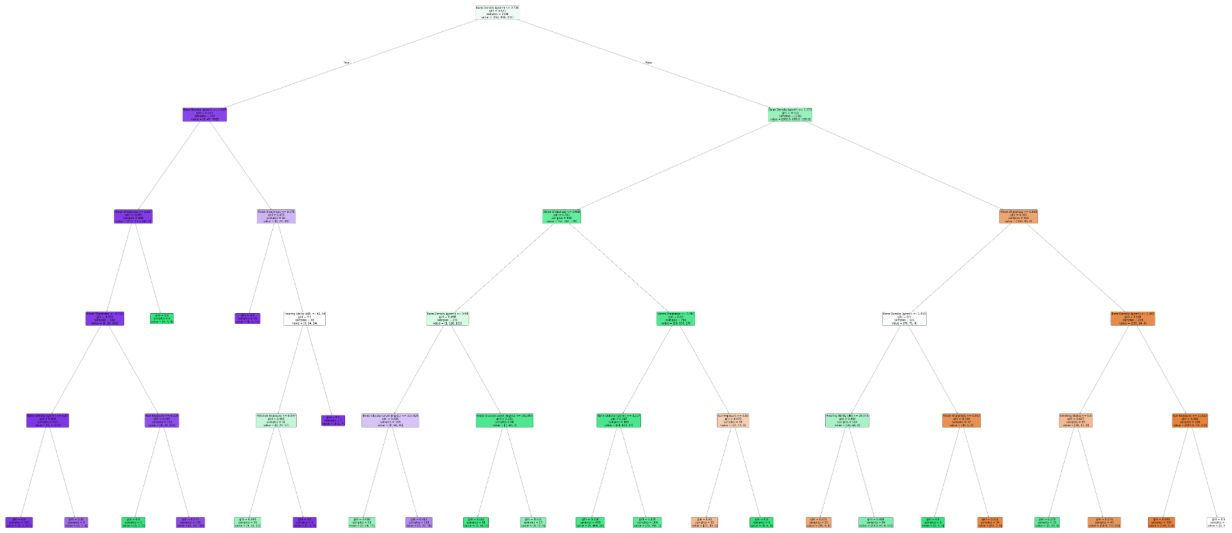
- **Recall (Recuperació):** En aquest model hi ha un recall per la classe 0, d'un recall del 79%, per la classe 1 un recall del 84% i per la classe 2 un recall del 89%.

- **F-score (Puntuació F1):** Per la classe 0 és d'aproximadament 0'80, per la classe 1 del 0'83 i per la classe 2 del 0'9. Una accuracy total del 85%.

- **Support (Suport):** mostra quantes mostres de cada classe s'ha utilitzat per calcular els resultats anteriors.

- **Matriu de Confusió:** El model fa 126 prediccions correctes per la classe 0, 330 prediccions correctes per la classe 1 i 311 prediccions correctes per la classe 2, fa 34 prediccions incorrectes de la classe 0, 62 prediccions incorrectes de la classe 1 i 37 prediccions incorrectes per la classe 2.

- **Accuracy del model:** Ens retorna que accuracy del model és del 85'2%.



L'arbre es pot veure amb més detall a **images/tree.png**

A l'arbre hi ha aquestes dades en cada node:

- **Categoria:** regla de decisió, menys a les fulles, serà la condició que divideix les mostres que arribin aquell node.

- **Entropia:** valor per mesurar la impuresa del node, si totes les mostres d'aquest node pertanyen a la mateixa classe.

- **Samples:** El nombre de mostres de dades que arriben aquell node.

- **Value:** La distribució de les classes de les mostres en aquest node, la classe 0, la classe 1 i la classe 2.

- **SVM**

  - **SVM de kernel lineal**

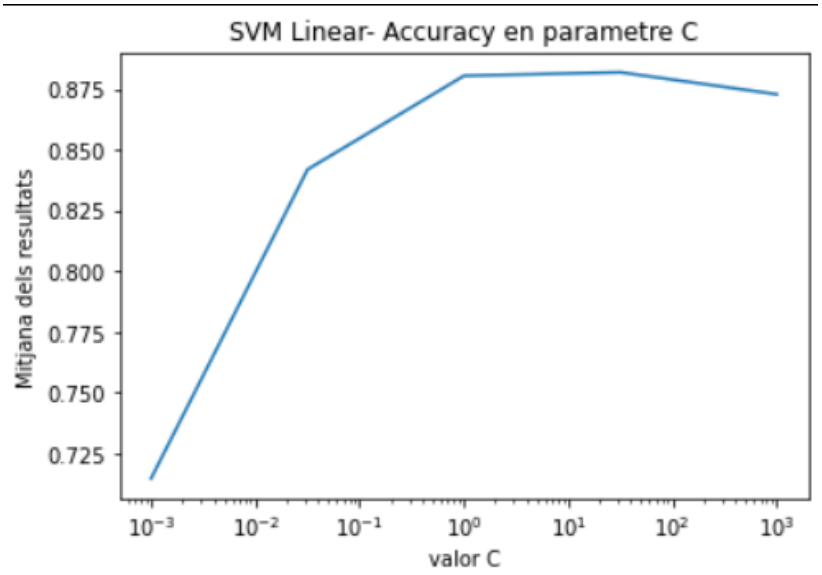
    Amb el Kernel lineal, intenta separar les dades amb una línia. Per optimitzar el SVM lineal, s'ha de fer servir GridSearchCV per trobar el

paràmetre C més òptim, quan més petit retorna una línia a la superfície més plana i un
valor més gran intenta classificar tots els valors del training dins del model:

- **Cs = np.logspace(-3, 3, num=5, base=10.0):** Crea una llista de 5 valors per C entre el rang de 10^-3 a 10^3 a una escala logarítmica, perquè entrin dades des de underfitting i overfitting del model.

- **cv= 5:** es fan 5 folds de cross fold validation.

```
Cs = np.logspace(-3, 3, num=5, base=10.0)
param_grid_linear = {'C': Cs, 'kernel': ['linear']}
```



Quan el valor de C més a prop de 10^1 l'accuracy en el model arribarà al 87'4%.

- **SVM de kernel polinomial**

Amb el SVM el Kernel polinomial o poly és utilitzat quan els atributs i l'objectiu no són linears, permet ajustar les dades en una forma no lineal. Per optimitzar el SVM poly, hem de fer servir el GridSearchCV per trobar el valor de C i de degree, el valor del grau polinominal que es trobi, quan més alts i més complexes hi ha risc de overfitting dins el model:

- **Cs = np.logspace(-3, 3, num=4, base=10.0):** Crea una llista de 4 valors per C entre el rang de 10^-3 a 10^3 a una escala logarítmica.

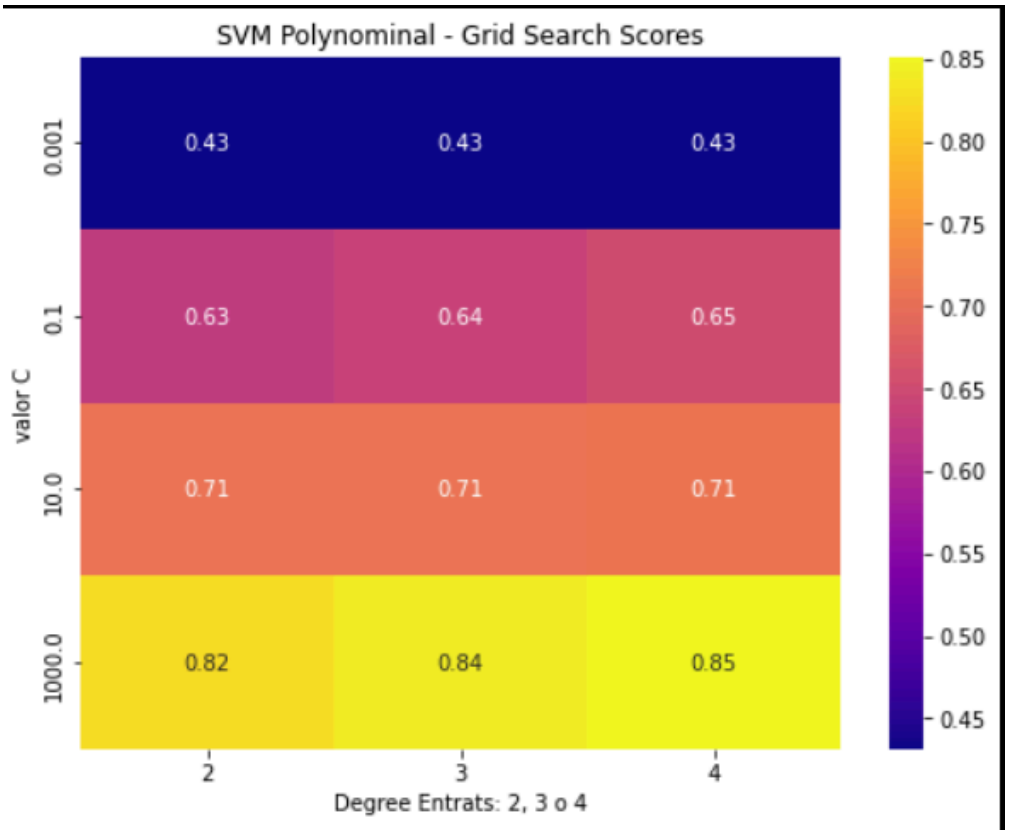
- **degree=[2,3,4]:** el grau polinominal del model

- **gamma=scale:** determina la influència d'un sol exemple del training, amb scale és un valor correcte per defecte.

- **cv= 3:** es fan 3 folds de cross fold validation.

- **n_jobs=-1** per millorar el temps d'execució i realitzar una cerca en paral·lel.

```
Cs = np.logspace(-3, 3, num=4, base=10.0)
param_grid_poly = {'C': Cs, 'kernel': ['poly'], 'degree': [2, 3,4], 'gamma': ['scale']}
```



Heatmap entre els valors de C i els grau polinominals entrats.

- **SVM de kernel rbf**

Per optimitzar el SVM rbf, s'ha fet servir GridSearchCV per trobar el valor de C i de gammas, el valor que determina la propagació del kernel, un valor petit determina una gran variància i punts fora del límit de decisió poden entrar en els càlculs, en canvi valors més grans resulten en una petita variància fent que el límit de decisió sigui dependent només dels punts que tingui aporop:

- **gammas= np.logspace(-4, 1, num=5, base=10.0):** Crea una llista de 5 valors per Gamma entre el rang de 10^-4 a 10^1 a una escala logarítmica.

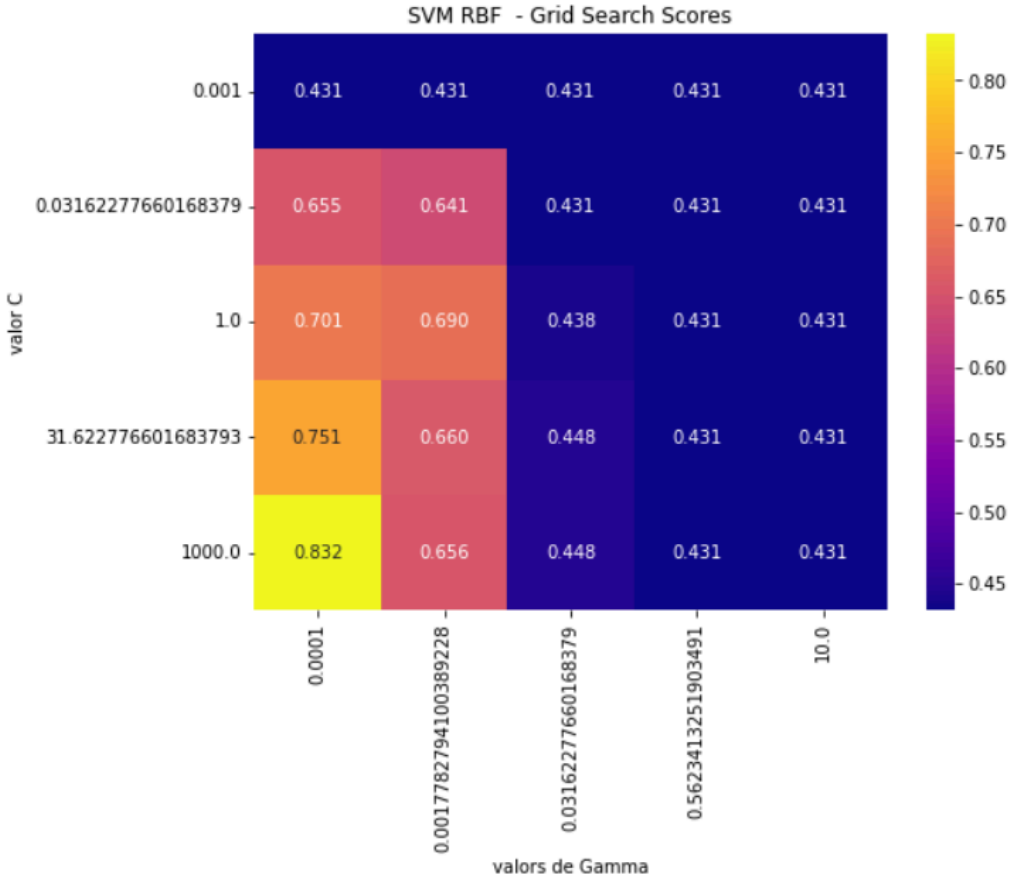
- **Cs = np.logspace(-3, 3, num=5, base=10.0):** Crea una llista de 5 valors per C entre el rang de 10^-3 a 10^3 a una escala logarítmica, perquè entrin dades des de underfitting i overfitting del model.

- **gamma=scale:** determina la influència d'un sol exemple del training, amb scale es un valor correcte per defecte.

- **cv=5:** es fan 5 folds de cross fold validation.

```
gammas = np.logspace(-4, 1, num=5, base=10.0)
Cs = np.logspace(-3, 3, num=5, base=10.0)

param_grid_rbf = {'C': Cs, 'gamma': gammas, 'kernel': ['rbf']}
grid_search_rbf = GridSearchCV(SVC(), param_grid_rbf, cv=5)
```

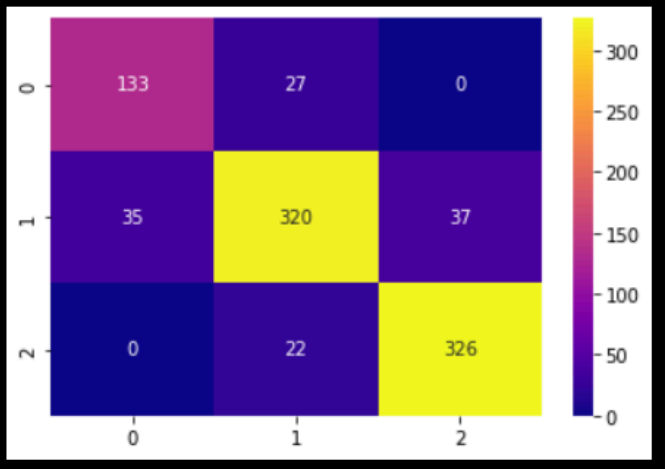


Heatmap entre els valors de C i els graus polinominals donats, com més groc, millor.

Resultats

RESULTATS KERNEL SVM LINEAR:

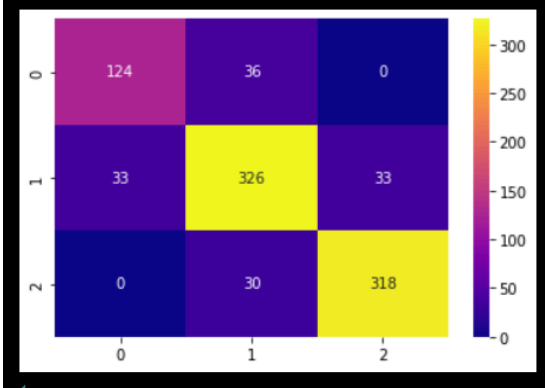
MATRIU DE CONFUSIÓ



ACCURACY en el test set: 0.8655555555555555
Millor parametre C: 31.622776601683793
Numero de suports: 483
Proporció dels supports: 0.23

RESULTATS KERNEL SVM POLYNOMIAL:

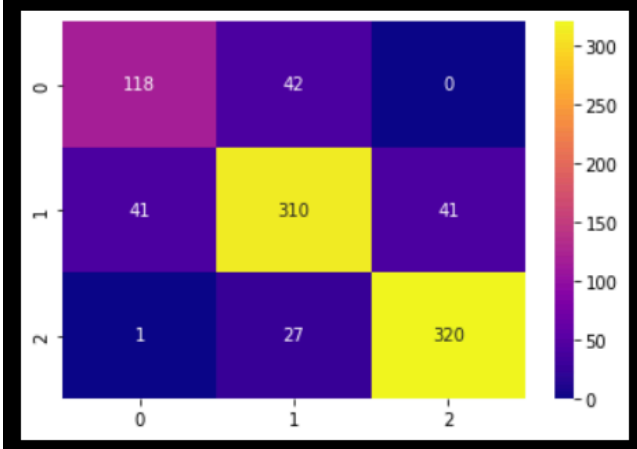
MATRIU DE CONFUSIÓ:



ACCURACY on the test set: 0.8533333333333334
Millors parameters: {'C': np.float64(1000.0), 'degree': 4, 'gamma': 'scale', 'kernel': 'poly'}
Numero de suports: 973
Proporció dels supports: 0.4633333333333333

```
RESULTATS KERNEL SVM RBF:

MATRIU DE CONFUSIÓ:
```




```
ACCURACY on test set:  0.8311111111111111
Milllors parameters:  {'C': np.float64(1000.0), 'gamma': np.float64(0.0001), 'kernel': 'rbf'}
Numero de suports:  928
Proporció dels supports:   0.4419047619047619
```

Tenint els resultats dels tres tipus de kernels per svm, podem comparar i triar quin és el kernel més adequat pel nostre dataset.

- Accuracy dels kernels:
    1. Linear: 86’5%
    2. Poly: 85’3%
    3. Rbf: 83’1%
- Paràmetres per C:
    1. Linear: 31’62
    2. Poly: 1000
    3. Rbf: 1000
- Número de suports / Proporció dels suports:
    1. Linear: 483 0’23
    2. Poly: 973 0’463
    3. Rbf: 928 0’442

Tenint en compte aquests factors:

1. **Accuracy** més alta amb el kernel **linear**.
2. **Model més senzill** (menys complexitat i menys suports) amb el kernel **linear**.
3. Les dades semblen ser **linealment separables** o gairebé.

El kernel **linear** és la millor elecció per aquest dataset. No només proporciona el millor rendiment en termes de precisió, sinó que també ofereix un model més simple, fàcil d’interpretar i menys propens al overfitting. Els kernels **Poly** i **Rbf** poden

ser més adequats en datasets més complexes o no linealment separables, però aquí no són necessaris.

5. Algorismes de meta-learning

- **Performance Majority Voting scheme**

  Una manera de millorar és utilitzant el mètode de Voting, aquest ajunta diversos classificadors independents i individuals i voten entre ells la predicció final. Hi ha dues variacions per votar: hard voting, on cada classificador fa una predicció d'una classe i la classe que rep més vots és la predicció final, mentres que soft voting fa la mitjana ponderada de la probabilitat de les classes.

  - **Hard voting**

```
eclf = VotingClassifier(estimators=[('nb', clf1), ('knn3', clf2), ('dt', clf3)], voting='hard')
scores = cross_val_score(eclf, X, y, cv=cv, scoring='accuracy')
print("Accuracy: %0.3f [%s]" % (scores.mean() , "Majority Voting"))
```

✓ 8.5s

```
Accuracy: 0.860 [Majority Voting]
```

    La accuracy que ens retorna és del 86%.

  - **Soft voting**

```
eclf = VotingClassifier(estimators=[('nb', clf1), ('knn3', clf2), ('dt', clf3)],voting='soft', weights=[2,1,2])
scores = cross_val_score(eclf, X, y, cv=cv, scoring='accuracy')
print("Accuracy: %0.3f [%s]" % (scores.mean(), "Weighted Voting"))
```

✓ 11.3s

```
Accuracy: 0.857 [Weighted Voting]
```

    L'accuracy que ens retorna es de 85,7%, sent aquesta la mitjana de la precisió dels tres mètodes.

  El hard voting té una millor accuracy amb un 86%, la diferència entre el soft voting i hard voting pot ser perquè els classificadors tenen diferents nivells de confiança a les seves prediccions i el soft voting aprofita totes les dades.

- **Bagging**

  El Bagging és una tècnica per millorar l'estabilitat i l'accuracy dels models d'aprenentatge automàtic. Per la seva millora es creen multiples subconjunt de les dades originals amb diferents estimadors per veure com l'augment d'aquests afecta a l'accuracy del model. Es fa servir la funció

BaggingClassifier per el model de DecisionTreeClassifier trobar l'accuracy amb cada iteració del codi.

```
lb=[]
for nest in [1,2,5,10,20,50,100]:
    scores = cross_val_score(BaggingClassifier(DecisionTreeClassifier(), n_estimators=nest), X, y, cv=cv, scoring='accuracy')
    print("Accuracy: %0.3f [%s]" % (scores.mean(), nest))
    lb.append(scores.mean())
```

✓ 1m 10.7s

```
Accuracy: 0.818 [1]
Accuracy: 0.816 [2]
Accuracy: 0.854 [5]
Accuracy: 0.858 [10]
Accuracy: 0.870 [20]
Accuracy: 0.870 [50]
Accuracy: 0.874 [100]
```

Després es pot comparar amb el mateix codi, però ara limitant el valor max_features= 0.45, aquest valor farà que cada arbre només utilitza el 45% de les seves característiques/atributs disponibles quan faci una divisió entre els nodes.

```
lb2=[]
print()
for nest in [1,2,5,10,20,50,100]:
    scores = cross_val_score(BaggingClassifier(DecisionTreeClassifier(), n_estimators=nest, max_features=0.45), X, y, cv=cv, scoring='accuracy')
    print("Accuracy: %0.3f [%s]" % (scores.mean(), nest))
    lb2.append(scores.mean())
```

✓ 32.1s

```
Accuracy: 0.734 [1]
Accuracy: 0.665 [2]
Accuracy: 0.789 [5]
Accuracy: 0.838 [10]
Accuracy: 0.846 [20]
Accuracy: 0.872 [50]
Accuracy: 0.876 [100]
```

És millor que en el mètode de bagging s'utilitza el valor de màximes característiques, ja que ajuda a tenir una accuracy de 87’6% amb el nombre d’estimadors a 100, mentres que amb el bagging sense límit podem aconseguir una accuracy del 87’4% amb també 100 estimadors, però tardant el doble de temps.

- **Random forest**

Combina la idea de bagging amb la selecció de característiques per múltiples arbres. Donada una llista d’estimadors, calcula pel model RandomForestClassifier la seva accuracy i si aquesta és millor que la millor accuracy anterior guardar aquell nombre d’arbres:

```
lrf=[]
for nest in [1,2,5,10,20,50,100]:
    scores = cross_val_score(RandomForestClassifier(n_estimators=nest), X, y, cv=cv, scoring='accuracy')
    print("Accuracy: %0.3f [%s]" % (scores.mean(), nest))
    lrf.append(scores.mean())
```

✓ 15.4s

```
Accuracy: 0.756 [1]
Accuracy: 0.761 [2]
Accuracy: 0.840 [5]
Accuracy: 0.855 [10]
Accuracy: 0.869 [20]
Accuracy: 0.875 [50]
Accuracy: 0.877 [100]
```

Podem observar que l'accuracy va augmentant bastant fins que es queda estancada a partir dels 50 arbres, per tant ens quedarem amb aquesta accuracy (87'5%).

- **Adaboost**

Adaboost és un mètode que utilitza la combinació de models febles, arbres de decisió amb poca profunditat, per aconseguir un model molt més fort. Va incrementant el pes de les dades mal classificades per cada iteració de manera que és concentra en els punts més problemàtics.
Tenint una llista d'estimadors, es fa servir la funció AdaBoostClassifier per trobar l'accuracy en cada iteració del codi.

```
lboo=[]
for nest in [1,2,5,10,20,50,100]:
    scores = cross_val_score(AdaBoostClassifier(n_estimators=nest), X, y, cv=cv, scoring='accuracy')
    print("Accuracy: %0.3f [%s]" % (scores.mean(), nest))
    lboo.append(scores.mean())
```

✓ 26.4s

```
Accuracy: 0.737 [1]
Accuracy: 0.770 [2]
Accuracy: 0.767 [5]
Accuracy: 0.796 [10]
Accuracy: 0.779 [20]
Accuracy: 0.759 [50]
Accuracy: 0.775 [100]
```

Podem veure que amb 10 estimadors és on més precisió s'aconsegueix (un 79'6%).

També es pot modificar el codi fent servir el AdaBoostClassifier amb un DecisionTreeClassifier com a base, especificant la profunditat amb max_depth=5, fent que els arbres puguin ser més complexos amb la profunditat màxima de 5 nivells

```
lboodt=[]
for nest in [1,2,5,10,20,50,100]:
    scores = cross_val_score(AdaBoostClassifier(DecisionTreeClassifier(max_depth=5),n_estimators=nest), X, y, cv=cv, scoring='accuracy')
    print("Accuracy: %0.3f [%s]" % (scores.mean(), nest))
    lboodt.append(scores.mean())
```

✓ 1m 8.9s

```
Accuracy: 0.852 [1]
Accuracy: 0.802 [2]
Accuracy: 0.804 [5]
Accuracy: 0.811 [10]
Accuracy: 0.833 [20]
Accuracy: 0.871 [50]
Accuracy: 0.867 [100]
```

On obtenim millor precisió és utilitzant 50 arbres de profunditat 5 amb una precisió del 87'1%.

- **Feature selection with forest of trees**

Aquesta és una tècnica per identificar quines són les característiques més importants a l'hora de fer una predicció i entrenar un model. En el nostre dataset els resultats obtinguts són:

```
[0.0031642  0.02107035 0.02013042 0.02809787 0.0405254  0.02168345
 0.03976994 0.29592434 0.22493158 0.10730358 0.00634646 0.02168666
 0.00580333 0.00714198 0.00589333 0.00530183 0.00683898 0.05109111
 0.00816747 0.00605884 0.02228771 0.02267178 0.02106658 0.00704281]
```

Hi ha un valor per cada una de les columnes del nostre dataset X (totes les columnes menys age_group). Com més gran el valor, més important és la característica.

Els valors més alts són 0.29592434, 0.22493158 i 0.10730358, que corresponen a les posicions 7,8 i 9 de l'array.

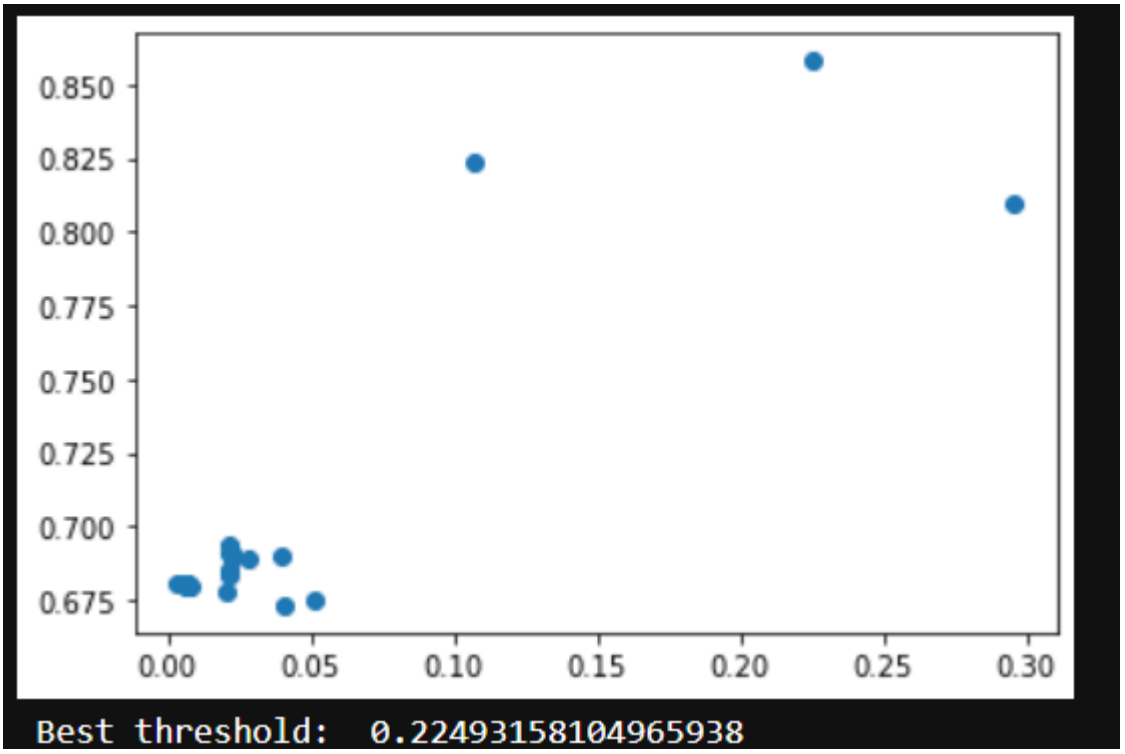
```
model = SelectFromModel(clf, prefit=True, threshold=0.05)
X_new = model.transform(X)
print(X_new.shape)

print(np.mean(cross_val_score(KNeighborsClassifier(), X=X, y=y, cv=cv, scoring='accuracy')))
print(np.mean(cross_val_score(KNeighborsClassifier(), X=X_new, y=y, cv=cv, scoring='accuracy')))
```

```
(3000, 4)
0.6410000000000001
0.654
```

Aplicant la selecció de característiques conseguim una millor accuracy, però molt lleu, aumenta d'un 64% a un 65%.

Calculant els thresholds obtenim els següents resultats:



On l'eix de les y es l'accuracy i el de les x el threshold, es veu clarament que el millor valor és el 0.22.

```
Original:  0.5923333333333334
With FS:  0.858
```

Amb Feature Selection i amb un threshold òptim, passem d'una accuracy del 59% a una de asi 86%.

6. Conclusions

El resum dels resultats obtinguts per a les diferents tècniques i models de predicció és el següent:

Cross Validation	Accuracy	En %
Single-fold Cross Validation	0.9016666	90,17%
K-Fold Cross Validation	0.8075	80,75%

Single-fold cross validation té una bona accuracy, del 90%, com només divideix una vegada el dataset potser no és tant representatiu com el F-Fold Cross Validation, en el que em obtingut un 80,75% de precisió, que no és un resultat dolent del tot.

Model / Classificador	Accuracy	En %
Naive Bayes	0.8787	87,9%
KNN	0.6788	67,9%
Decision Trees	0.8522	85,2%
SVM (linear)	0.8655	86,6%
Metha learning (Random forest)	0.875	87,5%

Obtenim uns resultats bastant similars d'entre el 85-88% per a tots els classificadors a excepció del KNN, que té un accuracy considerablement més baix.

Per poder fer una millor comparació, podem veure els resultats de cadascun dels models per fer comparacions entre precisió i recall per cada classe:

Model / Classificador	precision	recall	f1-score	support
Naive Bayes	0	0.80	0.89	0.84	522
	1	0.88	0.83	0.86	1298
	2	0.92	0.92	0.92	1180
	accuracy			0.88	3000
KNN	0	0.62	0.40	0.49	160
	1	0.61	0.72	0.66	392
	2	0.79	0.76	0.77	348
	accuracy			0.68	900

Decision Trees	0	0.81	0.79	0.80	160
	1	0.82	0.84	0.83	392
	2	0.90	0.89	0.90	348
	accuracy			0.85	900
SVM (linear)	0	0.79	0.84	0.82	101
	1	0.87	0.84	0.85	273
	2	0.90	0.92	0.91	226
	accuracy			0.87	600
Meta learning (Random forest)	0	0.81	0.78	0.80	101
	1	0.85	0.86	0.86	273
	2	0.91	0.92	0.91	226
	accuracy			0.87	600

El mètode que té un millor equilibri entre les diferents classes entre precisió i recall és Naive Bayes, que a més també és el mètode amb millor accuracy.