
MILK CLASSIFIER
Yaire Catalina López Santana - 2182061
Jeicob Gilmar Restrepo Gómez - 2183076

Introducción

Este proyecto busca poner en práctica algunos de los métodos de clasificación vistos en el curso de Inteligencia Artificial I. Nos basamos en un dataset llamado “milknew” el cual fue tomado de: https://www.kaggle.com/datasets/cpluzshrijayan/milkquality. Se decidió usar los métodos de Gaussian Naive Bayes, Decision Tree Classifier, Random Forest Classifier, Support Vector Machine y se hicieron algunas redes neuronales.

Dataset

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	2
1	6.6	36	0	1	0	1	253	2
2	8.5	70	1	1	1	1	246	0
3	9.5	34	1	1	0	1	255	0
4	6.6	37	0	0	0	0	255	1

El dataset consta de 8 columnas y 1059 filas

tomado de: https://www.kaggle.com/datasets/cpluzshrijayan/milkquality

Implementación

```
[8]  1 X_train20, X_test20, y_train20, y_test20 = train_test_split(X, y, test_size=0.3, random_state=21)
```

Para comparar los métodos con tres tipos de particionamientos procedemos a hacer el mismo pre-procesamiento para particiones 70-30 y 60-40

2 cells hidden

Gaussian Naive Bayes

A continuación entrenamos el modelo con el método de GNB y sacamos una métrica de Accuracy con este estimador que se obtuvo para las particiones 80-20, 70-30 y 60-40, respectivamente

3 cells hidden

Decision Tree Classifier

14 cells hidden

Random Forest classifier (RFC)

9 cells hidden

Support vector machine (SVM)

9 cells hidden

Red Neuronal con diferentes parametros

30 cells hidden

Métodos de Clasificación Usando ML

- Gaussian Naive Bayes:

En este método usamos 3 tipos de particionamientos: 80-20, 70-30 y 60-40 para así poder comparar los resultados obtenidos.

Gaussian Naive Bayes

A continuación entrenamos el modelo con el método de GNB y sacamos una métrica de Accuracy con este estimador que se obtuvo para las particiones 80-20, 70-30 y 60-40, respectivamente

```
est1GNB = GaussianNB()
est1GNB.fit(X_train20,y_train20)
y_pred20GNB = est1GNB.predict(X_test20)
print("predicciones = ")
print(y_pred20GNB)
print("ground truth")
print(y_test20)
print("accuracy de Gaussian Naive Bayes con el particionamiento 80-20 = ", accuracy_score(est1GNB.predict(X_test20), y_test20))
```

```
predicciones =
[2. 2. 1. 1. 0. 2. 0. 2. 0. 0. 2. 2. 1. 2. 0. 0. 0. 0. 2. 2. 2. 1. 1. 0.
 2. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 1. 1. 0. 0. 2. 2. 2. 1. 2. 2. 0. 2. 0. 0. 2. 1. 2. 2. 2. 0. 2. 1.
 2. 1. 2. 1. 1. 0. 1. 0. 2. 2. 1. 2. 2. 1. 2. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 1. 1. 1. 0. 2. 1.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 1. 2. 2. 1. 1. 0. 1. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 1. 1. 2. 0. 1. 2. 1. 0. 1. 2. 0. 1. 1. 2. 1. 2. 2. 1. 2. 2.
 0. 0. 0. 0. 1. 0. 2. 1. 0. 1. 2. 0. 0. 1. 2. 1. 1. 2. 2. 1. 0. 2. 1. 2.
 0. 0. 0. 1. 0. 0. 0. 2. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 2. 1. 1. 2. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 1. 0. 0. 1. 0. 2. 2. 2. 1. 0. 2. 0. 1. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 0. 0. 0. 2. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.
 0. 0. 2. 1. 0. 2.]
ground truth
[2. 1. 1. 1. 0. 2. 0. 0. 0. 0. 2. 2. 1. 2. 0. 0. 0. 0. 2. 2. 2. 1. 1. 0.
 1. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 1. 1. 0. 0. 2. 2. 1. 1. 2. 1. 0. 2. 0. 0. 1. 1. 2. 2. 2. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 2. 0. 1. 2. 2. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 1. 1. 1. 0. 2. 1.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 1. 1. 2. 1. 1. 0. 1. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 1. 1. 2. 0. 1. 2. 1. 0. 0. 2. 0. 1. 1. 2. 1. 1. 2. 1. 2. 1.
 0. 0. 0. 0. 1. 0. 2. 1. 0. 1. 1. 0. 0. 1. 2. 1. 1. 2. 2. 1. 0. 2. 1. 1.
 0. 0. 0. 1. 0. 0. 0. 1. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 2. 1. 1. 1. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 1. 0. 0. 0. 0. 0. 2. 2. 1. 0. 2. 0. 1. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1.
 0. 1. 0. 2. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 2. 1. 0. 1. 2. 0. 2. 1.
 2. 1. 0. 1. 1. 0. 1. 0. 0. 0. 2. 0. 2. 1. 1. 0. 2. 1. 1. 2. 0. 1. 0. 1.
```

Resultados de GNB

```
print("accuracy de GNB con el particionamiento 80-20 = ", accuracy_score(est1GNB.predict(X_test20), y_test20))
print("accuracy de GNB con el particionamiento 70-30 = ", accuracy_score(est2GNB.predict(X_test30), y_test30))
print("accuracy de GNB con el particionamiento 60-40 = ", accuracy_score(est3GNB.predict(X_test40), y_test40))
```

```
accuracy de GNB con el particionamiento 80-20 =  0.9150943396226415
accuracy de GNB con el particionamiento 70-30 =  0.9150943396226415
accuracy de GNB con el particionamiento 60-40 =  0.9221698113207547
```

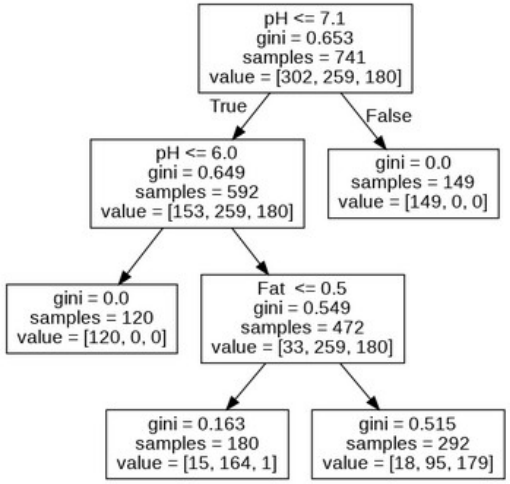
Con lo anterior se puede ver que este método no es muy recomendable para este dataset, ya que con los 3 tipos de particiones el resultado del accuracy no es muy confiable ya que supera el 90%.

Decision Tree Classifier

Con este método también se quiso comparar los resultados entre los 3 particionamientos y adicionalmente los resultados variando el max_depht

```
#creamos el arbol
export_graphviz(est1DTC, out_file='tree_est1DTC.dot', feature_names
!dot -Tpng tree_est1DTC.dot > tree_est1DTC.png

Image(filename='tree_est1DTC.png')
```



Decision Tree Classifier

```
#estimador de decision tree con max_depht = 3 y particionamiento 80-20
est1DTC = DecisionTreeClassifier(max_depth=3)
est1DTC.fit(X_train20,y_train20)
print("estimador de decision tree con max_depht = 3 y particionamiento 80-20 = ",accuracy_score(est1DTC.predict(X_test20), y_test20))
```

```
estimador de decision tree con max_depht = 3 y particionamiento 80-20 =  0.7924528301886793
```

```
y_pred20DTC = est1DTC.predict(X_test20)
print("predicciones = ")
print(y_pred20DTC)
print("ground truth")
print(y_test20)
```

```
predicciones =
[2. 2. 1. 2. 0. 2. 0. 2. 0. 0. 2. 2. 2. 2. 0. 0. 0. 0. 2. 2. 2. 1. 2. 0.
 2. 2. 2. 0. 2. 2. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 2. 1. 1. 0. 1. 2. 2. 2. 1. 2. 2. 0. 2. 0. 0. 2. 1. 2. 2. 2. 0. 2. 1.
 2. 1. 2. 1. 1. 0. 2. 0. 2. 2. 2. 2. 2. 1. 2. 0. 0. 1. 0. 1. 0. 0. 0. 2.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 2. 2. 1. 0. 2. 2.
 2. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 2. 2. 2. 1. 2. 0. 2. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 2. 1. 2. 0. 1. 2. 1. 1. 2. 2. 1. 1. 2. 2. 1. 2. 2. 1. 2. 2.
 0. 0. 0. 0. 2. 0. 2. 1. 0. 1. 2. 0. 0. 2. 2. 1. 2. 2. 2. 1. 0. 2. 1. 2.
 0. 0. 0. 1. 0. 0. 0. 2. 0. 2. 2. 0. 2. 0. 1. 0. 0. 0. 0. 0. 2. 1. 1. 2. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 2. 0. 0. 1. 0. 2. 2. 2. 2. 1. 0. 2. 0. 1. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 2. 0. 0. 2. 0. 2. 1. 2. 2.
 0. 2. 0. 2. 1. 0. 1. 2. 0. 2. 2. 1. 0. 0. 1. 1. 2. 2. 0. 1. 2. 1. 2. 2.
 2. 1. 0. 2. 1. 0. 2. 0. 0. 0. 2. 1. 2. 1. 1. 0. 2. 2. 1. 2. 0. 2. 0. 1.
 1. 0. 2. 1. 0. 2.]
ground truth
[2. 1. 1. 1. 0. 2. 0. 0. 0. 0. 2. 2. 1. 2. 0. 0. 0. 0. 2. 2. 2. 1. 1. 0.
 1. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 1. 1. 0. 0. 2. 2. 1. 1. 2. 1. 0. 2. 0. 0. 1. 1. 2. 2. 2. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 2. 0. 1. 2. 2. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 1. 1. 1. 0. 2. 1.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 1. 1. 2. 1. 1. 0. 1. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 1. 1. 2. 0. 1. 2. 1. 0. 0. 2. 0. 1. 1. 2. 1. 1. 2. 1. 2. 1.
 0. 0. 0. 0. 1. 0. 2. 1. 0. 1. 1. 0. 0. 1. 2. 1. 1. 2. 2. 1. 0. 2. 1. 1.
 0. 0. 0. 1. 0. 0. 0. 1. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 2. 1. 1. 1. 1.
```

Resultados de DTC

```
1 print("estimador de DTC con max_depht = 3 y particionamiento 80-20 = ",accuracy_
2 print("accuracy del DTC con max_depht = 6 y particionamiento 70-30= ",accuracy_s
3 print("estimador de DTC con max_depht = 10 y particionamiento 60-40 = ",accuracy
```

```
estimador de DTC con max_depht = 3 y particionamiento 80-20 =  0.7924528301886793
accuracy del DTC con max_depht = 6 y particionamiento 70-30=  0.8930817610062893
estimador de DTC con max_depht = 10 y particionamiento 60-40 =  0.9952830188679245
```

Podemos notar que el que obtuvo mejor accuracy fue el DTC con max_depht = 3 y particionamiento 80-20, aunque los otros valores sean mayores, pueden traer overfitting y por eso no son confiables.

- Random Forest Classifier

```
[79]  1 #RFC con un particionamiento de 70-30 y un n_estimators de 2
      2
      3 est2RFC = RandomForestClassifier(n_estimators=2)
      4 est2RFC.fit(X_train30,y_train30)
      5 print(accuracy_score(est2RFC.predict(X_test30), y_test30))
```

```
0.9905660377358491
```

```
[80]  1 y_pred30RFC = est2RFC.predict(X_test30)
      2 print("predicciones = ")
      3 print(y_pred30RFC)
      4 print("ground truth")
      5 print(y_test30)
```

```
predicciones =
[2. 1. 1. 1. 0. 2. 0. 0. 0. 0. 2. 2. 1. 2. 0. 0. 0. 0. 2. 2. 2. 1. 1. 0.
 1. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 1. 1. 0. 0. 2. 2. 1. 1. 2. 1. 0. 2. 0. 0. 1. 1. 2. 2. 2. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 2. 0. 1. 2. 2. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 1. 1. 1. 0. 2. 1.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 1. 1. 2. 1. 1. 0. 1. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 1. 1. 2. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 2. 1. 1. 2. 1. 2. 1.
 0. 0. 0. 0. 1. 0. 2. 1. 0. 1. 1. 0. 0. 1. 2. 1. 1. 2. 2. 1. 0. 2. 1. 1.
 0. 0. 0. 1. 0. 0. 0. 1. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 1. 0. 0. 0. 0. 0. 2. 2. 1. 0. 2. 0. 1. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1.
 0. 1. 0. 2. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 2. 1. 0. 1. 2. 0. 2. 1.
 2. 1. 0. 1. 1. 0. 1. 0. 0. 0. 2. 0. 2. 1. 1. 0. 2. 1. 1. 2. 0. 1. 0. 1.
 0. 0. 2. 1. 0. 2.]
ground truth
[2. 1. 1. 1. 0. 2. 0. 0. 0. 0. 2. 2. 1. 2. 0. 0. 0. 0. 2. 2. 2. 1. 1. 0.
 1. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 1. 1. 0. 0. 2. 2. 1. 1. 2. 1. 0. 2. 0. 0. 1. 1. 2. 2. 2. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 2. 0. 1. 2. 2. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 2. 0. 0. 2. 0. 1. 0. 0. 1. 0. 1. 2. 2. 0. 2. 1. 0. 1. 1. 1. 1. 0. 2. 1.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 2. 1. 1. 1. 2. 1. 1. 0. 1. 0. 2. 2. 2. 0.
 1. 0. 0. 2. 1. 1. 2. 0. 1. 2. 1. 0. 0. 2. 0. 1. 1. 2. 1. 1. 2. 1. 2. 1.
 0. 0. 0. 0. 1. 0. 2. 1. 0. 1. 1. 0. 0. 1. 2. 1. 1. 2. 2. 1. 0. 2. 1. 1.
 0. 0. 0. 1. 0. 0. 0. 1. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 2. 1. 1. 1. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 1. 0. 0. 0. 0. 0. 2. 2. 1. 0. 2. 0. 1. 0. 2. 0.
 2. 1. 2. 0. 1. 0. 1. 0. 0. 0. 2. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1.
```

Resultados de RFC

```
1 print("Accuracy para random_forest 80-20", accuracy_score(est1RFC.predict(X_test20), y_test20))
2 print("Accuracy para random_forest 70-30", accuracy_score(est2RFC.predict(X_test30), y_test30))
3 print("Accuracy para random_forest 60-40", accuracy_score(est3RFC.predict(X_test40), y_test40))
```

```
Accuracy para random_forest 80-20 0.9968553459119497
Accuracy para random_forest 70-30 0.9905660377358491
Accuracy para random_forest 60-40 0.9952830188679245
```

Con lo anterior podemos decir que este método no es recomendable usarlo con este dataset ya que los porcentajes de accuracy no son confiables, se varió el n_estimators pero despues del 4 siempre daba 1.0. Así que variando este tampoco mejora.

- Cross validation con 10 iteraciones y para 3 métricas distintas: accuracy, tpr y tnr. Esto se realizó para los 3 particionamientos usados en RFC

```
1 s = cross_val_score(est1RFC, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
2 print("accuracy %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
3 s = cross_val_score(est1RFC, X, y, cv=KFold(10, shuffle=True), scoring=tpr)
4 print("tpr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
5 s = cross_val_score(est1RFC, X, y, cv=KFold(10, shuffle=True), scoring=tnr)
6 print("tnr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
```

```
accuracy 0.997 (+/- 0.00432)
tpr      0.995 (+/- 0.00977)
tnr      0.998 (+/- 0.00577)
```

- Support Vector Machine:

Para este método se usó el particionamiento 70-30 y se varió el kernel en tres casos: linear, poly y rbf

Support vector machine (SVM)

```
[ ]  1 #SVM con una partición de 70-30
     2 est1SVM = SVC(kernel='linear')
     3 est1SVM.fit(X_train30,y_train30)
     4 print("accurracy para SVM con kernel linear =",accuracy_score(est1SVM.predict(X_test30), y_test30))
```

```
accurracy para SVM con kernel linear = 0.8679245283018868
```

```
     1 y_pred30SVM1 = est1SVM.predict(X_test30)
     2 print("predicciones = ")
     3 print(y_pred30SVM1)
     4 print("ground truth")
     5 print(y_test30)
```

```
predicciones =
[2. 1. 1. 1. 0. 2. 0. 0. 0. 0. 2. 2. 2. 2. 0. 0. 0. 0. 2. 2. 0. 1. 1. 0.
 2. 2. 2. 0. 0. 1. 1. 0. 2. 2. 0. 2. 1. 2. 1. 1. 0. 0. 0. 2. 1. 0. 1. 0.
 2. 1. 0. 1. 0. 0. 2. 2. 1. 1. 2. 1. 0. 0. 0. 0. 2. 1. 2. 2. 2. 2. 2. 0.
 2. 1. 2. 0. 1. 0. 2. 0. 2. 2. 1. 2. 2. 1. 2. 0. 0. 0. 0. 1. 2. 0. 0. 1.
 2. 0. 0. 0. 0. 1. 2. 0. 0. 0. 1. 1. 2. 0. 0. 1. 0. 1. 2. 1. 1. 0. 2. 1.
 1. 2. 1. 1. 0. 0. 0. 0. 0. 2. 2. 1. 1. 2. 0. 1. 2. 0. 1. 0. 2. 2. 2. 0.
 0. 0. 0. 2. 1. 1. 2. 0. 1. 0. 1. 0. 1. 2. 0. 1. 1. 2. 1. 2. 2. 1. 0. 2.
 0. 0. 0. 0. 2. 0. 2. 1. 0. 1. 1. 0. 0. 2. 2. 1. 2. 2. 2. 1. 0. 2. 1. 2.
 0. 0. 0. 1. 0. 0. 0. 2. 0. 2. 1. 0. 2. 0. 1. 0. 0. 0. 0. 2. 0. 1. 1. 1.
 0. 1. 1. 1. 2. 1. 2. 0. 1. 0. 0. 0. 0. 0. 2. 2. 1. 0. 2. 0. 1. 2. 2. 0.
 2. 0. 2. 0. 1. 2. 1. 0. 0. 0. 2. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1.
 0. 2. 0. 2. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 2. 1. 2. 1. 2. 0. 2. 1.
 2. 1. 0. 1. 1. 0. 1. 0. 0. 0. 2. 0. 2. 1. 1. 0. 2. 2. 1. 2. 0. 2. 0. 1.
 0. 0. 2. 1. 0. 2.]
ground truth
```

0s completed at 9:01 AM

- Cross validation con 10 iteraciones y para 3 métricas distintas: accuracy, tpr y tnr. Esto se realizó para los 3 casos de variación de kernel usados en SVM

```
[ ]  1 s = cross_val_score(est1SVM, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
     2 print("accuracy %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     3 s = cross_val_score(est1SVM, X, y, cv=KFold(10, shuffle=True), scoring=tpr)
     4 print("tpr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     5 s = cross_val_score(est1SVM, X, y, cv=KFold(10, shuffle=True), scoring=tnr)
     6 print("tnr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))

accuracy 0.873 (+/- 0.04069)
tpr      0.853 (+/- 0.05970)
tnr      0.930 (+/- 0.02659)
```

```
[ ]  1 s = cross_val_score(est2SVM, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
     2 print("accuracy %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     3 s = cross_val_score(est2SVM, X, y, cv=KFold(10, shuffle=True), scoring=tpr)
     4 print("tpr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     5 s = cross_val_score(est2SVM, X, y, cv=KFold(10, shuffle=True), scoring=tnr)
     6 print("tnr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))

accuracy 0.514 (+/- 0.05024)
tpr      0.922 (+/- 0.04538)
tnr      0.463 (+/- 0.07629)
```

```
     1 s = cross_val_score(est3SVM, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
     2 print("accuracy %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     3 s = cross_val_score(est3SVM, X, y, cv=KFold(10, shuffle=True), scoring=tpr)
     4 print("tpr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
     5 s = cross_val_score(est3SVM, X, y, cv=KFold(10, shuffle=True), scoring=tnr)
     6 print("tnr      %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))

accuracy 0.528 (+/- 0.03558)
tpr      1.000 (+/- 0.00000)
tnr      0.438 (+/- 0.04507)
```

✓ 0s completed at 9:01 AM

Redes Neuronales

Se hicieron 3 redes neuronales variando la cantidad de epochs (20, 50, 100 y 200 epochs) para cada una de las 3 particiones utilizadas(80-20,70-30,60-40)

▾ Particion 80-20

```
[ ]
from tensorflow.keras.optimizers import Adam

nclasses = len(np.unique(y))

epochs = [20,50,100,200]

for epoch in epochs:
  modelo = tf.keras.Sequential([
      tf.keras.layers.Flatten(input_shape=[X_train20.shape[1],]),
      tf.keras.layers.Dense(512, activation=tf.nn.relu),
      tf.keras.layers.Dense(256, activation=tf.nn.relu),
      tf.keras.layers.Dense(128, activation=tf.nn.relu),
      tf.keras.layers.Dense(128, activation=tf.nn.relu),
      tf.keras.layers.Dense(64, activation=tf.nn.relu),
      tf.keras.layers.Dense(nclasses, activation=tf.nn.softmax)
      ])
  modelo.compile(optimizer=Adam(learning_rate=0.001),
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])
  historial = modelo.fit(X_train20, y_train20, epochs=epoch, batch_size=32,verbose=0)

  scores = modelo.evaluate(X_test20, y_test20,verbose=0)
  print(f"modelo entrenado con {epoch} epochs tiene estas metricas: loss={scores[0]} accuracy={scores[1]}")
```

```
modelo entrenado con 20 epochs tiene estas metricas: loss=0.8524550199508667 accuracy=0.5613207817077637
modelo entrenado con 50 epochs tiene estas metricas: loss=0.48103073239326477 accuracy=0.8537735939025879
modelo entrenado con 100 epochs tiene estas metricas: loss=0.22156760096549988 accuracy=0.900943398475647
modelo entrenado con 200 epochs tiene estas metricas: loss=0.17029215395450592 accuracy=0.9339622855186462
```

Resultados de las redes neuronales

Inicialmente se utilizó el optimizador SGD, pero no se obtuvieron grandes resultados, entonces se optó por probar con el optimizador Adam, con el cual se lograron resultados mucho mejores en comparación

SGD

- accuracy: 0.4575	- accuracy: 0.4372	- accuracy: 0.3738
- accuracy: 0.4561	- accuracy: 0.4656	- accuracy: 0.4022
- accuracy: 0.4737	- accuracy: 0.4629	- accuracy: 0.4278
- accuracy: 0.4858	- accuracy: 0.4669	- accuracy: 0.4170
- accuracy: 0.4602	- accuracy: 0.4588	- accuracy: 0.4035
- accuracy: 0.4561	- accuracy: 0.4467	- accuracy: 0.4211
- accuracy: 0.4926	- accuracy: 0.4224	- accuracy: 0.4615
- accuracy: 0.4399	- accuracy: 0.4885	- accuracy: 0.4413
- accuracy: 0.4467	- accuracy: 0.4521	- accuracy: 0.4197
- accuracy: 0.4696	- accuracy: 0.4872	- accuracy: 0.4345

Adam

```
loss=0.8524550199508667 accuracy=0.5613207817077637
loss=0.48103073239326477 accuracy=0.8537735939025879
loss=0.22156760096549988 accuracy=0.900943398475647
loss=0.17029215395450592 accuracy=0.9339622855186462
```

¡GRACIAS!