Clasificación Ejercicio 2

```python
# Importar las librerías necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.multiclass import OneVsRestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import SelectKBest, f_classif, SequentialFeatureSelector
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, cross_val_predict
from sklearn.metrics import recall_score, accuracy_score, classification_report
from sklearn.utils.class_weight import compute_class_weight
```

```python
df = pd.read_csv('datos_ej2.txt', delimiter='\t', header=None)
df = df.drop(columns=[632])
df_clean = df.dropna()
data = df_clean.to_numpy()

# Separar las columnas: la primera es la clase, la segunda se ignora
clases = data[:, 0]
variables = data[:, 2:]
```

```
print(clases)
print(variables)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.
 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.
 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.
 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.
 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5.
 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5.
 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5.
 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 5. 6. 6. 6. 6. 6. 6.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.
 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.
 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.
 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.
 7. 7. 7. 7. 7. 7.]
[[ 1.37948727  0.89411356  0.59709367 ... -0.4271621   1.54580044
   1.94779799]
 [-0.08642644  0.05872434 -0.36184551 ... -0.45192356  0.07679225
  -0.18774108]
 [ 0.12087848  0.27917388 -0.0760135  ... -0.61593363  0.35172315
  -0.73744005]
 ...
 [-6.60980215 -5.10891288 -2.90782293 ... -1.35575065 -3.99313729
  -5.95784039]
 [-6.37641239 -5.97726837 -3.83217252 ... -1.77263488 -4.03853368
  -5.92188239]
 [-5.58714073 -5.65547073 -2.82554096 ... -1.00629159 -4.40704293
  -6.03750229]]
```

Vamos a checar si nuestos datos están balanceados o no

```
[ ]  unique_classes = np.unique(clases)

     # Calculate total count of instances
     total = len(clases)

     # Calculate and print the count and percentage for each class
     for cls in unique_classes:
         count = np.sum(clases == cls)
         percentage = (count * 100 / total) if total > 0 else 0
         print(f'Clase {cls}: {count} ({percentage:.2f}%)')
```

```
Clase 1.0: 90 (14.29%)
Clase 2.0: 90 (14.29%)
Clase 3.0: 90 (14.29%)
Clase 4.0: 90 (14.29%)
Clase 5.0: 90 (14.29%)
Clase 6.0: 90 (14.29%)
Clase 7.0: 90 (14.29%)
```

Cross Validation

```
[ ]  def CV_Standard(variables, classes, clf, kf, model):
         cv_y_test = []
         cv_y_pred = []

         for train_index, test_index in kf.split(variables, classes):
             # Training phase
             x_train = variables[train_index, :]
             y_train = classes[train_index]

             clf.fit(x_train, y_train)

             # Test phase
             x_test = variables[test_index, :]
             y_test = classes[test_index]
             y_pred = clf.predict(x_test)

             cv_y_test.append(y_test)
             cv_y_pred.append(y_pred)

         # Classification results
         print(f"{model} Classification Report:")
         print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred), zero_division=0))
```

Maquinas de Soporte Vectorial

```
[5]  # Maquinas de Soporte Vectorial Upsampled
     def SVM(variables, classes):
       clf = SVC(kernel='linear')
       kf = StratifiedKFold(n_splits=5, shuffle=True)

       CV_Standard(variables, classes, clf, kf, 'SVM')

     SVM(variables, clases)
```

```
SVM Classification Report:
              precision    recall  f1-score   support

         1.0       0.98      0.96      0.97        90
         2.0       0.80      0.86      0.83        90
         3.0       0.93      0.87      0.90        90
         4.0       0.92      0.94      0.93        90
         5.0       0.91      0.92      0.92        90
         6.0       0.98      0.90      0.94        90
         7.0       0.90      0.96      0.92        90

    accuracy                           0.91       630
   macro avg       0.92      0.91      0.91       630
weighted avg       0.92      0.91      0.91       630
```

K-Vecinos

```
[6]  # K-Vecinos
     def KNN(variables, classes, n_neighbors=5):
         clf = KNeighborsClassifier(n_neighbors=n_neighbors)
         kf = StratifiedKFold(n_splits=5, shuffle=True)

         CV_Standard(variables, classes, clf, kf, 'KNN')

     KNN(variables, clases)
```

```
KNN Classification Report:
              precision    recall  f1-score   support

         1.0       0.90      0.93      0.92        90
         2.0       0.68      0.84      0.76        90
         3.0       0.92      0.86      0.89        90
         4.0       0.86      0.72      0.78        90
         5.0       0.91      0.87      0.89        90
         6.0       0.93      0.82      0.87        90
         7.0       0.88      0.98      0.93        90

    accuracy                           0.86       630
   macro avg       0.87      0.86      0.86       630
weighted avg       0.87      0.86      0.86       630
```

Arboles de Decisión

```
[7]  # Decision Tree Upsampled
     def DecisionTree(variables, classes):
         clf = DecisionTreeClassifier()
         kf = StratifiedKFold(n_splits=5, shuffle=True)

         CV_Standard(variables, classes, clf, kf, 'Decision Tree')

     DecisionTree(variables, clases)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

         1.0       0.76      0.76      0.76        90
         2.0       0.46      0.49      0.47        90
         3.0       0.69      0.76      0.72        90
         4.0       0.60      0.61      0.60        90
         5.0       0.67      0.60      0.63        90
         6.0       0.69      0.66      0.67        90
         7.0       0.89      0.88      0.88        90

    accuracy                           0.68       630
   macro avg       0.68      0.68      0.68       630
weighted avg       0.68      0.68      0.68       630
```

Discriminante Lineal

```
[8]  # Discriminante Lineal
     def LDA(variables, classes):
         clf = LinearDiscriminantAnalysis()
         kf = StratifiedKFold(n_splits=5, shuffle=True)

         CV_Standard(variables, classes, clf, kf, 'Discriminante Lineal')

     LDA(variables, clases)
```

```
Discriminante Lineal Classification Report:
              precision    recall  f1-score   support

         1.0       0.81      0.77      0.79        90
         2.0       0.48      0.53      0.51        90
         3.0       0.62      0.62      0.62        90
         4.0       0.54      0.59      0.56        90
         5.0       0.65      0.64      0.65        90
         6.0       0.65      0.62      0.64        90
         7.0       0.90      0.80      0.85        90

    accuracy                           0.65       630
   macro avg       0.66      0.65      0.66       630
weighted avg       0.66      0.65      0.66       630
```

Discriminante Multiclase

```
[9]  # Discriminante Multiclase
     def Multiclase(variables, classes):
         clf = OneVsRestClassifier(SVC(kernel='linear'))
         kf = StratifiedKFold(n_splits=5, shuffle=True)

         CV_Standard(variables, classes, clf, kf, 'Multiclase')

     Multiclase(variables, clases)
```

```
Multiclase Classification Report:
              precision    recall  f1-score   support

         1.0       0.94      0.97      0.95        90
         2.0       0.78      0.79      0.78        90
         3.0       0.91      0.87      0.89        90
         4.0       0.90      0.94      0.92        90
         5.0       0.94      0.90      0.92        90
         6.0       0.91      0.89      0.90        90
         7.0       0.89      0.91      0.90        90

    accuracy                           0.90       630
   macro avg       0.90      0.90      0.90       630
weighted avg       0.90      0.90      0.90       630
```

Discriminante Cuadrático

```python
# Discriminante Cuadrático
def QDA(variables, classes):
    clf = QuadraticDiscriminantAnalysis()
    kf = StratifiedKFold(n_splits=5, shuffle=True)

    CV_Standard(variables, classes, clf, kf, 'Discriminante Cuadrático')

QDA(variables, clases)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Discriminante Cuadrático Classification Report:
              precision    recall  f1-score   support

         1.0       0.19      0.16      0.17        90
         2.0       0.21      0.16      0.18        90
         3.0       0.21      0.26      0.23        90
         4.0       0.16      0.12      0.14        90
         5.0       0.16      0.17      0.16        90
         6.0       0.18      0.19      0.19        90
         7.0       0.27      0.38      0.32        90

    accuracy                           0.20       630
   macro avg       0.20      0.20      0.20       630
weighted avg       0.20      0.20      0.20       630

/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

```python
# Regresión Logística
from sklearn.linear_model import LogisticRegression

def SKLogisticRegression(variables, classes):
    clf = LogisticRegression(class_weight='balanced')
    kf = StratifiedKFold(n_splits=5, shuffle=True)

    CV_Standard(variables, classes, clf, kf, 'Regresión Logística')

SKLogisticRegression(variables, clases)
```

Regresión Logística Classification Report:

	precision	recall	f1-score	support
1.0	0.97	0.97	0.97	90
2.0	0.82	0.89	0.85	90
3.0	0.95	0.86	0.90	90
4.0	0.89	0.93	0.91	90
5.0	0.94	0.91	0.93	90
6.0	0.93	0.90	0.92	90
7.0	0.89	0.92	0.91	90
accuracy			0.91	630
macro avg	0.91	0.91	0.91	630
weighted avg	0.91	0.91	0.91	630

```python
def Hiperparametros(x, y):


    print("----- Model evaluation -----")
    kf = StratifiedKFold(n_splits=5, shuffle = True)

    cv_y_test = []
    cv_y_pred = []

    for train_index, test_index in kf.split(x, y):

        x_train = x[train_index, :]
        y_train = y[train_index]

        x_test = x[test_index, :]
        y_test = y[test_index]

        parameters = {'n_neighbors': np.arange(1, 100)}
        clf_cv = GridSearchCV(KNeighborsClassifier(), parameters, cv = 5)
        clf_cv.fit(x_train, y_train)

        y_pred = clf_cv.predict(x_test)

        cv_y_test.append(y_test)
        cv_y_pred.append(y_pred)

    print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred)))


    print("----- Model evaluation with cross_val_predict -----")

    clf = GridSearchCV(KNeighborsClassifier(), {'n_neighbors': np.arange(1, 100)}, cv = 5)
    y_pred = cross_val_predict(clf, x, y, cv = 5)

    print(classification_report(y, y_pred))


    print("----- Production model -----")

    clf = GridSearchCV(KNeighborsClassifier(), {'n_neighbors': np.arange(1, 100)}, cv = 5)
    clf.fit(x, y)

    print(clf.best_estimator_)


Hiperparametros(variables, clases)
```

```
----- Model evaluation -----
              precision    recall  f1-score   support

         1.0       0.88      0.93      0.90        90
         2.0       0.73      0.82      0.77        90
         3.0       0.96      0.84      0.90        90
         4.0       0.85      0.80      0.82        90
         5.0       0.92      0.87      0.89        90
         6.0       0.95      0.87      0.91        90
         7.0       0.88      1.00      0.94        90

    accuracy                           0.88       630
   macro avg       0.88      0.88      0.88       630
weighted avg       0.88      0.88      0.88       630

----- Model evaluation with cross_val_predict -----
              precision    recall  f1-score   support

         1.0       0.93      0.91      0.92        90
         2.0       0.71      0.81      0.76        90
         3.0       0.90      0.84      0.87        90
         4.0       0.84      0.80      0.82        90
         5.0       0.92      0.86      0.89        90
         6.0       0.90      0.83      0.87        90
         7.0       0.88      1.00      0.94        90

    accuracy                           0.87       630
   macro avg       0.87      0.87      0.87       630
weighted avg       0.87      0.87      0.87       630

----- Production model -----
KNeighborsClassifier(n_neighbors=13)
```

1.- ¿Observas un problema en cuanto al balanceo de las clases? ¿Por qué?

Los datos se encuentran distribuidos equitativamente por lo que están muy bien balanceados.

2.- ¿Qué modelo o modelos fueron efectivos para clasificar tus datos? ¿Observas algo especial sobre los modelos?

En primer lugar estan las SVMs, con 0.92 y en segundo lugar el clasificador multiclase con 0.9. Ambos modelos son bastante flexibles ante el numero de clases lo cual los hace los mejores candidatos.

3.- ¿Observas alguna mejora importante al optimizar hiperparámetros? ¿Es el resultado que esperabas?

Si esperaba una mejora, sin embargo no dió el resultado que esperaba. No se si sea por la colinealidad de los datos, o el modelo para el cual optimicé los hiperparametros

4.- ¿Qué inconvenientes hay al encontrar hiperparámetros? ¿Por qué?

Tarda demasiado tiempo, secuencial duró más de 250 minutos procesando.