problemas-de-clasificacion

September 9, 2023

1 Problemas de Clasificación

Ernesto Reynoso Lizárraga A01639915

[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold, train_test_split,␣
↪cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SelectKBest,␣
↪SequentialFeatureSelector,RFE, f_classif
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, accuracy_score
import random
```

##Problema 1

[179]:
```
df = np.loadtxt("/content/drive/MyDrive/Inteligencia Artificial/P1_3.txt")
```

[180]:
```
x = df[:,2:]
x
```

[180]:
```
array([[ 0.3925073 ,  0.67657019,  0.60180412, …,  1.24975793,
          1.03738802,  1.05531121],
        [-1.31487611, -0.73287418,  0.41422541, …, -0.61989557,
         -1.05137325, -1.19338103],
        [-1.09345032, -0.68931183,  0.07082691, …, -0.29226011,
         -0.2000579 ,  0.28090627],
        …,
        [-0.72853565, -0.78422092,  0.02350863, …, -0.00920863,
          0.12140923,  0.39523656],
        [ 1.77147543,  0.83735529,  0.18184615, …, -0.45705499,
         -1.52412392, -1.73872657],
        [ 0.47996947, -0.54432989, -0.75249618, …, -0.18374824,
```

```
         -0.69901401, -1.41618733]])
```

[181]:
```
y = df[:,0]
y
```

[181]:
```
array([1., 1., 1., …, 2., 2., 2.])
```

###Determina si es necesario balancear los datos. En caso de que sea afirmativo, en todo este ejercicio tendrás que utilizar alguna estrategia para mitigar el problema de tener una muestra desbalanceada.

realizando diferentes estrategias para balancear los datos y comparandolos con el modelo de datos desvalanceados, podemos concluir que no hace falta balancear los datos

###Evalúa al menos 5 modelos de clasificación distintos utilizando validación cruzada, y determina cuál de ellos es el más efectivo.

[182]:
```
kf = StratifiedKFold(n_splits=5, shuffle = True)
clf = SVC(kernel = 'linear')
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(x, y):
  # Training phase
  x_train = x[train_index, :]
  y_train = y[train_index]
  clf.fit(x_train, y_train)
  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
  y_pred = clf.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)))
```

```
              precision    recall  f1-score   support

         1.0       0.73      0.66      0.69       298
         2.0       0.93      0.95      0.94      1496

    accuracy                           0.90      1794
   macro avg       0.83      0.80      0.82      1794
weighted avg       0.90      0.90      0.90      1794
```

[183]:
```
# RBF SVM
print('----- RBF-SVM -----')
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]

  clf = SVC(kernel = 'rbf')
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- RBF-SVM -----
              precision    recall  f1-score   support

         1.0       0.87      0.58      0.70       298
         2.0       0.92      0.98      0.95      1496

    accuracy                           0.92      1794
   macro avg       0.90      0.78      0.83      1794
weighted avg       0.91      0.92      0.91      1794
```

[184]:

```
# KNN
print('----- KNN -----')
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]

  clf = KNeighborsClassifier(n_neighbors=3)
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- KNN -----
```

```
              precision    recall  f1-score   support

         1.0       0.66      0.42      0.51       298
         2.0       0.89      0.96      0.92      1496

    accuracy                           0.87      1794
   macro avg       0.78      0.69      0.72      1794
weighted avg       0.85      0.87      0.86      1794
```

[185]:
```
# Decision tree
print('----- Decision tree -----')
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]
  clf = DecisionTreeClassifier()
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- Decision tree -----
              precision    recall  f1-score   support

         1.0       0.46      0.45      0.45       298
         2.0       0.89      0.89      0.89      1496

    accuracy                           0.82      1794
   macro avg       0.67      0.67      0.67      1794
weighted avg       0.82      0.82      0.82      1794
```

[186]:
```
# Linear Discriminant Analysis
print('----- Linear Discriminant Analysis -----')
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]
  clf = LinearDiscriminantAnalysis()
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- Linear Discriminant Analysis -----
              precision    recall  f1-score   support

         1.0       0.70      0.65      0.68       298
         2.0       0.93      0.95      0.94      1496

    accuracy                           0.90      1794
   macro avg       0.82      0.80      0.81      1794
weighted avg       0.89      0.90      0.90      1794
```

Comparando los 5 modelos, podemos determinar que le mejor modelo para estos datos es el modelo de clasificación lineal.

1.0.1 Implementa desde cero el método de regresión logística, y evalúalo con el conjunto de datos

[187]:
```
def gradient(X, y, beta):
    xbeta = X @ beta
    c0 = (y == 0)
    c1 = (y == 1)

    exp0 = np.exp(xbeta[c0])
    l0 = (exp0 / (1 + exp0)) * X[c0, :].transpose()

    exp1 = np.exp(xbeta[c1])
    l1 = (exp1 / (1 + exp1)) * X[c1, :].transpose()

    return l0.sum(axis=1) - l1.sum(axis=1)


def fit_model(X, y, alpha=0.0005, max_iterations=100000):
  npredictors = X.shape[1]
  beta = 2 * np.random.rand(npredictors) - 1.0
  it = 0

  while (np.linalg.norm(gradient(X, y, beta)) > 1e-4) and (it < max_iterations):
      beta = beta - alpha * gradient(X, y, beta)
```

```
        it = it + 1

    return beta

def predict(X, beta):
        xbeta = X @ beta
        tmp = 1. / (1. + np.exp(-xbeta))
        return (tmp > 0.5).astype("int32")
```

[188]:
```
# Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)

# Normaliza las características utilizando StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[189]:
```
beta = fit_model(X_train, y_train, max_iterations=100)
y_pred = predict(X_test, beta)
accuracy = np.mean(y_test == y_pred)
print('Precision:',accuracy * 100, '%')
print(classification_report(y_test, y_pred, zero_division=1))
```

```
Precision: 13.649025069637883 %
              precision    recall  f1-score   support

         0.0       0.00      1.00      0.00         0
         1.0       0.32      0.94      0.47        52
         2.0       1.00      0.00      0.00       307

    accuracy                           0.14       359
   macro avg       0.44      0.65      0.16       359
weighted avg       0.90      0.14      0.07       359
```

1.0.2 Con alguno de los clasificadores que probaste en los pasos anteriores, determina el número óptimo de características utilizando un método tipo Filter.

[190]:
```
# Find optimal number of features using cross-validation
################################################################################
print("----- Optimal selection of number of features -----")
n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
acc_nfeat = []
for n_feat in n_feats:
    print('---- n features =', n_feat)
    acc_cv = []
```

```python
  kf = StratifiedKFold(n_splits=5, shuffle = True)
  for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(kernel = 'linear')
    fselection_cv = SelectKBest(f_classif, k = n_feat)
    fselection_cv.fit(x_train, y_train)
    x_train = fselection_cv.transform(x_train)
    clf_cv.fit(x_train, y_train)

    # Test phase
    x_test = fselection_cv.transform(x[test_index, :])
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc = np.average(acc_cv)
  acc_nfeat.append(acc)
  print('ACC:', acc)

opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)
```

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8338914738332738
---- n features = 2
ACC: 0.8338914738332738
---- n features = 3
ACC: 0.8338914738332738
---- n features = 4
ACC: 0.8389054014098754
---- n features = 5
ACC: 0.8444826566657848
---- n features = 6
ACC: 0.8606526509080158
---- n features = 7
ACC: 0.8612081978182724
---- n features = 8
ACC: 0.8656759154074789
---- n features = 9
ACC: 0.8695662999330853
---- n features = 10
ACC: 0.8768133082273852
```

```
---- n features = 11
ACC: 0.8796143850858218
---- n features = 12
ACC: 0.8796050481629605
---- n features = 13
ACC: 0.8868489441496397
Optimal number of features:  13
```

[191]:

```python
# Find optimal number of features using cross-validation
#################################################################################
print("----- Optimal selection of number of features -----")
n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9]
acc_nfeat = []
for n_feat in n_feats:
  print('---- n features =', n_feat)
  acc_cv = []
  kf = StratifiedKFold(n_splits=5, shuffle = True)
  for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(kernel = 'linear')
    fselection_cv = SequentialFeatureSelector(clf_cv,
 ↪n_features_to_select=n_feat)
    fselection_cv.fit(x_train, y_train)
    x_train = fselection_cv.transform(x_train)
    clf_cv.fit(x_train, y_train)
    # Test phase
    x_test = fselection_cv.transform(x[test_index, :])
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc = np.average(acc_cv)
  acc_nfeat.append(acc)
  print('ACC:', acc)
opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)
```

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8338914738332738
---- n features = 2
ACC: 0.8338914738332738
---- n features = 3
ACC: 0.8338914738332738
```

```
---- n features = 4
ACC: 0.8338914738332738
---- n features = 5
ACC: 0.8338914738332738
---- n features = 6
ACC: 0.8338914738332738
---- n features = 7
ACC: 0.8338914738332738
---- n features = 8
ACC: 0.8338914738332738
---- n features = 9
ACC: 0.8338914738332738
Optimal number of features:  1
```

[195]:

```python
# Find optimal number of features using cross-validation
#################################################################################
print("----- Optimal selection of number of features -----")
n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
acc_nfeat = []
for n_feat in n_feats:
  print('---- n features =', n_feat)
  acc_cv = []
  kf = StratifiedKFold(n_splits=5, shuffle = True)
  for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(kernel = 'linear')
    fselection_cv = RFE(clf_cv, n_features_to_select=n_feat)
    fselection_cv.fit(x_train, y_train)
    x_train = fselection_cv.transform(x_train)
    clf_cv.fit(x_train, y_train)

    # Test phase
    x_test = fselection_cv.transform(x[test_index, :])
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc = np.average(acc_cv)
  acc_nfeat.append(acc)
  print('ACC:', acc)
opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)
```

```
----- Optimal selection of number of features -----
```

```
---- n features = 1
ACC: 0.8338914738332738
---- n features = 2
ACC: 0.8355627830254744
---- n features = 3
ACC: 0.8623161793311651
---- n features = 4
ACC: 0.8623255162540266
---- n features = 5
ACC: 0.8757037705606823
---- n features = 6
ACC: 0.8756913213302002
---- n features = 7
ACC: 0.8795941550862887
---- n features = 8
ACC: 0.8918706524952926
---- n features = 9
ACC: 0.8907408848290566
---- n features = 10
ACC: 0.8896422402390252
---- n features = 11
ACC: 0.893527956303201
---- n features = 12
ACC: 0.892981746315806
---- n features = 13
ACC: 0.8873858172141734
Optimal number of features:  11
```

1.0.3 Escoge alguna de las técnicas de selección de características que probaste con anteioridad, y con el número óptimo de características encontrado, prepara tu modelo para producción haciendo lo siguiente:

Aplica el método de selección de características con todos los datos.

Ajusta el modelo con las características encontradas.

Filter

[196]:
```
# Find optimal number of features using cross-validation
###############################################################################
print("----- Optimal selection of number of features -----")
n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
acc_nfeat = []
for n_feat in n_feats:
  print('---- n features =', n_feat)
  acc_cv = []
  kf = StratifiedKFold(n_splits=5, shuffle = True)
  for train_index, test_index in kf.split(x, y):
    # Training phase
```

```python
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(kernel = 'linear')
    fselection_cv = SelectKBest(f_classif, k = n_feat)
    fselection_cv.fit(x_train, y_train)
    x_train = fselection_cv.transform(x_train)
    clf_cv.fit(x_train, y_train)

    # Test phase
    x_test = fselection_cv.transform(x[test_index, :])
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc = np.average(acc_cv)
  acc_nfeat.append(acc)
  print('ACC:', acc)

opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)
```

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8338914738332738
---- n features = 2
ACC: 0.8338914738332738
---- n features = 3
ACC: 0.8338914738332738
---- n features = 4
ACC: 0.8378020883584135
---- n features = 5
ACC: 0.8455999751015391
---- n features = 6
ACC: 0.8600893232287079
---- n features = 7
ACC: 0.8600908793825182
---- n features = 8
ACC: 0.8612019732030314
---- n features = 9
ACC: 0.8756944336378207
---- n features = 10
ACC: 0.8779150651250369
---- n features = 11
ACC: 0.8795925989324784
---- n features = 12
```

```
ACC: 0.881835016573038
---- n features = 13
ACC: 0.8874247210594295
Optimal number of features:  13
```

[198]:
```python
# Fit model with optimal number of features
clf = SVC(kernel = 'linear')
fselection = SelectKBest(f_classif, k = opt_features)
fselection.fit(x, y)
print("Selected features: ", fselection.get_feature_names_out())
x_transformed = fselection.transform(x)
clf.fit(x_transformed, y)
y_pred = clf.predict(x_transformed)
print(classification_report(y, y_pred))
```

```
Selected features:  ['x11' 'x12' 'x16' 'x17' 'x18' 'x19' 'x20' 'x21' 'x27' 'x28'
'x29' 'x64'
 'x65']
              precision    recall  f1-score   support

         1.0       0.73      0.47      0.57       298
         2.0       0.90      0.96      0.93      1496

    accuracy                           0.88      1794
   macro avg       0.81      0.72      0.75      1794
weighted avg       0.87      0.88      0.87      1794
```

1.0.4 Contesta las siguientes preguntas:

¿Qué pasa si no se considera el problema de tener datos desbalanceados para este caso? ¿Por qué?

Si no se considera ese caso, el modelo prodria optar por predecir la clase mayoritaria ya que tendria mas posibilidades de estar correcto en vez de realizar un correcto analisis.

De todos los clasificadores, ¿cuál o cuales consideras que son adecuados para los datos? ¿Qué propiedades tienen dichos modelos que los hacen apropiados para los datos? Argumenta tu respuesta.

Considero que el metodo de clasificación lineal es el mas adecuado para los datos debido que en la mayoria de los casos, el puntaje f1 es mayor en ambas clases comparandolo con los demas metodos

¿Es posibles reducir la dimensionalidad del problema sin perder rendimiento en el modelo? ¿Por qué?

En este caso no es posible reducir la dimensionalidad del problema debido a que las pruebas nos muestran que al intentar ajustar el modelo con ciertas caracteristicas se pierde rendimiento significante en el caso de la clase 1.

¿Qué método de selección de características consideras el más adecuado para este caso? ¿Por qué?

El metodo mas adecuado es Filter debido a que nos arroja el mejor resultado (y sobretodo en mucho menor tiempo)

Si quisieras mejorar el rendimiento de tus modelos, ¿qué más se podría hacer?

Se podria ajustar los hiperparametros o incluso probar diferentes algoritmos de clasificación para buscar uno que se adapte mejor a estos datos

1.1 Problema 2

###Determina si es necesario balancear los datos. En caso de que sea afirmativo, en todo este ejercicio tendrás que utilizar alguna estrategia para mitigar el problema de tener una muestra desbalanceada.

[26]:
```
df2 = np.loadtxt("/content/drive/MyDrive/Inteligencia Artificial/M_1.txt")
```

[27]:
```
x = df2[:,2:]
```

[28]:
```
y = df2[:,0]
```

[29]:
```
kf = StratifiedKFold(n_splits=5, shuffle = True)
clf = SVC(kernel = 'linear')
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(x, y):
  # Training phase
  x_train = x[train_index, :]
  y_train = y[train_index]
  clf.fit(x_train, y_train)

  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
  y_pred = clf.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)))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.99      0.99        90
         2.0       0.93      0.97      0.95        90
         3.0       0.97      0.93      0.95        90
         4.0       1.00      0.99      0.99        90
```

```
         5.0       0.99      0.98      0.98        90
         6.0       0.90      0.90      0.90        90
         7.0       0.99      1.00      0.99        90

    accuracy                           0.97       630
   macro avg       0.97      0.97      0.97       630
weighted avg       0.97      0.97      0.97       630
```

[30]:
```
print("----- Subsamplig -----")
clf = SVC(kernel = 'linear')
kf = StratifiedKFold(n_splits=5, shuffle = True)
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(x, y):
  # Training phase
  x_train = x[train_index, :]
  y_train = y[train_index]
  x1 = x_train[y_train==1, :]
  y1 = y_train[y_train==1]
  n1 = len(y1)
  x2 = x_train[y_train==2, :]
  y2 = y_train[y_train==2]
  n2 = len(y2)
  ind = random.sample([i for i in range(n2)], n1)
  x_sub = np.concatenate((x1, x2[ind,:]), axis=0)
  y_sub = np.concatenate((y1, y2[ind]), axis=0)
  clf.fit(x_sub, y_sub)
  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
  y_pred = clf.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)))
```

```
----- Subsamplig -----
              precision    recall  f1-score   support

         1.0       0.49      1.00      0.66        90
         2.0       0.20      1.00      0.33        90
         3.0       0.00      0.00      0.00        90
         4.0       0.00      0.00      0.00        90
         5.0       0.00      0.00      0.00        90
         6.0       0.00      0.00      0.00        90
         7.0       0.00      0.00      0.00        90
```

```
    accuracy                           0.29       630
   macro avg       0.10      0.29      0.14       630
weighted avg       0.10      0.29      0.14       630


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[31]:
```
print("----- Upsampling -----")
clf = SVC(kernel = 'linear')
kf = StratifiedKFold(n_splits=5, shuffle = True)
cv_y_test = []
cv_y_pred = []

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

  x1 = x_train[y_train==1, :]
  y1 = y_train[y_train==1]
  n1 = len(y1)

  x2 = x_train[y_train==2, :]
  y2 = y_train[y_train==2]
  n2 = len(y2)

  ind = random.choices([i for i in range(n1)], k = n2)
  x_sub = np.concatenate((x1[ind,:], x2), axis=0)
  y_sub = np.concatenate((y1[ind], y2), axis=0)

  clf.fit(x_sub, y_sub)
  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
```

```
  y_pred = clf.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)))
```

```
----- Upsampling -----
              precision    recall  f1-score   support

         1.0       0.50      1.00      0.66        90
         2.0       0.20      1.00      0.33        90
         3.0       0.00      0.00      0.00        90
         4.0       0.00      0.00      0.00        90
         5.0       0.00      0.00      0.00        90
         6.0       0.00      0.00      0.00        90
         7.0       0.00      0.00      0.00        90

    accuracy                           0.29       630
   macro avg       0.10      0.29      0.14       630
weighted avg       0.10      0.29      0.14       630


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[33]:
```
print("----- Weighted loss function -----")
clf = SVC(kernel = 'linear', class_weight='balanced')
kf = StratifiedKFold(n_splits=5, shuffle = True)
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(x, y):
  # Training phase
  x_train = x[train_index, :]
  y_train = y[train_index]
  clf.fit(x_train, y_train)
```

```
  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
  y_pred = clf.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)))
```

```
----- Weighted loss function -----
              precision    recall  f1-score   support

         1.0       1.00      0.97      0.98        90
         2.0       0.94      0.99      0.96        90
         3.0       0.99      0.94      0.97        90
         4.0       1.00      0.99      0.99        90
         5.0       0.97      0.99      0.98        90
         6.0       0.96      0.96      0.96        90
         7.0       0.99      1.00      0.99        90

    accuracy                           0.98       630
   macro avg       0.98      0.98      0.98       630
weighted avg       0.98      0.98      0.98       630
```

1.1.1 Evalúa al menos 5 modelos de clasificación distintos utilizando validación cruzada, y determina cuál de ellos es el más efectivo.

[34]:
```
kf = StratifiedKFold(n_splits=5, shuffle = True)
clf = SVC(kernel = 'linear')
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(x, y):
  # Training phase
  x_train = x[train_index, :]
  y_train = y[train_index]
  clf.fit(x_train, y_train)

  # Test phase
  x_test = x[test_index, :]
  y_test = y[test_index]
  y_pred = clf.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)))
```

```
              precision    recall  f1-score   support

         1.0       1.00      0.99      0.99        90
         2.0       0.92      0.99      0.95        90
         3.0       0.97      0.96      0.96        90
         4.0       1.00      0.99      0.99        90
         5.0       0.99      0.98      0.98        90
         6.0       0.95      0.91      0.93        90
         7.0       0.99      1.00      0.99        90

    accuracy                           0.97       630
   macro avg       0.97      0.97      0.97       630
weighted avg       0.97      0.97      0.97       630
```

[35]:
```
print('----- RBF-SVM -----')
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]
  clf = SVC(kernel = 'rbf')
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- RBF-SVM -----
              precision    recall  f1-score   support

         1.0       0.97      1.00      0.98        90
         2.0       0.93      0.99      0.96        90
         3.0       0.98      0.94      0.96        90
         4.0       1.00      1.00      1.00        90
         5.0       0.99      0.94      0.97        90
         6.0       0.95      0.92      0.94        90
         7.0       0.99      1.00      0.99        90

    accuracy                           0.97       630
   macro avg       0.97      0.97      0.97       630
```

```
weighted avg       0.97      0.97      0.97       630
```

[36]:
```python
print('----- KNN -----')
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]
  clf = KNeighborsClassifier(n_neighbors=3)
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- KNN -----
              precision    recall  f1-score   support

         1.0       0.97      0.99      0.98        90
         2.0       0.88      0.99      0.93        90
         3.0       0.95      0.91      0.93        90
         4.0       1.00      1.00      1.00        90
         5.0       1.00      0.93      0.97        90
         6.0       0.93      0.89      0.91        90
         7.0       0.99      1.00      0.99        90

    accuracy                           0.96       630
   macro avg       0.96      0.96      0.96       630
weighted avg       0.96      0.96      0.96       630
```

[37]:
```python
print('----- Decision tree -----')
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]
  clf = DecisionTreeClassifier()
  clf.fit(x_train, y_train)
```

```
  y_pred = clf.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)))
```

```
----- Decision tree -----
              precision    recall  f1-score   support

         1.0       0.94      0.92      0.93        90
         2.0       0.79      0.79      0.79        90
         3.0       0.86      0.87      0.86        90
         4.0       0.91      0.89      0.90        90
         5.0       0.90      0.88      0.89        90
         6.0       0.77      0.80      0.79        90
         7.0       0.97      0.99      0.98        90

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

[38]:
```
print('----- Linear Discriminant Analysis -----')
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]
  clf = LinearDiscriminantAnalysis()
  clf.fit(x_train, y_train)
  y_pred = clf.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)))
```

```
----- Linear Discriminant Analysis -----
              precision    recall  f1-score   support

         1.0       0.87      0.84      0.86        90
         2.0       0.78      0.84      0.81        90
         3.0       0.74      0.78      0.76        90
         4.0       0.95      0.90      0.93        90
         5.0       0.82      0.83      0.82        90
         6.0       0.66      0.63      0.64        90
```

```
         7.0       1.00      0.97      0.98        90

    accuracy                           0.83       630
   macro avg       0.83      0.83      0.83       630
weighted avg       0.83      0.83      0.83       630
```

1.1.2 Escoge al menos dos clasificadores que hayas evaluado en el paso anterior e identifica sus hiperparámetros. Lleva a cabo el proceso de validación cruzada anidada para evaluar los dos modelos con la selección óptima de hiperparámetros.

SVM

[20]:
```python
print("----- SVM classifier - Regularization parameter -----")
cc = np.logspace(-3, 1, 100)
acc = []
for c in cc:
  acc_cv = []
  kf = StratifiedKFold(n_splits=5, shuffle = True)

  for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(C=c, kernel = 'linear')
    clf_cv.fit(x_train, y_train)

    # Test phase
    x_test = x[test_index, :]
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc_hyp = np.average(acc_cv)
  acc.append(acc_hyp)
opt_index = np.argmax(acc)
opt_hyperparameter_linear = cc[opt_index]
print(f"Best parameter : C = {opt_hyperparameter_linear:.4f}")
print(f"Accuracy = {acc[opt_index]:.4f}")
```

```
----- SVM classifier - Regularization parameter -----
Best parameter : C = 0.0112
Accuracy = 0.9825
```

RB-SVM

[21]:
```
print("----- RB-SVM classifier - Smoothing parameter -----")
gg = np.logspace(-5, -1, 100)
acc = []
for g in gg:
  acc_cv = []
  kf = StratifiedKFold(n_splits=5, shuffle = True)
  for train_index, test_index in kf.split(x, y):
    # Training phase
    x_train = x[train_index, :]
    y_train = y[train_index]
    clf_cv = SVC(kernel ='rbf', gamma = g)
    clf_cv.fit(x_train, y_train)

    # Test phase
    x_test = x[test_index, :]
    y_test = y[test_index]
    y_pred = clf_cv.predict(x_test)
    acc_i = accuracy_score(y_test, y_pred)
    acc_cv.append(acc_i)

  acc_hyp = np.average(acc_cv)
  acc.append(acc_hyp)

opt_index = np.argmax(acc)
opt_hyperparameter_rb = gg[opt_index]
print(f"Best parameter : G = {opt_hyperparameter_rb:.4f}")
print(f"Accuracy = {acc[opt_index]:.4f}")
```

```
----- RB-SVM classifier - Smoothing parameter -----
Best parameter : G = 0.0003
Accuracy = 0.9746
```

###Prepara tus modelos para producción haciendo lo siguiente: Opten los hiperparámetros óptimos utilizando todo el conjunto de datos con validación cruzada.

Con los hiperparámetros óptimos, ajusta el modelo con todos los datos.

[264]:
```
clf = SVC(C=opt_hyperparameter_linear, kernel = 'linear')
clf.fit(x, y)
```

[264]:
```
SVC(C=0.003678379771828634, kernel='linear')
```

[265]:
```
clf = SVC(C=opt_hyperparameter_rb, kernel = 'rbf')
clf.fit(x, y)
```

[265]:
```
SVC(C=0.0010476157527896651)
```

1.1.3 Contesta lo siguientes:

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

Si, en el metodo de subsampling y upsampling solo toma en cuenta las primeras dos clases y el resto las ignora por completo.

¿Qué modelo o modelos fueron efectivos para clasificar tus datos? ¿Observas algo especial sobre los modelos? Argumenta tu respuesta.

el modelo de SVM y RB-SVM fueron los modelos mas efectivos para estos datos. Algo curioso de los modelos es que tanto en estos dos modelos como en KNN se predice con un 1 de efectividad la clase 4.

¿Observas alguna mejora importante al optimizar hiperparámetros? ¿Es el resultado que esperabas? Argumenta tu respuesta.

Existe una leve mejora al optimizar los hiperparametros. Debido a que el modelo o modelos originales ya presentaban un buen resultado, realmente no esperaba que mejorasen mucho más.

¿Qué inconvenientes hay al encontrar hiperparámetros? ¿Por qué?

Encontrar los hiperparametros optimos puede suponer una mayor carga computacional y gasto de tiempo para que en algunos casos no haya una mejora significativa o incluso puedan llegar a empeorar el modelo al evaluarlo con los datos de prueba.