# Clasificación2

September 10, 2024

### 0.0.1 EJERCICIO 2

A01285158 | Grace Aviance

En este ejercicio trabajarás con datos que vienen de un experimento en el que se midió actividad muscular con la técnica de la Electromiografía en el brazo derecho de varios participantes cuando éstos realizaban un movimiento con la mano entre siete posible (Flexionar hacia arriba, Flexionar hacia abajo, Cerrar la mano, Estirar la mano, Abrir la mano, Coger un objeto, No moverse). Al igual que en el ejercicio anterior, los datos se cargan con la función loadtxt de numpy (https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html)

A su vez, la primera columna corresponde a la clase (1, 2, 3, 4, 5, 6, y 7), la segunda columna se ignora, y el resto de las columnas indican las variables que se calcularon de la respuesta muscular. El archivo de datos con el que trabajarás depende de tu matrícula.

Para este conjunto de datos:

1. 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.
2. Evalúa al menos 8 modelos de clasificación distintos utilizando validación cruzada, y determina cuál de ellos es el más efectivo.
3. 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.
4. 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.

5. Contesta lo siguientes:

- ¿Observas un problema en cuanto al balanceo de las clases? ¿Por qué?
- ¿Qué modelo o modelos fueron efectivos para clasificar tus datos? ¿Observas algo especial sobre los modelos? Argumenta tu respuesta.
- ¿Observas alguna mejora importante al optimizar hiperparámetros? ¿Es el resultado que esperabas? Argumenta tu respuesta.
- ¿Qué inconvenientes hay al encontrar hiperparámetros? ¿Por qué?

[1]:
```
import pandas as pd
import numpy as np
```

[7]:
```
df_txt = np.loadtxt("M_3.txt")
df = pd.DataFrame(df_txt)
df.drop(columns = 1, inplace = True)
```

[9]:
```
df.head()
```

[9]:

| | 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | -1.558677 | 1.108312 | 0.362590 | -0.471151 | 0.385332 | 0.316969 | -1.481753 | |
| 1 | 1.0 | -1.978207 | 0.055298 | -0.273350 | -0.581922 | -0.955664 | -0.529833 | -1.983923 | |
| 2 | 1.0 | -2.002521 | 0.267381 | 0.263170 | -1.390732 | -0.285152 | -0.352082 | -1.917751 | |
| 3 | 1.0 | -2.152173 | 0.251282 | 0.183969 | -1.082561 | -0.088401 | 0.023352 | -2.048161 | |
| 4 | 1.0 | -2.590018 | -0.091138 | -0.265972 | -1.763059 | -0.181685 | -0.666207 | -2.570451 | |

| | 9 | 10 | … | 622 | 623 | 624 | 625 | 626 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.399394 | 1.148663 | … | -0.513296 | -0.044463 | 0.211337 | 0.404540 | -1.020636 | |
| 1 | -0.728869 | 0.253809 | … | -1.029816 | -0.305097 | -0.195680 | 0.641412 | -0.508980 | |
| 2 | -0.270610 | 0.568314 | … | -0.531808 | -0.482899 | -0.068502 | 0.126576 | -0.880254 | |
| 3 | 0.244747 | 0.376057 | … | -0.453654 | -0.107637 | -0.621580 | 0.807897 | 0.047029 | |
| 4 | -0.535319 | -0.031750 | … | -0.737064 | 0.055352 | 0.197176 | -1.152101 | -1.551049 | |

| | 627 | 628 | 629 | 630 | 631 |
|---|---|---|---|---|---|
| 0 | 0.598305 | 0.688470 | 0.292100 | -0.435294 | 1.384082 |
| 1 | 0.785134 | 0.580631 | 0.134605 | -0.663639 | 1.234545 |
| 2 | 0.533680 | 0.672030 | 0.207432 | -0.563343 | 1.046445 |
| 3 | 1.071315 | 1.204314 | 0.039504 | -0.899660 | 1.491114 |
| 4 | 1.236166 | 0.134769 | 0.805767 | -0.454501 | 1.101017 |

[5 rows x 631 columns]

### 0.0.2 1. Determina si es necesario balancear los datos.

[13]:
```
df.loc[:,0].value_counts()
```

[13]:

| 0 | |
|---|---|
| 1.0 | 90 |
| 2.0 | 90 |
| 3.0 | 90 |
| 4.0 | 90 |
| 5.0 | 90 |
| 6.0 | 90 |
| 7.0 | 90 |

Name: count, dtype: int64

El conjunto de datos está perfectamente balanceado

### 0.0.3 Definir X y y

[65]:
```
X = df.drop(columns = 0)
y = df.loc[:,0] - 1# For simplicity in future model training
```

### 0.0.4 2. Evalúa al menos 8 modelos de clasificación distintos utilizando validación cruzada, y determina cuál de ellos es el más efectivo.

### 0.0.5 Funcion para evaluar modelos

[18]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
```

[69]:
```
def evaluate_model(X, y, classifier):
    """
    Función para evaluar modelos utilizando validación cruzada sin subsampling␣
 ↪ni manejo de desbalanceo de clases.

    Parámetros:
    X (pd.DataFrame): Características
    y (pd.Series): Etiquetas (deben ser siete clases numeradas del 1 al 7)
    classifier: Modelo clasificador a evaluar

    Retorna:
    None: Imprime el reporte de clasificación de la validación cruzada.
    """

    # Validación cruzada con 5 pliegues estratificados
    kf = StratifiedKFold(n_splits=5, shuffle=True)

    cv_y_test = []
    cv_y_pred = []

    # Iterar sobre cada pliegue
    for train_index, test_index in kf.split(X, y):
        X_train = X.iloc[train_index]
        y_train = y.iloc[train_index]

        # Entrenar el modelo sin subsampling
        classifier.fit(X_train, y_train)

        # Fase de prueba
        X_test = X.iloc[test_index]
        y_test = y.iloc[test_index]
        y_pred = classifier.predict(X_test)

        # Guardar predicciones y etiquetas reales
        cv_y_test.append(y_test)
```

```
        cv_y_pred.append(pd.Series(y_pred, index=y_test.index))

    # Concatenar predicciones y etiquetas reales
    y_test_concat = pd.concat(cv_y_test)
    y_pred_concat = pd.concat(cv_y_pred)

    # Imprimir reporte de clasificación
    print(f"--- Reporte de clasificación ---")
    print(classification_report(y_test_concat, y_pred_concat, labels=[0, 1, 2,␣
↪3, 4, 5, 6]))
```

### 0.0.6 Evaluacion de modelos

[72]:
```
# Eight Classifiers
from sklearn.svm import SVC # Linear and RBF
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from xgboost import XGBClassifier
```

**MODELO 1. K-nearest-neighbors**

[75]:
```
evaluate_model(X, y, KNeighborsClassifier(n_neighbors=5))
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        90
           1       0.97      0.98      0.97        90
           2       0.96      0.90      0.93        90
           3       1.00      1.00      1.00        90
           4       1.00      1.00      1.00        90
           5       0.92      0.93      0.93        90
           6       0.95      0.99      0.97        90

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

**MODELO 2. Linear Discriminant Analysis**

[77]:
```
evaluate_model(X, y, LinearDiscriminantAnalysis())
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.91      0.98      0.94        90
```

```
           1       0.87      0.89      0.88        90
           2       0.87      0.86      0.86        90
           3       0.98      0.99      0.98        90
           4       0.99      0.89      0.94        90
           5       0.81      0.84      0.83        90
           6       0.94      0.90      0.92        90

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

### MODELO 3. Gaussian Naive-Bayes

[80]:
```
evaluate_model(X, y, GaussianNB())
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.87      0.86      0.86        90
           1       0.71      0.80      0.75        90
           2       0.87      0.77      0.82        90
           3       0.94      0.92      0.93        90
           4       0.89      0.91      0.90        90
           5       0.67      0.64      0.66        90
           6       0.94      0.99      0.96        90

    accuracy                           0.84       630
   macro avg       0.84      0.84      0.84       630
weighted avg       0.84      0.84      0.84       630
```

### MODELO 4. Linear Support Vector Classifier

[82]:
```
evaluate_model(X, y, SVC(kernel='linear'))
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        90
           1       0.99      0.98      0.98        90
           2       0.97      0.94      0.96        90
           3       1.00      1.00      1.00        90
           4       0.99      1.00      0.99        90
           5       0.94      0.93      0.94        90
           6       0.94      0.98      0.96        90

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

### MODELO 5. Radial Support Vector Classifier

[84]:
```
evaluate_model(X, y, SVC(kernel='rbf'))
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        90
           1       0.98      0.93      0.95        90
           2       0.99      0.87      0.92        90
           3       1.00      1.00      1.00        90
           4       0.99      0.99      0.99        90
           5       0.85      0.94      0.89        90
           6       0.94      0.99      0.96        90

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

### MODELO 6. Desicion Tree

[86]:
```
evaluate_model(X, y, DecisionTreeClassifier())
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.86      0.82      0.84        90
           1       0.70      0.66      0.68        90
           2       0.78      0.83      0.81        90
           3       0.92      0.87      0.89        90
           4       0.89      0.90      0.90        90
           5       0.64      0.70      0.67        90
           6       0.94      0.93      0.94        90

    accuracy                           0.82       630
   macro avg       0.82      0.82      0.82       630
weighted avg       0.82      0.82      0.82       630
```

### MODELO 7. Random Forest

[92]:
```
evaluate_model(X, y, RandomForestClassifier())
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        90
           1       0.87      0.91      0.89        90
           2       0.94      0.86      0.90        90
```

```
              3       1.00      0.98      0.99        90
              4       0.96      0.98      0.97        90
              5       0.82      0.82      0.82        90
              6       0.94      1.00      0.97        90

       accuracy                           0.93       630
      macro avg       0.93      0.93      0.93       630
   weighted avg       0.93      0.93      0.93       630
```

**MODELO 8. XGBoost**

[94]:
```
evaluate_model(X, y, XGBClassifier())
```

```
--- Reporte de clasificación ---
              precision    recall  f1-score   support

           0       0.97      0.93      0.95        90
           1       0.86      0.84      0.85        90
           2       0.88      0.84      0.86        90
           3       1.00      0.97      0.98        90
           4       0.97      1.00      0.98        90
           5       0.75      0.78      0.77        90
           6       0.94      1.00      0.97        90

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

### 0.0.7 3. 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.

**Modelos seleccionados: Linear SVC y KNN**

**Malla de hiperparámetros para ambos modelos**

[128]:
```
from sklearn.svm import LinearSVC
```

[152]:
```
# Grid para KNN
param_grid_knn = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}


# Grid para Linear SVC
```

```
param_grid_svc = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'loss': ['squared_hinge'],
    'dual': [False],
    'max_iter': [1000, 2000, 5000]
}
```

**Funcion de validacion cruzada anidada para encontrar los mejores hiperparametros**

[107]:
```
from sklearn.model_selection import GridSearchCV
```

[121]:
```
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import classification_report
import pandas as pd

def evaluate_model_with_nested_cv(X, y, classifier, param_grid):
    """
    Función para evaluar modelos utilizando validación cruzada anidada con␣
 ↪búsqueda de hiperparámetros.

    Parámetros:
    X (pd.DataFrame): Características (ya estandarizadas)
    y (pd.Series): Etiquetas (deben ser siete clases numeradas del 1 al 7)
    classifier: Modelo clasificador a evaluar
    param_grid (dict): Diccionario de hiperparámetros a explorar en GridSearchCV

    Retorna:
    None: Imprime el reporte de clasificación de la validación cruzada.
    """

    # Validación cruzada externa con 5 pliegues
    outer_cv = StratifiedKFold(n_splits=5, shuffle=True)

    cv_y_test = []
    cv_y_pred = []

    # Validación cruzada anidada (con GridSearchCV)
    for train_index, test_index in outer_cv.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # GridSearchCV para buscar los mejores hiperparámetros
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True)
        grid_search = GridSearchCV(classifier, param_grid, cv=inner_cv,␣
 ↪scoring='accuracy')

        # Ajustar GridSearchCV en el conjunto de entrenamiento
        grid_search.fit(X_train, y_train)
```

```
        # Predecir en el conjunto de prueba externo con los mejores␣
↪hiperparámetros
        y_pred = grid_search.predict(X_test)

        # Guardar predicciones y etiquetas reales
        cv_y_test.append(y_test)
        cv_y_pred.append(pd.Series(y_pred, index=y_test.index))

        # Imprimir los mejores hiperparámetros encontrados en este pliegue␣
↪externo
        print(f"Mejores hiperparámetros en este pliegue: {grid_search.
↪best_params_}")

    # Concatenar todas las predicciones y etiquetas reales
    y_test_concat = pd.concat(cv_y_test)
    y_pred_concat = pd.concat(cv_y_pred)

    # Imprimir reporte de clasificación final
    print(f"--- Reporte de clasificación con validación cruzada anidada ---")
    print(classification_report(y_test_concat, y_pred_concat, labels=[0, 1, 2,␣
↪3, 4, 5, 6]))
```

### 0.0.8 Evaluar modelos con hiperparametros

**K-nearest neighbors**

[125]:
```
evaluate_model_with_nested_cv(X, y, KNeighborsClassifier(), param_grid_knn)
```

```
Mejores hiperparámetros en este pliegue: {'metric': 'manhattan', 'n_neighbors':
3, 'weights': 'distance'}
Mejores hiperparámetros en este pliegue: {'metric': 'euclidean', 'n_neighbors':
3, 'weights': 'distance'}
Mejores hiperparámetros en este pliegue: {'metric': 'manhattan', 'n_neighbors':
5, 'weights': 'distance'}
Mejores hiperparámetros en este pliegue: {'metric': 'manhattan', 'n_neighbors':
3, 'weights': 'distance'}
Mejores hiperparámetros en este pliegue: {'metric': 'euclidean', 'n_neighbors':
3, 'weights': 'uniform'}
--- Reporte de clasificación con validación cruzada anidada ---
              precision    recall  f1-score   support

           0       0.99      0.98      0.98        90
           1       0.96      0.94      0.95        90
           2       0.99      0.90      0.94        90
           3       1.00      0.99      0.99        90
           4       0.99      1.00      0.99        90
           5       0.90      0.97      0.93        90
           6       0.95      0.98      0.96        90
```

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

**Linear Support Vector Classifier**

[154]:
```
evaluate_model_with_nested_cv(X, y, LinearSVC(), param_grid_svc)
```

```
Mejores hiperparámetros en este pliegue: {'C': 0.001, 'dual': False, 'loss':
'squared_hinge', 'max_iter': 1000}
Mejores hiperparámetros en este pliegue: {'C': 0.01, 'dual': False, 'loss':
'squared_hinge', 'max_iter': 1000}
Mejores hiperparámetros en este pliegue: {'C': 0.001, 'dual': False, 'loss':
'squared_hinge', 'max_iter': 1000}
Mejores hiperparámetros en este pliegue: {'C': 0.01, 'dual': False, 'loss':
'squared_hinge', 'max_iter': 1000}
Mejores hiperparámetros en este pliegue: {'C': 0.001, 'dual': False, 'loss':
'squared_hinge', 'max_iter': 1000}
--- Reporte de clasificación con validación cruzada anidada ---
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        90
           1       0.98      0.97      0.97        90
           2       0.98      0.96      0.97        90
           3       1.00      1.00      1.00        90
           4       1.00      1.00      1.00        90
           5       0.95      0.93      0.94        90
           6       0.94      0.99      0.96        90

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

### 0.0.9 4. Preparacion de modelos para produccion

**K-nearest neighbors**

[162]:
```
clf_knn = KNeighborsClassifier(metric= 'manhattan', n_neighbors= 3, weights=␣
↪'distance')
```

[164]:
```
clf_knn.fit(X, y)
```

[164]:
```
KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')
```

[170]:
```
# Guardar el modelo en un archivo pickle
'''
with open('tuned_knn_model.pkl', 'wb') as model_file:
```

```
    pickle.dump(clf_knn, model_file)

print("Modelo KNN guardado en 'tuned_knn_model.pkl'")
'''
```

[170]:
```
'\nwith open(\'tuned_knn_model.pkl\', \'wb\') as model_file:\n
pickle.dump(clf_knn, model_file)\n\nprint("Modelo KNN guardado en
\'tuned_knn_model.pkl\'")\n'
```

**Linear SVC**

[174]:
```
clf_svc = LinearSVC(C = 0.001, dual = False, loss = 'squared_hinge', max_iter =␣
↪1000)
```

[176]:
```
clf_svc.fit(X, y)
```

[176]:
```
LinearSVC(C=0.001, dual=False)
```

[178]:
```
# Guardar el modelo en un archivo pickle
'''
with open('tuned_svc_model.pkl', 'wb') as model_file:
    pickle.dump(clf_svc, model_file)

print("Modelo KNN guardado en 'tuned_svc_model.pkl'")
'''
```

[178]:
```
'\nwith open(\'tuned_svc_model.pkl\', \'wb\') as model_file:\n
pickle.dump(clf_svc, model_file)\n\nprint("Modelo KNN guardado en
\'tuned_svc_model.pkl\'")\n'
```

### 0.0.10 5. Conclusiones

Contesta lo siguiente:

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

No, porque existían exactamente noventa observaciones para cada clase. Sin embargo, es necesario asegurarse de que a la hora de evaluar las observaciones hayan sido mezcladas previamente ya que en el dataset original no lo están.

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

Los modelos más efectivos fueron Linear Support Vector Machines y K-nearest neighbors. Los demás modelos tenían específicamente dificultad para clasificar el movimiento de abrir la mano. Este resultado sorprende bastante a primera vista, ya que otros modelos que sobre el papel son más potentes también fueron evaluados, tales como Random Forest o XGBoost. Este interesante resultado puede deberse a múltiples factores, tales como la dimensionalidad del conjunto de datos, el número de observaciones, o un comportamiento lineal entre las características del conjunto de datos.

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

Debido a que los resultados ya eran demasiado buenos, la mejora fue mínima. Sin embargo, cualquier mejora en un modelo de machine learning vale la pena.

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

En el caso de Linear Support Vector Machines, se debe tener cuidado al escoger la malla de hiperparámetros ya pueden existir problemas de convergencia o compatibilidad entre ciertos hiperparámetros. Otro problema que no se presentó en este caso pero puede pasar, es que este proceso sea excesivo en cuanto a tiempo para modelos más complejos que los que se utilizaron.