Reporte: Análisis de datos para aplicaciones médicas médicas

Jose Emilio Martinez Hernandez (A01403100)
Josue Tapia Hernandez (A01621056)

2025-06-10

Tabla de contenidos

1 Descripción de la aplicación implementada

Durante el transcurso de este proyecto estuvimos desarrollando una aplicación médica que consiste en un detector de señales referentes a la aceleración de un objeto, en este caso con un teléfono. A través de estas señales, la aplicación identificará características importantes de las señales recibidas para así clasificarlas en diferentes actividades ya registradas en la aplicación.

En cuanto la aplicación esté en uso, se espera que clasifique de forma correcta los diferentes movimientos que la persona está realizando. Por ejemplo, en nuestro caso, nosotros esperamos que la aplicación pueda detectar si es que está haciendo sentadillas, lagartijas, subir escaleras, bajar escaleras, crunches o jumping jacks. Todo esto se implementa a través de un modelo de clasificación utilizando varias librerías como Scikit-learn o NumPy.

Dentro del mercado actual se encuentran varios ejemplos de esta tecnología pero más avanzada y trabajada. Por ejemplo se encuentran las aplicaciones instaladas en los smartwatch que usan tecnologias similares para detectar si estas haciendo ejercicio o corriendo, la apcliacion “Health” de apple es un muy buen ejemplo ya que este aplicacion utiliza la diversidad de sensores que hay en los SmartWatch para detectar varias de tus actividades diarias y asi darte recomendaciones acerca de tu estilo de vida. Igualmente aplicaciones como Google Fit, Apple Health y FitBit utilizan los acelerometros para contar pasos mediante la deteccion de movimientos repetitivos en el cuerpp

Detallando más en qué tipo de información se va a monitorear, el objetivo es que a partir de señales del sensor de aceleración del celular, en los 3 ejes x, y, z, se pueda detectar si el usuario está realizando alguna actividad física, esto con el propósito de monitorear su estado de salud y actividad física. A través de los datos recopilados, nuestro programa calcula varias características de las señales como el promedio, la curtosis y la varianza, todo esto con el objetivo de tener tantas métricas posibles para poder dar la mejor clasificación posible. En el próximo apartado se estará detallando más acerca de las características y su implementación en la aplicación.

2 Descripción de las características extraídas de los datos

A continuacion se encuentran las diferentes caracteristicas que se calcularon a traves de los datos recopilados por la aplicacion pyphox. Estas caracteristicas nos serviran como variables para poder predecir que tipo de actividad se esta realizando. Estas caracterisitcas se aplican a una ventana de informacion que se extrae directamente de la aplicacion, este proceso se hace continuamente por varios segundos de prueba para asi entrenar el modelo posteriormente

2.1 Promedio

El promedio de los datos nos permite ver el valor central en un conjunto de datos, se calcula sumando todos los valores y dividiendo entre la cantidad total de elementos (Altin y Er (n.d.))

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

2.2 Desviacion estándar

La desviación estándar mide cuánto se alejan, en promedio, los datos respecto a la media. Es una forma de cuantificar la dispersión o variabilidad de un conjunto de datos. Un valor bajo indica que los datos están cerca del promedio; uno alto, que están más dispersos (Sara Abbaspour (2025))

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

2.3 Kurtosis

La kurtosis mide la “forma” de la distribución de los datos, específicamente cuán “picuda” o “aplanada” es en comparación con una distribución normal (Sanjuán (2024))

- Una kurtosis alta indica una distribución con colas más pesadas y un pico más pronunciado.
- Una kurtosis baja indica una distribución más plana con colas ligeras.

$$K = \frac{n(n+1)}{(n-1)(n-2)(n-3)}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

2.4 Skewness

- La **asimetría** mide el grado de **simetría** de una distribución de datos. Indica si los valores están más concentrados a un lado de la media (Altin y Er (n.d.))
- **Asimetría positiva** (skew > 0): la cola derecha (valores altos) es más larga.

- **Asimetría negativa** (skew < 0): la cola izquierda (valores bajos) es más larga.
- **Skew = 0**: la distribución es simétrica (como la normal)

$$g_1 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s} \right)^3$$

2.5 Median Absolute Deviation

La MAD es una medida robusta de dispersión. Indica qué tan lejos, en promedio, están los datos de la mediana, en lugar de la media. Es menos sensible a valores atípicos (outliers) que la desviación estándar, por lo que se usa mucho en análisis estadísticos resistentes (Lee (2025))

$$\text{MAD} = \text{median}\left(|x_i - \text{median}(x)|\right)$$

2.6 Root Mean Square (RMS)

La **RMS (Root Mean Square)** es una medida de la magnitud promedio de una señal. En señales fisiológicas como el EMG, la RMS refleja el nivel de activación muscular y es útil para cuantificar la intensidad de la señal, ya que considera tanto valores positivos como negativos.

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2}$$

2.7 Zero Crossing Rate (ZCR)

La ZCR mide cuántas veces cambia el signo de una señal (de positivo a negativo o viceversa) en un intervalo. Es útil para identificar el contenido de alta frecuencia en una señal, como distinguir entre sonidos suaves y ruidosos (Sara Abbaspour (2025))

$$\text{ZCR} = \frac{1}{T-1} \sum_{t=1}^{T-1} \mathbb{1}_{\{x_t \cdot x_{t-1} < 0\}}$$

2.8 Slope Sign Changes (SSP)

El SSC cuenta cuántas veces cambia el signo de la pendiente de una señal entre tres puntos consecutivos. Es una medida de la frecuencia de oscilación o “actividad” en una señal. Es útil para detectar variaciones rápidas en señales como la EMG (Altin y Er (n.d.)) @

$$\text{SSC} = \sum_{t=2}^{T-1} \mathbb{1}_{\{(x_t - x_{t-1})(x_t - x_{t+1}) > \theta\}}$$

2.9 Waveform Length (WL)

El Waveform Length mide la complejidad o actividad de una señal sumando las diferencias absolutas entre muestras consecutivas. Refleja tanto la amplitud como la frecuencia de las variaciones en la señal (Sara Abbaspour (2025))

$$\text{WL} = \sum_{t=1}^{T-1} |x_{t+1} - x_t|$$

2.10 Energia de una señal

La energía mide la intensidad total de una señal a lo largo del tiempo. Es especialmente útil para detectar la presencia y fuerza de actividad dentro de una ventana de análisis (Sara Abbaspour (2025))

$$\text{Energy} = \sum_{t=1}^{T} x_t^2$$

3 Evaluacion de algoritmos de clasificacion

Como primer paso estaremos evaluando algunos modelos de clasificacion, tanto los que vimos en clase como algunos nuevos

Primeramente cargamos librerias y la base de datos que anteriormente recopilamos a traves de un codigo de recoleccion de datos

```
import numpy as np
from sklearn.model_selection import StratifiedKFold,
 ↪  cross_validate, cross_val_predict, LeaveOneOut, RepeatedKFold,
 ↪  cross_val_score
from sklearn.metrics import classification_report,
 ↪  confusion_matrix, accuracy_score, recall_score, precision_score
```

```
data = np.loadtxt('../Class Scripts/activity_data.txt')
x = data[:,1:]
y = data[:,0]
```

3.1 SVM Lineal

```
from sklearn.svm import SVC

#------ SVM LINEAL
n_folds = 5
```

```
clf_cv = SVC(kernel='linear')
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.88      0.73      0.80        30
         2.0       1.00      0.97      0.98        30
         3.0       0.77      0.90      0.83        30
         4.0       1.00      1.00      1.00        30
         5.0       0.87      0.87      0.87        30
         6.0       0.87      0.90      0.89        30

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180
```

3.2 SVM Base radial

```
#------ SVM BASE RADIAL
n_folds = 5
clf_cv = SVC(kernel='rbf')
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.83      0.17      0.28        30
         2.0       0.97      0.97      0.97        30
         3.0       0.08      0.03      0.05        30
         4.0       0.42      1.00      0.59        30
         5.0       0.92      0.73      0.81        30
         6.0       0.81      0.97      0.88        30

    accuracy                           0.64       180
   macro avg       0.67      0.64      0.60       180
weighted avg       0.67      0.64      0.60       180
```

3.3 Linear Discriminat Analysis (LDA)

```
from sklearn.discriminant_analysis import
 ↪  LinearDiscriminantAnalysis

#------ LDA
n_folds = 5
clf_cv = LinearDiscriminantAnalysis()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

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

```
              1.0       0.93      0.93      0.93        30
              2.0       1.00      1.00      1.00        30
              3.0       0.90      0.90      0.90        30
              4.0       0.97      0.97      0.97        30
              5.0       0.96      0.90      0.93        30
              6.0       0.91      0.97      0.94        30

         accuracy                           0.94       180
        macro avg       0.95      0.94      0.94       180
     weighted avg       0.95      0.94      0.94       180
```

3.4 K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

#------ KNN
n_folds = 5
clf_cv = KNeighborsClassifier()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.87      0.90        30
         2.0       0.90      0.90      0.90        30
         3.0       0.88      0.93      0.90        30
         4.0       1.00      1.00      1.00        30
         5.0       0.77      0.67      0.71        30
         6.0       0.79      0.90      0.84        30

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180
```

3.5 Red Neuronal (MLP)

```
from sklearn.neural_network import MLPClassifier

#---------MLP
n_folds = 5
clf_cv = MLPClassifier(hidden_layer_sizes=(100,100),max_iter=10000)
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.88      0.93      0.90        30
         2.0       0.96      0.90      0.93        30
         3.0       0.93      0.90      0.92        30
         4.0       0.97      1.00      0.98        30
```

```
         5.0       0.79      0.73      0.76        30
         6.0       0.78      0.83      0.81        30

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180
```

3.6 Gaussian Naive Bayes

```python
from sklearn.naive_bayes import GaussianNB

##----------Gaussian Naive Bayes
n_folds = 5
clf_cv = GaussianNB()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.93      0.93        30
         2.0       1.00      0.93      0.97        30
         3.0       0.97      0.93      0.95        30
         4.0       1.00      1.00      1.00        30
         5.0       0.82      0.90      0.86        30
         6.0       0.90      0.90      0.90        30

    accuracy                           0.93       180
   macro avg       0.94      0.93      0.93       180
weighted avg       0.94      0.93      0.93       180
```

3.7 Ridge Classifier

```python
from sklearn.linear_model import RidgeClassifier

#-----------Ridge Classifier
n_folds = 5
clf_cv = RidgeClassifier()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.83      0.63      0.72        30
         2.0       1.00      1.00      1.00        30
         3.0       0.85      0.77      0.81        30
         4.0       0.71      0.97      0.82        30
         5.0       0.88      0.77      0.82        30
         6.0       0.82      0.90      0.86        30

    accuracy                           0.84       180
   macro avg       0.85      0.84      0.84       180
weighted avg       0.85      0.84      0.84       180
```

3.8 Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

#--------- Gradient Boosting Classifier
n_folds = 5
clf_cv = GradientBoostingClassifier()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.87      0.90        30
         2.0       0.96      0.90      0.93        30
         3.0       0.90      0.93      0.92        30
         4.0       1.00      1.00      1.00        30
         5.0       0.74      0.83      0.78        30
         6.0       0.86      0.83      0.85        30

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.90       180
weighted avg       0.90      0.89      0.90       180
```

3.9 Nearest Centroid

```
from sklearn.neighbors import NearestCentroid

#----------- NearestCentroid
n_folds = 5
clf_cv = NearestCentroid()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.87      0.67      0.75        30
         2.0       0.82      0.77      0.79        30
         3.0       0.73      0.53      0.62        30
         4.0       0.65      1.00      0.79        30
         5.0       0.80      0.53      0.64        30
         6.0       0.71      0.97      0.82        30

    accuracy                           0.74       180
   macro avg       0.76      0.74      0.73       180
weighted avg       0.76      0.74      0.73       180
```

3.10 Stochastic Gradient Descent Classifier

```
from sklearn.linear_model import SGDClassifier

#--------SGDClassifier
n_folds = 5
clf_cv = SGDClassifier()
y_pred = cross_val_predict(clf_cv, x, y, cv=n_folds)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.38      0.27      0.31        30
         2.0       0.74      0.67      0.70        30
         3.0       0.42      0.33      0.37        30
         4.0       1.00      0.93      0.97        30
         5.0       0.41      0.67      0.51        30
         6.0       0.26      0.27      0.26        30

    accuracy                           0.52       180
   macro avg       0.53      0.52      0.52       180
weighted avg       0.53      0.52      0.52       180
```

3.11 Conclusion de resultados obtenidos

En estos primeros resultados donde los modelos aún no están optimizados, podemos ver que en su gran mayoría todos dan un accuracy bueno. Pero hubo algunos modelos que no tuvieron el rendimiento esperado, como fue el caso del SVM con base radial que tuvo un accuracy de 0.64, Nearest Centroid con un accuracy de 0.73 y SGD Classifier con un accuracy de 0.57, este último siendo el peor de todos. Los modelos que mejor se desempeñaron fueron el SVM lineal con 0.89, LDA con 0.94 y Gaussian Naive Bayes con 0.93

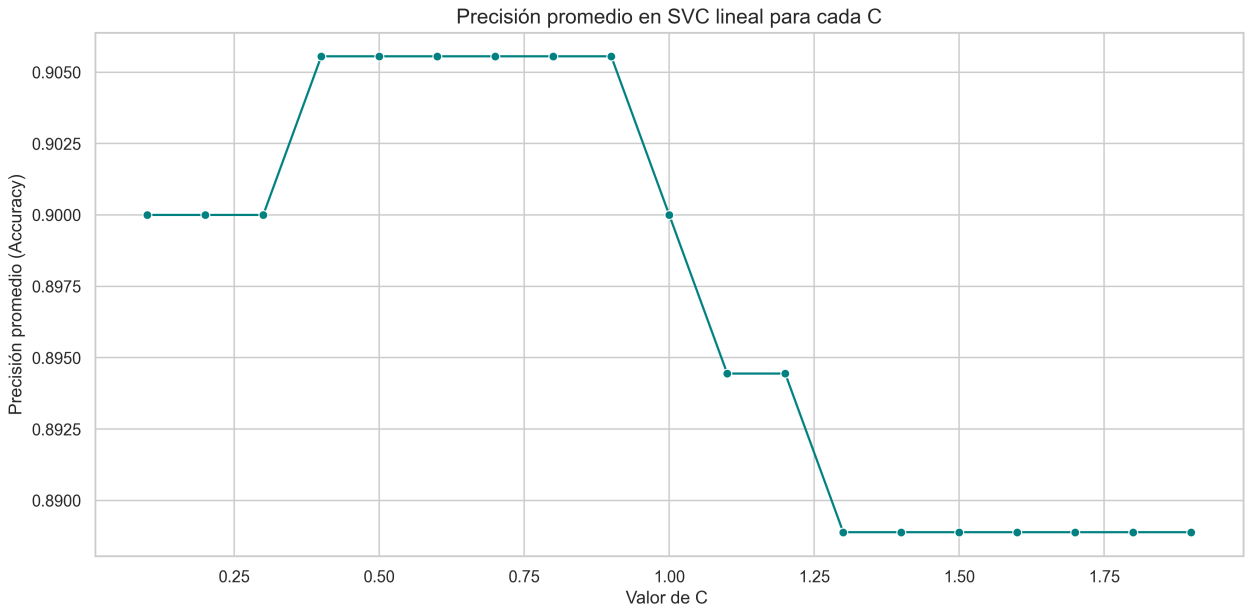
4 Evaluacion del los algoritmos con optimización

Para este siguiente apartado elegimos los dos clasificadores con mejor desempeño en la seccion anterior para optimizar mediante validacion cruzada sus hiperparametros, igualmente estaremos usando esta misma metodologia para ver si podemos hacer una reduccion de dimensionalidad para los mismos dos modelos

4.1 Optimizacion Hiperparametros (SVM lineal)

```
from sklearn.svm import SVC
import numpy as np
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
```

```
data=np.loadtxt("../Class Scripts/activity_data.txt")
x=data[:,1:]
y=data[:,0]
kf1 = StratifiedKFold(n_splits=5, shuffle = True,random_state=45)


best_c = 1
best_acc = 0
Mean_test_score=[]
for c in  np.arange(1e-1,2, 1e-1):
    acc_hyp_cv = []
    clf_cv_hyp = SVC(C=c,kernel='linear')


    for train_index, test_index in kf1.split(x, y):
        x_train = x[train_index, :]
        y_train = y[train_index]
        x_test = x[test_index, :]
        y_test = y[test_index]
        clf_cv_hyp.fit(x_train,y_train)
        y_pred = clf_cv_hyp.predict(x_test)
        acc_i = accuracy_score(y_test, y_pred)
        acc_hyp_cv.append(acc_i)

    acc_hyp = np.average(acc_hyp_cv)

    if (best_acc < acc_hyp):
        best_c = c
        best_acc = acc_hyp

    Mean_test_score.append( acc_hyp)

# Valores de C usados
k_values = np.arange(1e-1, 2, 1e-1)
accuracies = Mean_test_score

sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

# Gráfica
sns.lineplot(x=k_values, y=accuracies, marker='o', color='teal')
 # Escala logarítmica en X



# Etiquetas y diseño
plt.title('Precisión promedio en SVC lineal para cada C',
 ↪  fontsize=14)
plt.xlabel('Valor de C ', fontsize=12)
plt.ylabel('Precisión promedio (Accuracy)', fontsize=12)
plt.tight_layout()
plt.show()
print("Best estimator:",best_c)
```




```
Best estimator: 0.4
```

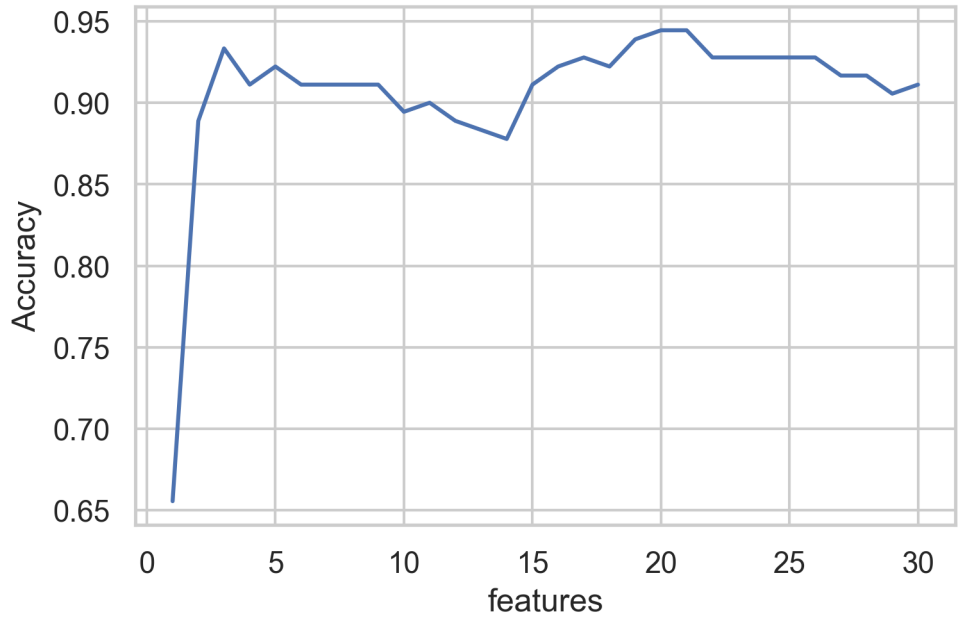
4.2 Optimizacion Features (SVM lineal)

```
from sklearn.feature_selection import SelectKBest, f_classif,
 ↪  SequentialFeatureSelector, RFE
n_feats = np.arange(1,31,1)
acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

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

    for train_idx, test_idx in kf.split(x,y):
        x_train = x[train_idx,:]
        y_train = y[train_idx]

        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)

        x_test = fselection_cv.transform(x[test_idx, :])
        y_test = y[test_idx]
        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)

    plt.plot(n_feats, acc_nfeat)
    plt.xlabel("features")
    plt.ylabel("Accuracy")

    plt.show()

    # Fit model with optimal number of features
    clf = SVC(kernel = 'linear')
    fselection = SelectKBest(f_classif,k=n_feat)
    fselection.fit(x, y)

    print("Selected features: ", fselection.get_feature_names_out())
```

```
---- n features = 1
ACC: 0.6555555555555556
---- n features = 2
ACC: 0.888888888888889
---- n features = 3
ACC: 0.9333333333333333
---- n features = 4
ACC: 0.9111111111111111
---- n features = 5
ACC: 0.9222222222222222
---- n features = 6
ACC: 0.9111111111111111
---- n features = 7
ACC: 0.9111111111111111
---- n features = 8
ACC: 0.9111111111111111
---- n features = 9
ACC: 0.9111111111111111
---- n features = 10
ACC: 0.8944444444444445
---- n features = 11
ACC: 0.9
---- n features = 12
ACC: 0.888888888888889
---- n features = 13
ACC: 0.8833333333333334
---- n features = 14
ACC: 0.8777777777777779
---- n features = 15
ACC: 0.9111111111111111
---- n features = 16
ACC: 0.9222222222222223
---- n features = 17
ACC: 0.9277777777777778
---- n features = 18
```

```
ACC: 0.9222222222222222
---- n features = 19
ACC: 0.9388888888888889
---- n features = 20
ACC: 0.9444444444444444
---- n features = 21
ACC: 0.9444444444444444
---- n features = 22
ACC: 0.9277777777777777
---- n features = 23
ACC: 0.9277777777777777
---- n features = 24
ACC: 0.9277777777777777
---- n features = 25
ACC: 0.9277777777777777
---- n features = 26
ACC: 0.9277777777777777
---- n features = 27
ACC: 0.9166666666666666
---- n features = 28
ACC: 0.9166666666666666
---- n features = 29
ACC: 0.9055555555555556
---- n features = 30
ACC: 0.9111111111111111
Optimal number of features:  20
```




```
Selected features: ['x0' 'x1' 'x2' 'x3' 'x4' 'x5' 'x6' 'x7' 'x8' 'x9' 'x10' 'x11' 'x12
 'x14' 'x15' 'x16' 'x17' 'x18' 'x19' 'x20' 'x21' 'x22' 'x23' 'x24' 'x25'
 'x26' 'x27' 'x28' 'x29']
```

Aquí podemos ver que hacemos un cross validation para cada una de las *k*, que representan la cantidad de features que vamos a elegir a través del método filter “Select K Best”. Después de aplicar este procedimiento, podemos ver que en efecto se puede sacar un poco más de rendimiento

al reducir el número de características a 20. Cabe recalcar que al aplicar el procedimiento varias veces, pudimos comprobar que el número de features siempre se reduce a 19, 20 o 21.

4.3 Evaluacion del modelo utilizando validacion cruzada (SVM Lineal)

```
data=np.loadtxt("../Class Scripts/activity_data.txt")
x=data[:,1:]
y=data[:,0]
def Select_features(x,y,c):
    n_feats = np.arange(1,31,1)
    acc_nfeat = []

    for n_feat in n_feats:


        acc_cv = []

        kf = StratifiedKFold(n_splits=5, shuffle = True)

        for train_idx, test_idx in kf.split(x,y):
            x_train = x[train_idx,:]
            y_train = y[train_idx]

            clf_cv = SVC(C=c,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)

            x_test = fselection_cv.transform(x[test_idx, :])
            y_test = y[test_idx]
            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)



    opt_index = np.argmax(acc_nfeat)
    opt_features = n_feats[opt_index]
    return opt_features

kf1 = StratifiedKFold(n_splits=5, shuffle = True)
kf2 = StratifiedKFold(n_splits=5, shuffle = True)
cv_y_test = []
cv_y_pred = []
acc=[]

for train_index, test_index in kf1.split(x, y):
    x_train = x[train_index, :]
    y_train = y[train_index]
```

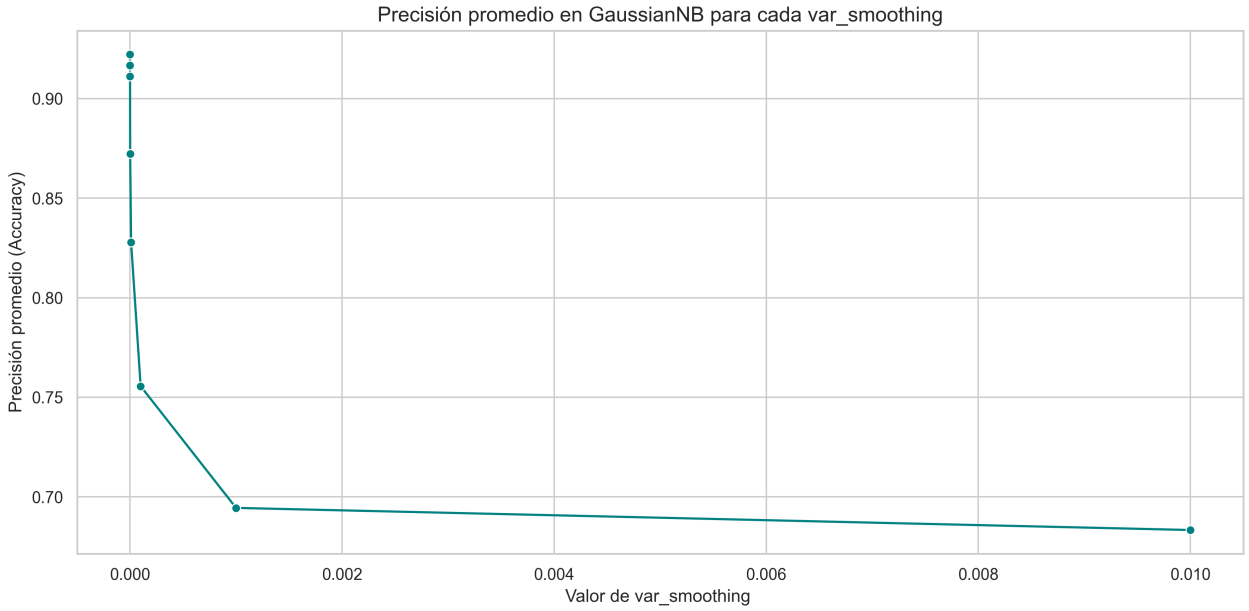
```
    x_test=x[test_index,:]
    y_test=y[test_index]

    best_c = 0
    best_acc = 0
    for c in np.arange(1e-1,2, 1e-1):
        acc_hyp_cv = []
        for sub_train_index, val_index in kf2.split(x_train,
        ↪  y_train):
            x_sub_train = x[sub_train_index, :]
            y_sub_train = y[sub_train_index]
            x_val = x[val_index, :]
            y_val = y[val_index]
           Opt_features=Select_features( x_sub_train,y_sub_train,c)
            clf_cv = SVC(C=c,kernel='linear')
            fselection_cv = SelectKBest(f_classif,k= Opt_features)
            fselection_cv.fit(x_sub_train,y_sub_train)
            x_sub_train = fselection_cv.transform(x_sub_train)
            clf_cv.fit(x_sub_train,y_sub_train)
            x_val=fselection_cv.transform(x_val)
            y_pred = clf_cv.predict(x_val)
            acc_i = accuracy_score(y_val, y_pred)
            acc_hyp_cv.append(acc_i)
        acc_hyp = np.average(acc_hyp_cv)
        if (best_acc < acc_hyp):
            best_c = c
            best_acc = acc_hyp
    best_features=0
    Opt_features=Select_features( x_train,y_train,best_c)
    clf_cv = SVC(C=best_c,kernel='linear')
    fselection_cv = SelectKBest(f_classif,k= Opt_features)
    fselection_cv.fit(x_train,y_train)
    x_train=fselection_cv.transform(x_train)
    clf_cv.fit(x_train,y_train)
    x_test=fselection_cv.transform(x_test)
    y_pred = clf_cv.predict(x_test)
    acc_f = accuracy_score(y_test, y_pred)
    acc.append(acc_f)

final_acc=np.average(acc)
print(final_acc)
```

```
0.9
```

4.4 Optimizacion Hiperparametros (Gaussian NB)

```
from sklearn.svm import SVC
import numpy as np
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score

data=np.loadtxt("../Class Scripts/activity_data.txt")
x=data[:,1:]
```

```
y=data[:,0]
kf1 = StratifiedKFold(n_splits=5, shuffle = True,random_state=45)


best_c = 1
best_acc = 0
Mean_test_score=[]
for c in [1e-12, 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4,
 ↪  1e-3, 1e-2]:
    acc_hyp_cv = []
    clf_cv_hyp = GaussianNB(var_smoothing=c)


    for train_index, test_index in kf1.split(x, y):
        x_train = x[train_index, :]
        y_train = y[train_index]
        x_test = x[test_index, :]
        y_test = y[test_index]
        clf_cv_hyp.fit(x_train,y_train)
        y_pred = clf_cv_hyp.predict(x_test)
        acc_i = accuracy_score(y_test, y_pred)
        acc_hyp_cv.append(acc_i)

    acc_hyp = np.average(acc_hyp_cv)

    if (best_acc < acc_hyp):
        best_c = c
        best_acc = acc_hyp

    Mean_test_score.append( acc_hyp)

# Valores de C usados
k_values = [1e-12, 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5,
 ↪  1e-4, 1e-3, 1e-2]
accuracies = Mean_test_score

sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

# Gráfica
sns.lineplot(x=k_values, y=accuracies, marker='o', color='teal')
 # Escala logarítmica en X



# Etiquetas y diseño
plt.title('Precisión promedio en GaussianNB para cada
 ↪  var_smoothing', fontsize=14)
plt.xlabel('Valor de var_smoothing ', fontsize=12)
plt.ylabel('Precisión promedio (Accuracy)', fontsize=12)
plt.tight_layout()
plt.show()
print("Best estimator:", best_c)
```




```
Best estimator: 1e-09
```

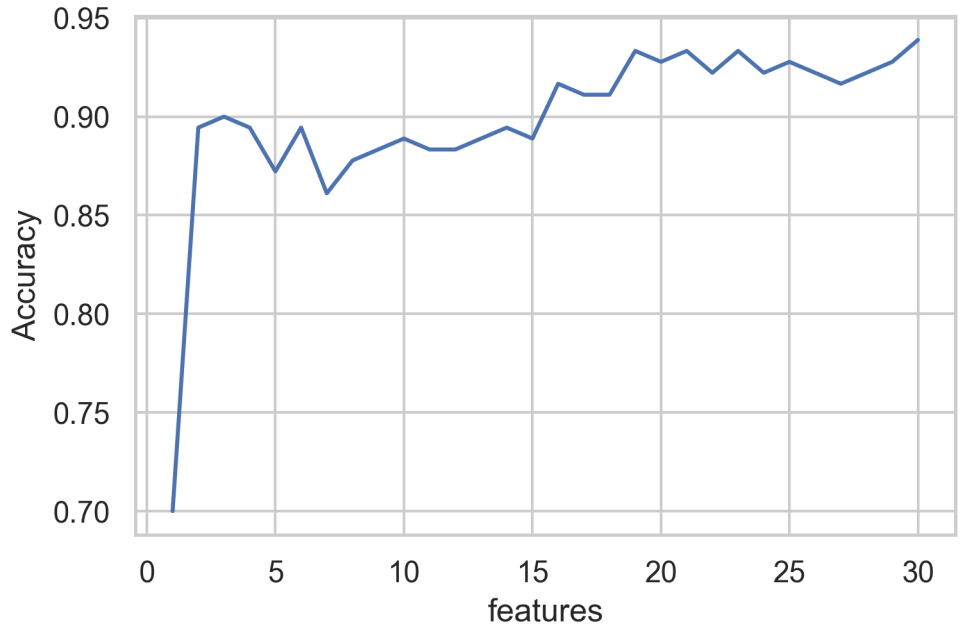
4.5 Optimizacion Features (Gaussian NB)

```
from sklearn.feature_selection import SelectKBest, f_classif,
 ↪  SequentialFeatureSelector, RFE, mutual_info_classif

data=np.loadtxt("../Class Scripts/activity_data.txt")
x=data[:,1:]
y=data[:,0]

n_feats = np.arange(1,31,1)
acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_idx, test_idx in kf.split(x,y):
        x_train = x[train_idx,:]
        y_train = y[train_idx]

        clf_cv = GaussianNB()

        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)

        x_test = fselection_cv.transform(x[test_idx, :])
```

```
            y_test = y[test_idx]
            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)

    plt.plot(n_feats, acc_nfeat)
    plt.xlabel("features")
    plt.ylabel("Accuracy")

    plt.show()
```

```
---- n features = 1
ACC: 0.7
---- n features = 2
ACC: 0.8944444444444443
---- n features = 3
ACC: 0.9
---- n features = 4
ACC: 0.8944444444444445
---- n features = 5
ACC: 0.8722222222222221
---- n features = 6
ACC: 0.8944444444444445
---- n features = 7
ACC: 0.861111111111111
---- n features = 8
ACC: 0.8777777777777779
---- n features = 9
ACC: 0.8833333333333334
---- n features = 10
ACC: 0.888888888888889
---- n features = 11
ACC: 0.8833333333333334
---- n features = 12
ACC: 0.8833333333333334
---- n features = 13
ACC: 0.888888888888889
---- n features = 14
ACC: 0.8944444444444445
---- n features = 15
ACC: 0.888888888888889
---- n features = 16
ACC: 0.9166666666666667
---- n features = 17
ACC: 0.9111111111111111
---- n features = 18
ACC: 0.9111111111111111
---- n features = 19
```

```
ACC: 0.9333333333333333
---- n features = 20
ACC: 0.9277777777777778
---- n features = 21
ACC: 0.9333333333333332
---- n features = 22
ACC: 0.9222222222222222
---- n features = 23
ACC: 0.9333333333333332
---- n features = 24
ACC: 0.9222222222222222
---- n features = 25
ACC: 0.9277777777777778
---- n features = 26
ACC: 0.9222222222222222
---- n features = 27
ACC: 0.9166666666666667
---- n features = 28
ACC: 0.9222222222222223
---- n features = 29
ACC: 0.9277777777777777
---- n features = 30
ACC: 0.9388888888888888
Optimal number of features:  30
```



Aqui al igual que en el caso del SVM lineal podemos ver que usando el metodo de SelectKbest conseguimos reducir la dimensionalidad hasta 20 lo cual es muy bueno, pero tambien al hacer varios intentos este resultado fue variando entre 19-21 caracteristicas y en algunos casos solo agarraba 4 caracterisiticas pero esto era muy raro

4.6 Evaluacion del modelo utilizando validacion cruzada (Gaussian NB)

```
data=np.loadtxt("../Class Scripts/activity_data.txt")
x=data[:,1:]
y=data[:,0]
#------Gaussian Naive Bayes

def Select_features(x,y,c):
    n_feats = np.arange(1,31,1)
    acc_nfeat = []

    for n_feat in n_feats:


        acc_cv = []

        kf = StratifiedKFold(n_splits=5, shuffle = True)

        for train_idx, test_idx in kf.split(x,y):
            x_train = x[train_idx,:]
            y_train = y[train_idx]

            clf_cv = GaussianNB(var_smoothing=c)

            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)

            x_test = fselection_cv.transform(x[test_idx, :])
            y_test = y[test_idx]
            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)



    opt_index = np.argmax(acc_nfeat)
    opt_features = n_feats[opt_index]
    return opt_features

kf1 = StratifiedKFold(n_splits=5, shuffle = True)
kf2 = StratifiedKFold(n_splits=5, shuffle = True)
cv_y_test = []
cv_y_pred = []
acc=[]

for train_index, test_index in kf1.split(x, y):
    x_train = x[train_index, :]
    y_train = y[train_index]
    x_test=x[test_index,:]
    y_test=y[test_index]
```

```
    best_c = 0
    best_acc = 0
    for c in [1e-12, 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5,
     ↪  1e-4, 1e-3, 1e-2]:
        acc_hyp_cv = []
        for sub_train_index, val_index in kf2.split(x_train,
         ↪  y_train):
            x_sub_train = x[sub_train_index, :]
            y_sub_train = y[sub_train_index]
            x_val = x[val_index, :]
            y_val = y[val_index]
           Opt_features=Select_features( x_sub_train,y_sub_train,c)
            clf_cv = GaussianNB(var_smoothing=c)
            fselection_cv = SelectKBest(f_classif,k= Opt_features)
            fselection_cv.fit(x_sub_train,y_sub_train)
            x_sub_train = fselection_cv.transform(x_sub_train)
            clf_cv.fit(x_sub_train,y_sub_train)
            x_val=fselection_cv.transform(x_val)
            y_pred = clf_cv.predict(x_val)
            acc_i = accuracy_score(y_val, y_pred)
            acc_hyp_cv.append(acc_i)
        acc_hyp = np.average(acc_hyp_cv)
        if (best_acc < acc_hyp):
            best_c = c
            best_acc = acc_hyp
    best_features=0
    Opt_features=Select_features( x_train,y_train,best_c)
    clf_cv = GaussianNB(var_smoothing=best_c)
    fselection_cv = SelectKBest(f_classif,k= Opt_features)
    fselection_cv.fit(x_train,y_train)
    x_train=fselection_cv.transform(x_train)
    clf_cv.fit(x_train,y_train)
    x_test=fselection_cv.transform(x_test)
    y_pred = clf_cv.predict(x_test)
    acc_f = accuracy_score(y_test, y_pred)
    acc.append(acc_f)

final_acc=np.average(acc)
print(final_acc)
```

```
0.9222222222222222
```

4.7 Conclusión de los resultados obtenidos

Como podemos ver en la parte de optimizacion de hiperparametros los modelos si suelen mejorar sus rendimientos un poco al cambiar entre varios valores del hiperparametro que escogimos como en el caso del SVM que tiene un buen rendimiento con el valor de C = 0. 50 aproximadamente. Tambien en la reduccion de dimensionalidad pudimos ver que en el caso de los dos modelos, usando Select Kbest, estos siempre dan medidas parecidas siendo que se escogen entre 19-21 caracteristicas asi mejorando el rendimiento del modelo. Usando cross validation anidado pudimos ver que el primer modelo consigue un accuracy del 91% lo cual es muy bueno a comparacion del modelo sencillo que consigue un 89% y en el caso del GaussianNB este tambien tiene una mejora pero esta es minima quedandose con un rendimiento de aproximadamente 93%. Cabe recalcar que corrimos muchas veces ambos codigos para asegurarnos que el accuracy fuera constante y

pudimos notar que el acurracy del SVM lineal si variaba mucho siendo que aveces llegaba a un 87%, el mas constante fue GaussianNB que siempre se mantuvo en un rango del 92% al 94%

5 Aplicación en línea

En este apartado estaremos usando el modelo con mejor rendimiento que obtuvimos en la seccion pasada el cual fue el Gaussian NB, esto incluyendo la seleccion de hiperparametros y caracteri-siticas. Con el modelo ya elegido aplicaremos una aplicacion en tiempo real para clasificar los ejercicios de algun usuario.

5.1 Entrenamiento del modelo para aplicacion en linea

```
data = np.loadtxt('../Class Scripts/activity_data.txt')
x = data[:,1:]
y = data[:,0]


#Model Fit --------------------------------
from sklearn.feature_selection import SelectKBest, f_classif,
 ↪  SequentialFeatureSelector, RFE, mutual_info_classif
n_feats = np.arange(1,31,1)
acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_idx, test_idx in kf.split(x,y):
        x_train = x[train_idx,:]
        y_train = y[train_idx]

        clf_cv = GaussianNB()

        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)

        x_test = fselection_cv.transform(x[test_idx, :])
        y_test = y[test_idx]
        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)


fselection_cv_final = SelectKBest(f_classif,k=opt_features)
fselection_cv_final.fit(x,y)
x_final = fselection_cv_final.transform(x)



parameters = {'var_smoothing': [1e-12, 1e-11, 1e-10, 1e-9, 1e-8,
 ↪  1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2]}
clf_cv = GridSearchCV(GaussianNB(), parameters, cv = 5)
clf_cv.fit(x_final, y)
j = pd.DataFrame(clf_cv.cv_results_)
print(clf_cv.best_estimator_)
best_hyper = clf_cv.best_estimator_.var_smoothing
print(best_hyper)


clf_final = GaussianNB(var_smoothing=best_hyper)
clf_final.fit(x_final,y)
```

5.2 Codigo aplicación en línea

El siguiente codigo se ejecuta en un programa especial que recibe los datos en tiempo real de la aplicacion pero este fragmento que adjuntamos aqui es el que calcula las nuevas caracteristicas y las procesa con el modelo que entrenamos anteriormente

```
Excercise = [('Sentadilla', 1), ('Jumping-Jacks', 2), ('Lagartija',
 ↪  3),('Quieto', 4),('Rodillas arriba', 5),('Mountain Climbers',
 ↪  6)]

feat = []
for s in range (last_data.shape[1]):
    sig = last_data[:,s]
    n=len(sig)
    feat.append(np.average(sig)) # Promedio
    feat.append(np.std(sig)) #Desviavón Standard
    feat.append(stats.kurtosis(sig)) #Kurtosis
    feat.append(stats.skew(sig)) #Skewness
    feat.append(stats.median_abs_deviation(sig)) # MAD
    ZCR=0
    for i in range(0,n-1):
        if sig[i]*sig[i+1] < 0:
            ZCR+=1
    ZCR/=n-1
    feat.append(ZCR) # Zero Crossing Rate
    rms = np.sqrt( np.sum(sig**2)/n)
    feat.append(rms)#Root Mean Square
    SSC=0
    for i in range(1,n-1):
        if (sig[i]-sig[i-1])*(sig[i]-sig[i+1]) > 0:
```

```
            SSC+=1
    feat.append(SSC)      #Slope sign changes
    WL= np.sum(np.abs(np.diff(sig)))
    feat.append(WL)  #Waveform Length
    Energy=np.sum(sig**2)
    feat.append(Energy) #Energy


# Build x and y arrays
processed_data =  np.array([feat])
x_online = processed_data


# YOUR classification code here
x_online = fselection_cv_final.transform(x_online)
y_pred_online = clf_final.predict(x_online)

for i,j in Excercise:
    if y_pred_online == j:
        print(i)
```

5.3 Resultados de la aplicación en línea

La aplicacion resulto ser constante con todos los miembros del equipo pero si cambiaba dependiendo de la intesidad con la que se hacia el ejercicio, por ejemplo cuando mi compañero hacia los mountain climbers el codigo arrojaba que eran rodillas arriba pero esto porque lo hacia con menos intensidad y como son movimientos parecidos entonces el modelo no lo clasificaba bien

El rendimiento resulto en lo esperado ya que en la validacion cruzada salio que tenia un 93% de accuracy y en efecto el modelo clasifica muy bien los ejercicios y aunque a veces se tarda un poco en clasificar el ejercicio correctamente en un 90% de los casos el programa los clasifica bien

6 Conclusiones

6.1 José Emilio Martínez Hernández

Durante este trabajo pude aprender muchas cosas importantes acerca de los modelos de aprendizaje supervisado y cómo se pueden usar en situaciones reales. Para empezar, pude aprender mucho acerca de varios modelos, tanto los que vimos en clase como los que tuvimos que investigar por nuestra cuenta. Pude ver las diferencias que hay en cada uno y así poder ver cuál era el que mejor le quedaba a mi conjunto de datos.

Otra de las cosas importantes es que pude aprender más de cómo se maneja la estimación de hiperparámetros y la reducción de dimensionalidad, ya que yo desde un principio pensé que al poner más características iba a sacar un mejor rendimiento, pero al aplicar el K-Best pude ver que estaba equivocado, ya que mi modelo se comportaba mejor con 20 características.

Hablando de la evaluación del modelo, también pude aprender mucho acerca de cómo es que tenemos que hacer *cross-validation* para cada punto de nuestro modelo, tanto para evaluarlo como

para optimizarlo. También pude entender mejor el *cross-validation* anidado, ya que es un concepto un poco complicado de entender y aplicar al principio, pero con este proyecto me quedó más claro cómo aplicarlo para futuros proyectos y así darle más validación a mis modelos de aprendizaje supervisado.

Respecto a la parte final del proyecto, también es importante mencionar que pude aprender acerca de la importancia y las complicaciones que conlleva la recolección de datos para el modelo, ya que muchas veces los datos que recolectábamos salían erróneos o no se capturaban bien, lo que ocasionaba que el modelo tuviera un mal rendimiento.

Respecto a la aplicación en línea, también pudimos aprender un poco del proceso que se lleva a cabo para recopilar las señales desde una aplicación web y, a través de eso, pasarlas por un *pre-processing* para sacar las características más importantes y después de eso predecir el resultado a través de nuestro modelo ya optimizado y entrenado. Cabe recalcar que la fase de decidir las características fue muy importante, ya que tuvimos que investigar un poco más a profundidad qué características eran importantes para este tipo de señales.

En conclusión, puedo decir que este fue un proyecto muy enriquecedor en todos los sentidos, ya que pudimos ver varios temas importantes como aprendizaje supervisado, recolección de datos, optimización de modelos y aplicaciones en línea, todo esto con un enfoque en aplicaciones médicas, que hoy en día es un tema muy importante y muy estudiado en el campo de la IA.

6.2 Josué Tapia Hernández

Primero se realiza el data preprocessing y se extrae que datos son los que se van a utilizar para realizar la predicción, en este caso tomamos medidas de Tendencia en relación a señales ya que eso se capturó con la aplicación Phyphox.

Después se prueban varios modelos con los parámetros en default y después de esos modelos elegimos los que tenían de las mejores accuracies, pero que también pudiéramos ajustar los hiper parámetros así como también realizar un feature selection para mejorar aún más al modelo.

Cabe mencionar que la evaluación de estos modelos se realizó con K-fold cross validation y también K-fold anidado lo cual se realiza cuando el conjunto de datos no es tan grande como se espera y esto hace que la evaluación del modelo sea más exigente y no se produzca overfit.

Después de estos resultados entrenamos el modelo con todo el conjunto de datos para después hacer la clasificación en línea y al final mientras el usuario esta realizando un movimiento esta ventana la recibe nuestro programa realiza las transformaciones y después predice qué movimiento está realizando el usuario.

En conclusión aprendí sobre el ajuste de modelos y cómo realizar su evaluación correctamente. Así como la manera de elegir las features y como poner en práctica el modelo con la clasificación en línea.

Referencias

Altin, Cemil, y Orhan Er. n.d. «Comparison of Different Time and Frequency Domain Feature Extraction Methods on Elbow Gesture's EMG». *European Journal of Interdisciplinary Studies*, n.d. https://revistia.com/files/articles/ejis_v2_i3s_16/Cemil.pdf.

Lee, Sarah. 2025. «Deep dive & look at median absolute deviation». https://www.numberanalytics.com/blog/deep-dive-median-absolute-deviation#google_vignette.

Sanjuán, Francisco Javier Marco. 2024. «¿Qué es la curtosis? Definición, tipos, medidas y ejemplos». mayo de 2024. https://economipedia.com/definiciones/curtosis.html.

Sara Abbaspour, Hamid Gholamhosseini, Maria Lindén. 2025. «Optimizing the impact of time domain segmentation techniques on upper limb EMG decoding using multimodal features». *PLoS ONE* 20 (5): e0322580. https://doi.org/10.1371/journal.pone.0322580.