TUTORIAL ANÁLISIS DE PATRONES GITA

Luis Castillo Chicaiza
UNIVERSIDAD DE ANTIOQUIA

RESUMEN

Este tutorial se enfoca en el análisis de patrones en Python, comenzando con el preprocesamiento de audio, hasta entrenar clasificadores y evaluar su desempeño reconociendo patrones. Se exploraron métodos clave como *PCA, LDA, SVM, KNN, Grid-Search, cross-validation* y *Medidas de desempeño*, aplicados a datos proporcionados. El proyecto guio a los participantes a través de la implementación de estos métodos, complementando el proceso con herramientas de evaluación como la matriz de confusión y la curva ROC. Con un enfoque práctico, el tutorial proporciona una comprensión integral del análisis de patrones, fortaleciendo las habilidades en el manejo de datos y machine learning con Python.

1. INTRODUCCIÓN

El desarrollo de este tutorial está dividido en dos etapas fundamentales, cada una ayudando a la comprensión y manejo de los distintos algoritmos empleados para el procesamiento y clasificación de características de audios.

La primera etapa aborda la adquisición y procesamiento de un audio. En este proceso se observa la importancia de la normalización con el fin de garantizar un estándar a la hora de procesarlo y compararlo con otros audios, esto, para su posterior implementación en clasificadores, permitiendo explorar características propias de este.

La segunda etapa se centra en la caracterización de datos ya procesados previamente dados, para luego, a partir de estos archivos emplear técnicas avanzadas como *PCA* (Principal Component Analysis) y *LDA* (Linear Discriminant Analysis) para reducir la dimensionalidad y resaltar las características relevantes de estos datos (High emotions, Low emotions); posteriormente, en la etapa de clasificadores, donde a los datos ya caracterizados se les aplica métodos de clasificación como *KNN* (*K-nearest neighbors)* y *SVM* (*Support vector machine*), además de estrategias como *Grid-Search* y *cross-validation* con el fin de avanzar a la última etapa, la cual calcula las medidas de desempeño utilizando la matriz de confusión acompañadas de la *curva ROC* que proporciona una visión completa del rendimiento de los modelos desarrollados.

2. MARCO TEÓRICO

2.1 Adquisición de datos

La Adquisición de datos es el primer paso crucial para los proyectos de análisis de patrones. Este proceso implica la recolección de información de audios. Existen diversas técnicas de adquisición de datos, como el uso de micrófonos especializados, y es importante destacar que la calidad de los datos es fundamental, ya que influye en la efectividad de las etapas posteriores.

2.2 Normalización

La normalización de datos busca estandarizar los datos, es decir, tener una media de cero, y una desviación estándar de uno, que, en caso de audios, esto es estandarizar la amplitud y la escala de la señal acústica, con el fin de garantizar coherencia y comparabilidad entre las muestras. Esta es una etapa esencial para eliminar sesgos relacionados con variaciones de los datos, facilitando un análisis consistente y preciso en las siguientes etapas.

2.3 Cross-Validation

K-Fold o cross-validation es una técnica importante en el ámbito del aprendizaje automático. Su propósito principal es evaluar la capacidad de generalización de un modelo de manera robusta para reducir el riesgo de sobreajustes. Este, funciona dividiendo el conjunto de datos en *k* folds y realizando *k* iteraciones donde en cada una se toman diferentes datos de *train* y *test*, esto para garantizar el uso completo de los datos y así tener un mejor desempeño ante la variabilidad que puedan presentar.

2.4 Algoritmos de reducción de dimensión de datos

2.4.1 Principal Component Analisys (PCA)

PCA es una técnica estadística utilizada para transformar un conjunto de datos de alta dimensión (características o variables originales en el conjunto de datos) a un nuevo sistema de coordenadas donde predominan las dimensiones principales que son llamadas componentes las cuales capturan la mayor variabilidad de los datos, es decir, se retiene la mayor cantidad de información posible.

2.4.2 Linear Discriminant Analysis (LDA)

LDA es una técnica estadística y de aprendizaje supervisado usado para encontrar la combinación lineal de características que maximiza la separación entre diferentes clases de un conjunto de datos. A diferencia

de PCA que busca la variabilidad global, LDA se centra en maximizar la variabilidad de las clases y minimizar la variabilidad dentro de las mismas.

2.5 Clasificadores

2.5.1 K-nearest neighbors (KNN)

Es un algoritmo de aprendizaje supervisado que se utiliza para tareas de clasificación y regresión. KNN opera mediante la identificación de las *k* muestras más cercanas a un punto de prueba en el espacio de características para determinar su clase. Este algoritmo se destaca por su simplicidad y versatilidad.
KNN está adaptado para recibir parámetros definidos con el fin de adaptarse a los datos, en este caso, por ejemplo, el parámetro *k*, que es el número de vecinos considerados para clasificar una muestra de una clase u otra.

2.5.2 Support vector machine (SVM)

Es un algoritmo de aprendizaje supervisado usado para clasificación y regresión. En términos simples, en un problema de clasificación, una SVM busca la mejor manera de dividir un conjunto de datos en categorías mediante hiperplanos, la elección de este hiperplano se realiza maximizando la distancia entre las instancias más cercanas de las clases, conocidas como vectores de soporte.

Al igual que KNN este tiene diversos parámetros a modificar en base a lo que mejor se acomode al conjunto de datos sin hacer un sobre ajuste del modelo, como por ejemplo los parámetros:

- *C*: Controla la tolerancia a los errores de clasificación, o el parámetro.
- *Gamma*: Controla la forma de la función de decisión
- *Kernel*: Define el tipo de transformación que se aplica a los datos

2.6 Medidas de desempeño

2.6.1 Matriz de confusión

La matriz de confusión es una herramienta de evaluación de rendimiento de modelos de clasificación. Esta matriz, permite analizar el desempeño del modelo al comparar sus predicciones realizadas con las clases verdaderas de un conjunto de datos. En ella, se organizan las predicciones principalmente en cuatro categorías:

- Verdaderos positivos (VP): Instancias correctamente clasificadas como positivas.
- Verdaderos negativos (VN): Instancias correctamente clasificadas como negativas.
- Falsos positivos (FP): Instancias incorrectamente clasificadas como positivas.
- Falsos negativos (FN): Instancias incorrectamente clasificadas como negativas.

A través de estos datos, obtienen diferentes medidas de rendimiento, y gracias a estas, se permite ver una evaluación completa del modelo, por ejemplo:

- *Sensitivity*: Mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo y está definida por la ecuación

$$Sens = \frac{VP}{VP + FN}$$

- *Specificity*: Mide la proporción de instancias negativas que fueron correctamente identificadas por el modelo y está definida por la ecuación

$$Spec = \frac{VN}{VN + FP}$$

2.6.2 Curva ROC

Es una herramienta que proporciona una representación gráfica de la relación entre la tasa de verdaderos positivos y falsos positivos en diversos umbrales de clasificación.

La curva ROC ilustra el desempeño del modelo a lo largo de diferentes niveles de sensibilidad y especificidad, donde, la línea de referencia que representa un área bajo la curva de *0.5* significa que el modelo tiene una clasificación aleatoria, lo cual no es lo ideal, por lo contrario, una curva que abarque la máxima cantidad de área posible significa que el modelo tiene un funcionamiento ideal. Esta, entrega una representación clara y visualmente informativa de la capacidad de un modelo de clasificación para tomar decisiones acertadas.

3. EXPERIMENTOS Y RESULTADOS

3.1 Etapa de procesamiento de audio

En esta etapa se realiza la importación de un audio previamente dado (Fig. 1), al cual se debe normalizar y eliminar la señal DC, con el fin de procesar y estandarizar este mismo, para comparar y manipular posteriormente con mayor facilidad. (Fig. 2)

Cabe destacar que para este proceso se emplea Python con las librerías de Pydub para importar y leer el archivo de audio, MatPlotLib para graficar los audios (Original y

procesado), y numpy para realizar la normalización y la eliminación de la señal DC.

Figura 1. Gráfica de audio original

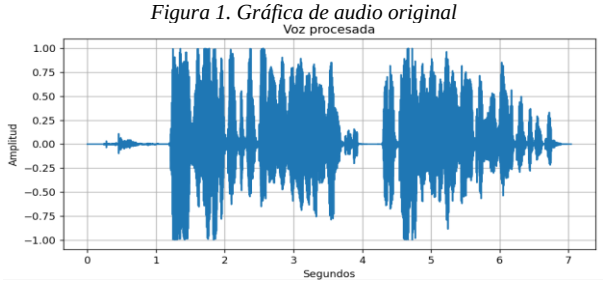

Figura 2. Gráfica de audio procesada

Como se puede observar en la gráfica del audio procesado (Fig. 2) en comparación al audio original (Fig. 1), la amplitud cambia, además de verificar que no se encuentre la señal DC, lo que indica la correcta normalización del audio.

3.2 Etapa de caracterización

Inicialmente, se proporciona dos archivos con datos ya procesados, uno de "High Emotions", y otro de "Low Emotions", donde ambos tienen 6373 características, pero diferente número de muestras, que en este caso serían, 313 para *High Emotions*, y 222 para *Low Emotions*.

3.2.1 Estandarizar datos

Para esta primera etapa se importa la librería de numpy, con la cual uniremos los dos archivos .txt en uno solo, con el fin de tener un solo conjunto de datos con el que trabajar. (Fig. 3)

Luego, se importa la librería de *sklearn* y usamos el método de *StandartScaler* el cual calcula la media y la desviación estándar del conjunto de datos, para posteriormente aplicar el método *.fit_transform()* que resta la media calculada y la divide por la desviación estándar, lo que resulta en una distribución de datos con una media de cero y desviación estándar de uno. (Fig. 3). Cabe destacar, que se hace la normalización en este punto ya que se debe aplicar *PCA* posteriormente, de lo contrario, se haría esto después del proceso de división de datos.

```
# Unir matrices
data = np.vstack((data_high, data_low))
# Estandarizar los datos, Normalizacion de caracteristicas
data_normalized = StandardScaler().fit_transform(data)
```

Figura 3. Código para estandarizar datos

3.2.2 Reducción de dimensionalidad

Para esta parte se opta por usar PCA como método para recudir la dimensionalidad de los datos, por lo que se usa la librería de *sklearn* y se importa el método *PCA*, además de especificar que se trabajará con dos componentes principales, las cuales son:

- High emotions
- Low emotions

Para posteriormente realizar nuevamente una normalización de los datos para el correcto funcionamiento. También se observa que se agrega un vector de etiquetas, el cual valida a que clase pertenece cada muestra, siendo cero "Low Emotions" y uno "High Emotions" (Fig. 4)

```
# Reducción de dimensionalidad
pca = PCA(n_components=2)
# ext = variable caracteristicas reducidas
X = pca.fit_transform(data_normalized)

Y = np.zeros((len(data),))
Y[0:len(data_high)-1] = 1
```

Figura 4. Código para reducción de dimensionalidad

Luego de aplicar PCA a los datos, se obtiene una gráfica la cual representa la distribución de las dos componentes mencionadas, esto, para tener una vista clara y representativa de estas (Fig. 5)

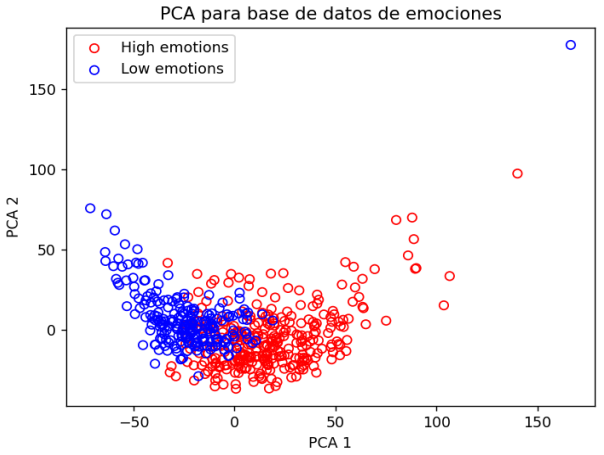

Figura 5. Gráfica PCA

En la gráfica de PCA (Fig. 5), se identifican las tendencias continuas en la distribución de los datos. La concentración de puntos en cierta región del gráfico indica la existencia de patrones específicos de cada

componente. Sin embargo, también se observan algunos puntos dispersos, lo que sugiere que sean datos atípicos.

3.2.3 División de los datos luego de PCA

Antes de pasar a la etapa de clasificadores, se debe realizar una división de los datos, tanto para *Train* como para *Test* de estos.
Inicialmente, se importa desde la librería de *sklearn* el método *KFold*, que es una técnica de validación cruzada que se usa para dividir el conjunto de datos en *k* subconjuntos o *folds* de igual tamaño, además de aleatorizar los datos.

```
# Dividir datos
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

Figura 6. Código para dividir datos

En la línea de código empleada (Fig. 6), se observan diferentes parámetros, donde *n_splits* se refiere a la cantidad de *folds* a usar. Por otro lado, *shuffle* se lo define como verdadero con el fin de que se aleatoricen los datos, y por último *random_state* se define como una semilla desde la cual el método utiliza para iniciar a aleatorizar los datos.

Luego, se realiza un ciclo *for*, el cual itera sobre la cantidad de *splits* antes mencionada, para obtener los datos de *Train* y de *Test* correspondientes para el Split. (Fig. 7)

```
# Realizar cross-validación de los datos
for train_index, test_index in kf.split(X):

    # Conjuntos de Train y test
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
```

Figura 7. Código ciclo cross-validation

Una vez se tiene los datos de *Train* y *Test*, deben ser normalizados ya que, si no se lo hace, estarían normalizados a partir de la información de ambos conjuntos, con lo cual puede afectar a la validez de la evaluación del modelo.

En conclusión, cada iteración tomará datos diferentes de *train* y *test* con el fin de abarcar la totalidad de los datos en la última iteración, esto, para poder tener un mejor desempeño ante la varianza de los datos en general. También antes de llevar los conjuntos de datos al clasificador, se debe normalizarlos respecto a los datos de entrenamiento, porque así, no se tomará en cuenta la media ni la desviación estándar del conjunto de *Test*, lo cual está correcto, porque este conjunto son datos que en teoría no se conocen y se usarán para evaluar el modelo y la efectividad del mismo respecto a lo que ya se conocía (*Train*). (Fig. 8)

```
# Normalizar los datos Train, Test
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

Figura 8. Código normalización Train y Test

3.2.4 Clasificadores

En este punto, después de dividir los datos originales, aplicar cross-validation, y normalizar los datos de *Train* y *Test*, se procede a inicializar dos clasificadores distintos y a entrenarlos, los cuales son *K-nearest neighbors (KNN)*, y *Support vector machine (SVM). (Fig. 9)*

```
# Clasificadores
svm = SVC(kernel='rbf')
knn = KNeighborsClassifier()
```

Figura 9. Código inicialización clasificadores

Como se puede observar en la Fig. 9, para el clasificador SVM, se opta solamente por un *kernel "rbf"*, y los demás parámetros como el *gamma*, *C*, y en caso de KNN el parámetro *n_neighbors*, estarán variando dentro de una malla de búsqueda (Grid-Search).

3.2.5 Grid-Search

Esta herramienta se importa desde la librería de *sklearn* con el método *GridSearchSV*, el cual se encarga de realizar internamente una evaluación con un diccionario de parámetros entregada manualmente (Fig. 10), donde al final devuelve los valores con los cuales el clasificador obtuvo un mejor rendimiento.

```
param_grid = {'C': [0.1, 0.5, 1, 5, 10],
              'gamma': [1e-2, 0.1, 1, 10]}

param_grid_knn = {'n_neighbors': [1, 3, 5, 7, 9, 13, 15, 17, 21]}
```

Figura 10. Parámetros para evaluar

A demás de los parámetros que se entregan para evaluar, *Grid-Search* posee varios parámetros dentro del método, como una *cross-validation* interna, y un *scoring*, el cual nos permite ajustar sobre qué medida de desempeño se quiere buscar los mejores valores, que en este caso se busca el mejor *accuracy*. (Fig. 11)

```
# Grid-Search
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
grid_search_knn = GridSearchCV(
    knn, param_grid_knn, cv=5, scoring='accuracy')

# Entrenar clasificador
grid_search. fit(X_train_normalized, Y_train)
grid_search_knn.fit(X_train_normalized, Y_train)
```

Figura 11. Código GridSearch

Al tener dos clasificadores con el fin de comparar resultados y medidas de desempeño, a ambos, se les aplica una malla de búsqueda independiente.

Con esto, también obtenemos los mejores valores para los parámetros de los clasificadores en el *Fold* que esté iterando en ese momento, para después agregarlos a una lista la cual al final tendrá todos mejores parámetros retornados por el método de *GridSearch* en cada Fold, esto, con el fin de sacar la moda de estos para tener un mejor desempeño general, que funcione mejor a la variabilidad de los datos. (Fig. 12)

```
# Obtener Mejor modelo
best_param = grid_search.best_estimator_
best_param_knn = grid_search_knn.best_estimator_

# Predicciones
Y_pred = best_param.predict(X_test_normalized)
params = grid_search.best_params_
moda_gamma.append(params['gamma'])
moda_c.append(params['C'])

Y_pred_knn = best_param_knn.predict(X_test_normalized)
params_knn = grid_search_knn.best_params_
moda_k.append(params_knn['n_neighbors'])
```

Figura 12. Código para guardar los mejores valores

Al implementar todo este código al conjunto de datos y aplicar la cross-validation ya antes mencionada, se obtiene estos resultados para los parámetros de *gamma*, *C*, *n_neighbors.* (Tabla 1)

Parámetro	C	Gamma	k
Valor	1	1	17

Tabla 1. Mejores parámetros

3.2.6 Medidas de desempeño

3.2.6.1 Matriz de confusión: Para desarrollar una matriz de confusión general de todos los Fold, al igual que los parámetros de cada iteración se guardan en variables (Fig.12), las medidas del *accuracy*, *sensitivity*, *specificity* de ambos clasificadores, para posteriormente hacer una media de estas, obteniendo así estos valores aproximados sobre el rendimiento del modelo en general. (Tabla 2, 3)

	Accuracy	*Sensitivity*	*Specificity*
Valor	85.97	86.66	85.59
Desviación	3.61	4.26	6.65

Tabla 2. Medidas de desempeño SVM

	Accuracy	*Sensitivity*	*specificity*
Valor	87.11	86.66	89.30
Desviación	2.42	3.79	4.62

Tabla 3. Medidas de desempeño KNN

Antes de analizar los resultados (Tabla 2, 3), es importante recordar que la métrica *sensitivity* indica el porcentaje de casos positivos que el modelo logró identificar, mientras que *specificity* indica el porcentaje de casos negativos que el modelo logró identificar. A demás de esto, la desviación estándar es una medida estadística propia de cada métrica lo que indica la cantidad de variabilidad en los resultados.

Ahora, al observar y comparar estos resultados de los dos clasificadores (Tabla 2, 3), teniendo en cuenta la desviación de cada uno, se puede decir que se tiene un buen desempeño en ambos clasificadores.

3.2.6.2 Curva ROC: Adicionalmente a los datos obtenidos mediante la matriz de confusión, se hace la curva ROC para cada clasificador con el fin de observar gráficamente el funcionamiento de los modelos.

Cabe destacar, que para el clasificador SVM, se emplean dos métodos para determinar el AUC (Area bajo la curva), que son:

- *Predict.proba()*
- *Decisión_function()*

Esto, con el fin de experimentar y comprobar si se da algún cambio en los resultados del clasificador SVM con distintos métodos de calcular el AUC. Por otro lado, para KNN, solamente se hace uso del método *predict.proba()*, esto, ya que el método *decisión_function()* solamente se usa en SVM. De este modo, estos son los resultados sobre el AUC, de los diferentes clasificadores. (Tabla 4, 5)

	Predict.proba()	*Decisión_function()*
AUC	0.926	0.925

Tabla 4. AUC en clasificador SVM

	Predict.proba()
AUC	0.926

Tabla 5. AUC en clasificador KNN

Como se observa en las tablas (Tabla 4, 5), el AUC de los clasificadores es prácticamente el mismo, lo que indica el correcto funcionamiento de ambos.

Ahora, al graficar las distribuciones, y las curvas ROC, de los clasificadores, se obtienen los siguientes resultados (Fig. 13-16)

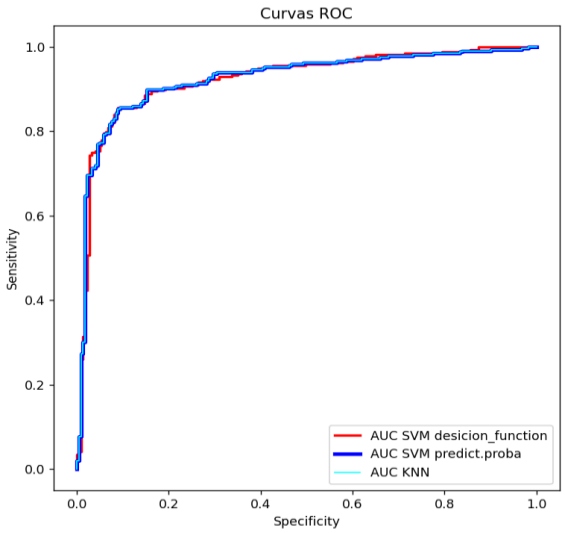

Figura 13. Curvas ROC clasificadores

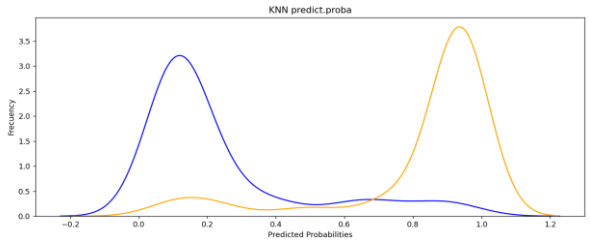

Figura 14. Distribución KNN

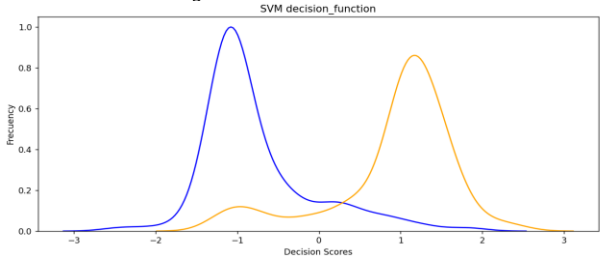

Figura 15. Distribución de SVM con decisión_function()

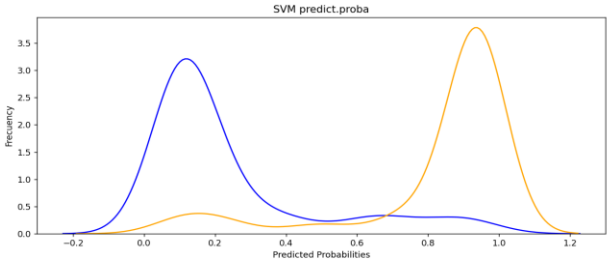

Figura 16. Distribución de SVM con predict.proba()

Gracias a la gráfica de las curvas ROC (Fig. 13), se observa un desempeño similar en los tres modelos realizados, pero, esto se corrobora en las gráficas de distribuciones de cada uno de estos, aunque existen detalles a denotar, por ejemplo, los modelos empleados con *decisión_function()* (Fig. 14, 15) tienen una escala en el eje de la frecuencia diferente a la empleada por *predict.proba()* (Fig. 16), pero a su vez, teniendo un AUC similar, lo que indica las diferencias internas de interpretación de cada método al momento de calcularlo. Además, se observa que los tres modelos, tienen una capacidad de clasificación ideal, esto, se apoya además de la curva ROC con las gráficas de distribución, por su clara separación de las curvas.

De este modo, estas son las curvas, y las distribuciones correspondientes a los valores previamente mencionados (Tablas 4, 5), de donde se puede concluir la coherencia que tienen estos con los datos comprobados mediante la matriz de confusión (Tablas 2, 3)

4. CONCLUSIÓN

En este tutorial se explora el procesamiento de datos (audios) mediante dos clasificadores como lo son SVM y KNN, además de obtener el conocimiento necesario y destacando la importancia de las técnicas de normalización, estandarización de datos, y reducción de dimensionalidad con PCA y LDA, para tener una mejor eficiencia y preservar la información lo máximo posible de los datos originales

En el entrenamiento de los clasificadores se llevaron a cabo diferentes herramientas como *Grid Search* o *cross-validation* con el fin de explorar diferentes combinaciones de parámetros y poder generalizar y evaluar los modelos, asegurando un mejor desempeño frente a la variabilidad de los datos

En las medidas de efectividad de los clasificadores se adoptan medias cruciales de desempeño como la matriz de confusión para ver una información detallada de los aciertos y errores del modelo, o las curvas ROC para evaluar las capacidades de los modelos en distintos umbrales de decisión

En resumen, el tutorial proporciona una guía muy útil y comprensiva desde la adquisición y procesamiento de datos, hasta la aplicación y evaluación de técnicas avanzadas de clasificación. El uso de SVM y KNN, a demás de diversos métodos para calcular su efectividad, junto con herramientas como Grid Search o las ayudas que nos entrega la librería de Python *sklearn* en el análisis detallado del desempeño de los modelos, así, esto nos ofrece una sólida base para abordar proyectos de clasificación en el campo del análisis y reconocimiento de patrones en datos.

5. REFERENCIAS

[1] "Comprende Principal Component Analisys". Aprendemachinelearning. [En línea].

Disponible: https://www.aprendemachinelearning.com/comprende-principal-component-analysis/

[2] "3.1. Cross-validation: evaluating estimator performance". scikit-learn. [En línea]. Disponible: https://scikit-learn.org/stable/modules/cross_validation.html

[3] R. Canadas. "Curvas ROC | Usos de ROC-AUC en machine learning". abdatum. [En línea]. Disponible: https://abdatum.com/ciencia/curvas-roc#google_vignette

[4] "Evaluación de modelos: Dibujo y AUC Cálculo de curvas ROC - programador clic". programador clic. [En línea]. Disponible: https://programmerclick.com/article/26342191461/#google_vignette

[5] "Descubra el algoritmo KNN : un algoritmo de aprendizaje supervisado". Formación en ciencia de datos | DataScientest.com. [En línea]. Disponible: https://datascientest.com/es/que-es-el-algoritmo-knn

[6] "Linear Discriminant Analysis (LDA) | Interactive Chaos". Home page | Interactive Chaos. [En línea]. Disponible: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/linear-discriminant-analysis-lda

[7] Colaboradores de los proyectos Wikimedia. "Máquinas de vectores de soporte - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. [En línea]. Disponible: https://es.wikipedia.org/wiki/Máquinas_de_vectores_de_soporte

[8] "sklearn.metrics.confusion_matrix". scikit-learn. [En línea]. Disponible: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[9] "Espacio de recursos de ciencia de datos". Espai de recursos de ciència de dades. [En línea]. Disponible: https://datascience.recursos.uoc.edu/es/preprocesamiento-de-datos-con-sklearn/