Clasificación de canciones en base al artista aplicando KNN sobre un FMA Dataset.

Universidad de Cuenca
Facultad de Ingeniería, Escuela de Sistemas
Abad Freddy, Cabrera Edwin, Càrdenas Paola.
freddy.abadl@ucuenca.ec, edwin.cabrera@ucuenca.ec , paola.cardenas0108@ucuenca.ec

ABSTRACT
This document describes an application exercise of the KNN (K Near Neighbors) algorithm, it refers to a supervised nonparametric classification method that is used to estimate the probability density function or directly the posterior probability $F(x/C_j)$ of the predictors x for each class C_j. Furthermore, during the learning process does not make any assumptions about the distribution of the predictor variables. A brief description of the functioning of the algorithm is presented and this is illustrated with a brief demonstration example, afterwards an analysis of a dataset with a great amount of musical information is carried out, having as objective the classification by these songs by the id of the artist, this with the application of the KNN algorithm, the same one that is implemented, executed and evaluated in several ways to obtain the best possible results. This report will explain the application of this algorithm and its respective performance through several metrics, to know which of them is the best.

Keywords: K Near Neighbors, knn, python, weka, performance, cross-validation.

RESUMEN
El presente documento describe un ejercicio aplicativo del algoritmo KNN (K vecinos próximos), este hace referencia a un método de clasificación supervisada no paramétrico que sirve para estimar la función de densidad de probabilidad o directamente la probabilidad a posteriori $F(x/C_j)$ de las variables predictoras x para cada clase C_j; además durante el proceso de aprendizaje no hace ninguna suposición acerca de la distribución de las variables predictoras; Se presenta una breve descripción del funcionamiento del algoritmo y este es ilustrado con un breve ejemplo demostrativo, posteriormente se realiza un análisis de un dataset con gran cantidad de información musical, teniendo como objetivo la clasificación por de estas canciones por el id del artista, esto con la aplicación del algoritmo KNN, el mismo que es implementado, ejecutado y evaluado de varias maneras para obtener los mejores resultados posibles. El presente informe explica la aplicación de este algoritmo y su respectivo rendimiento mediante varias métricas, para conocer cual de todas es la mejor.

Palabras clave: K Near Neighbors, knn, python, weka, rendimiento, validación cruzada.

1. INTRODUCCIÓN

La gran cantidad de datos disponibles en varios ámbitos de la sociedad requiere ser analizada y transformada en información útil; en el caso del contenido multimedia específicamente en una biblioteca musical la información es clasificada acorde a los gustos de los clientes, géneros, artistas, etc. Esta clasificación se hace en base a similitudes entre los diferentes atributos que posee una canción la cual se distingue como un datapoint del dataset, los algoritmos de machine learning son los ideales para el procesamiento de grandes cantidades de información, el algoritmo KNN (K vecinos próximos) permite clasificar elementos en función de densidad de probabilidad, por lo que en este documento se plantea como objetivo principal la aplicación de dicho algoritmo sobre el dataset de FMA (Archivo de Música Libre)[1] para medir el rendimiento, la precisión de clasificación(accuracy), para ello se prueban las distintas métricas de distancia que se pueden usar en el algoritmo y para un mejor resultado se usa la técnica de validación cruzada.

2. MARCO TEÓRICO

El algoritmo KNN (k vecinos más próximos) es un método de clasificación supervisada basada en las clases de K data points cercanos a un datapoint. En la figura 1 se ejemplifica la lógica del algoritmo, se pretende asignar una clase al elemento evaluado (círculo verde) en base a los 5 elementos más cercanos, como existen más cuadrados azules que triángulos rojos, el círculo verde pertenece al tipo de cuadrados azules.

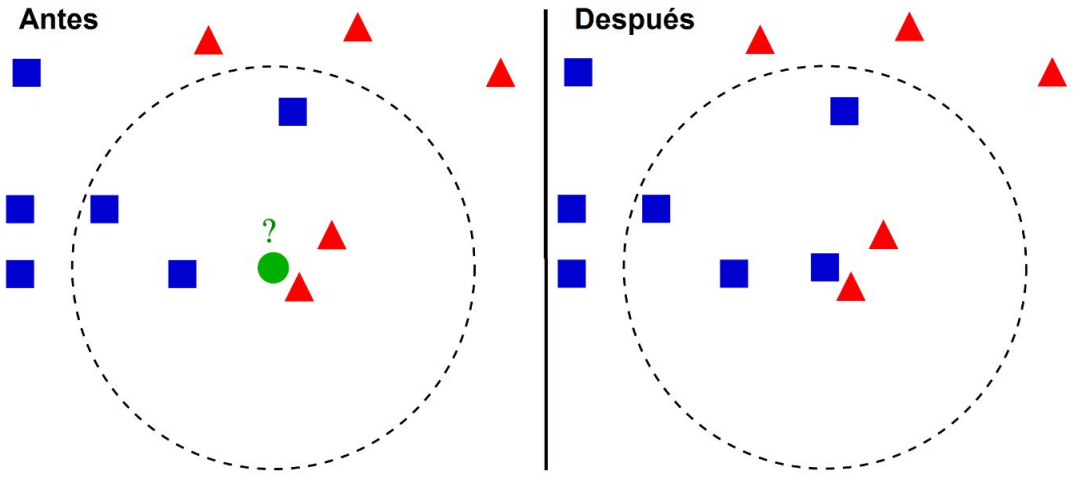

Figura 1: Ejemplo de aplicación de algoritmo de clasificación KNN.

El algoritmo KNN en sí es bastante sencillo de explicar ya que se puede resumir en los siguientes pasos:

1. Se elige la cantidad de k vecinos próximos y una métrica de distancia.
2. Encuentra los k vecinos más cercanos de la muestra que se desea clasificar.
3. Asigna la etiqueta de la clase por mayoría de votos (N° de integrantes del grupo pertenecientes a esa clase).

3. MÉTODO Y MATERIALES

Previo a la aplicación del algoritmo se describen las herramientas y métodos necesarios para la ejecución del mismo y un mayor entendimiento de los procesos del algoritmo.

3.1 Materiales

La librería Scikit-learn es una librería que provee algoritmos usados en Machine Learning para facilitar tareas de regresión, clasificación, predicción y clustering, se usan mayormente en minería de datos. El algoritmo KNN se encuentra en la función *KNeighborsClassifier* que cuenta con los siguientes parámetros:

- **N_neighbors:** Indica el número de vecinos, por defecto el número de vecinos es 5.
- **Weights:** Permite aplicar pesos uniformes sobre cada datapoint o ponderar los datapoints en base a su distancia.
- **Algorithm:** Permite establecer el tipo de algoritmo que puede ser ball_tree o kd_tree o brute (para aplicar fuerza bruta). Además de ello, existe la opción auto que selecciona el algoritmo más apropiado basándose en los valores pasados al método fit.
- **Leaf_size:** Establece el tamaño de las hojas aplicadas en el algoritmo (en caso de seleccionar el algoritmo BallTree o KDTree). El número de hojas óptimo dependerá de la problemática del problema
- **p:** Indica el valor de poder utilizado al aplicar la métrica de distancia *minkowski*, si p=1 la métrica *minkowski* es equivalente a la distancia *manhattan*
- **Metric:** Establece la métrica para calcular la distancia, por defecto se utiliza la métrica *minkowski* equivalente a la métrica Euclidiana estándar. Las métricas disponibles se presentan en la figura 2 :

identifier	class name	args	distance function
“euclidean”	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
“manhattan”	ManhattanDistance	•	`sum(|x - y|)`
“chebyshev”	ChebyshevDistance	•	`max(|x - y|)`
“minkowski”	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
“wminkowski”	WMinkowskiDistance	p, w	`sum(w * |x - y|^p)^(1/p)`
“seuclidean”	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
“mahalanobis”	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

Figura 2 : Métricas para calcular la distancia entre datapoints.

- **Metric_params:** Contiene parámetros necesarios dependiendo de la métrica seleccionada.
- **N_jobs:** Número de operaciones paralelas que se ejecutan para construir el árbol de decisión.

3.2 Ejecución de ejemplos

Para un mayor entendimiento del algoritmo se ha replicado un ejemplo del libro Raschka, Python machine learning [2], el ejemplo implementa un modelo KNN en scikit-learn usando

una métrica de distancia euclidiana, parte principal de la aplicación del código se encuentra en Raschka[2]. El código necesario se puede observar en la figura 3.

```
# Dividir datos en 70% de entrenamiento y 30% de datos de prueba:
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Estandarizando las características:
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

from sklearn.neighbors import KNeighborsClassifier
#Configurando el algoritmo knn
#Etsbleciondo la metrica de distancia euclidiana
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
#ejecucion de knn
knn.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std, y_combined, classifier=knn, test_idx=range(105,150)
```

Figura 3: Código fuente de knn con métrica euclidiana y 5 vecinos

Al especificar cinco vecinos en el modelo KNN para un conjunto de datos, se obtiene un límite de decisión relativamente uniforme, como se muestra en la figura 4.

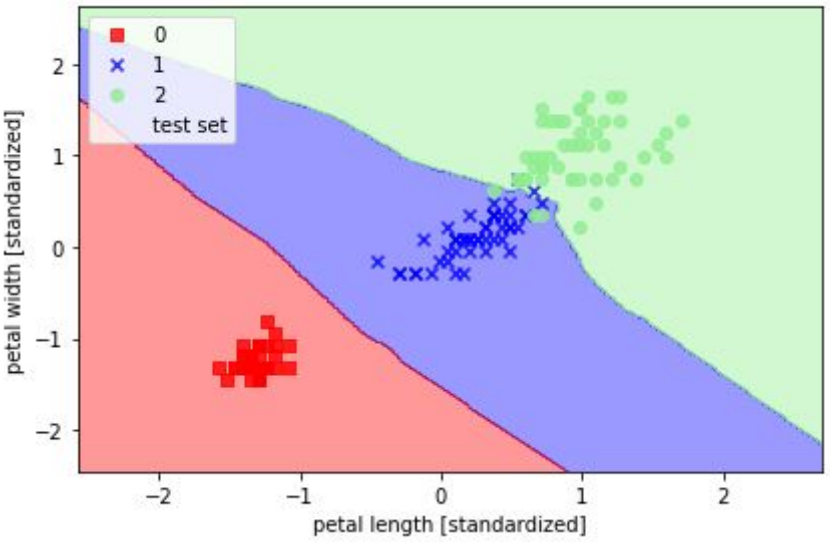

Figura 4: Resultado de knn aplicado sobre el data-set de Iris Setosa [2]

4. PROCEDIMIENTO PARA APLICACIÓN DE KNN SOBRE FMA DATASET

4.1 Descripción del dataset.

El dataset utilizado es el archivo *features.csv*[1] que tiene 106574 registros de pistas de audio con 518 atributos que conforman las características de audio entre las cuales tenemos:

- Centroide espectral

- Ancho de banda espectral
- Contraste espectral
- Coeficientes Cepstrales en las Frecuencias de Mel

Además de ello, el archivo *metadata.csv* [1] cuenta con información de cada pista relacionada con el nombre de la canción, artista relacionado, fecha de lanzamiento, entre otros. Para generar el dataset, se vinculó el nombre del artista del archivo *metadata.csv* con el archivo *features.csv* utilizando la herramienta *Microsoft Excel*.[5]

Ya que el algoritmo KNN requiere calcular las distancias entre cada datapoint con respecto a otro datapoint, para realizar una clasificación, esto requiere de una gran capacidad computacional, es por ello que se procedió a tomar una muestra del dataset, para ello se utilizó la herramienta *IBM SPSS Statistics 23*[6], para la generació un dataset *muestra.csv* que contiene 16032 instancias aleatorias que conforman el 15% del dataset original. En la figura 5 se muestra la configuración de *IBM SPSS Statistics* que permite establecer la cantidad de instancias que se pueden cargar de un documento de forma sistemática o aleatoria.

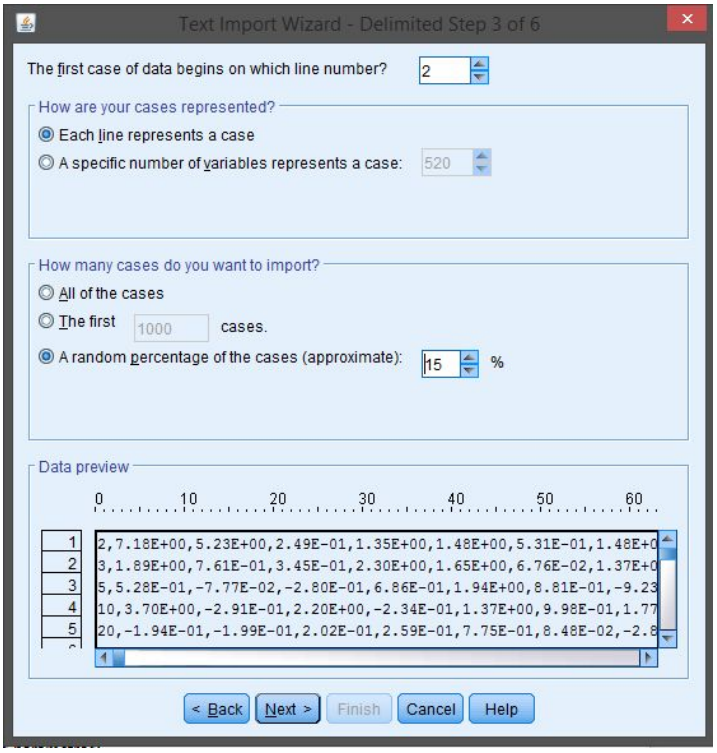

Figura 5: Ventana para establecer la cantidad de datos en SPSS Statistics [6]

4.2 Aplicación e implementación del algoritmo.

Durante el experimento se aplicarón las siguientes condiciones:

- El número de vecinos variará de 2 en 2 (1, 3, 5 , …).
- Métrica aplicada: Minkowski
- Valor de p: 2

Para asegurarse que los valores de error no son resultado de la aleatoriedad se aplicó validación cruzada (cross-validation) con un número de particiones igual a 10 (se puede hacer 40 particiones, pero habrán semejantes resultados). El código utilizado se presenta en la Figura 6 .

```
myList = list(range(1,20))          #Creando lista de numeros impares para KNN
neighbors = filter(lambda x: x % 2 != 0, myList)        #Subconjunto Impares
cv_scores = []          #Lista vacia que tendra Puntajes CV

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')#Validacion Cruzada 10 veces
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]        #Se obtiene error de clasificacion
#print (MSE)

optimal_k = neighbors[MSE.index(min(MSE))]      #Determinar el mejor k
print ("El numero optimo de vecinos es %d" % optimal_k)

#Grafica Error vs k
missclasification_graph(neighbors,MSE)
```

Fig.6: Código de implementación de algoritmo KNN

Luego de obtener el valor del k óptimo se calcula mediante medidas de puntuaciones tales como: Accuracy, Macro Average, Micro Average, Weighted, Macro Average weighted. La implementación se presenta en la Figura 7.

```
def medirRendimiento(y_test,pred,X,y):
      print("Rendimiento Accuracy 1: ",knn.score(X,y))
      print("Rendimiento Accuracy Int: ",accuracy_score(y_test, pred, normalize=False))
      print("Rendimiento Accuracy Float: ",accuracy_score(y_test, pred))
      print("Rendimiento Macro: ",f1_score(y_test, pred, average='macro'))
      print("Rendimiento Micro: ",f1_score(y_test, pred, average='micro'))
      print("Rendimiento weighted: ",f1_score(y_test, pred, average='weighted'))
      print("Rendimiento None: ",f1_score(y_test, pred, average=None))

#-------------------------------------------------------------------------------

knn = KNeighborsClassifier(n_neighbors=optimal_k,p=2,metric='minkowski')
knn.fit(X, y)
pred = knn.predict(X_test)      #test
print("PREDICCION",pred)
medirRendimiento(y_test,pred,X,y)
```

Fig. 7: Código para implementación de medidas de rendimiento.

En la figura 8 , a su vez se verifica la predicción del knn mediante el método de sklearn, knn.predict, el cual después de generar el clasificador k-NN, predice la etiqueta de un nuevo punto de datos, pero no es un buen indicador de la capacidad del modelo para generalizar a datos nuevos y no vistos.

5. RESULTADOS

Empleando el dataset de 15000 datos (recogidos por muestreo de los 106574 datos) se obtuvieron rendimientos bastante bajos. Al manejar una gran cantidad de atributos, el número de k vecinos para clasificar nuevos datos es bajo, a tal punto de que el k óptimo para este dataset es de 1 k-vecino. Este k óptimo provocaría problemas de clasificación ya que el algoritmo tendería a dar preferencia a todas las clases de vecinos cercanos del dato a clasificar. La figura 8 muestra el vector de salida de realizar una predicción aplicando la función *KNearNeighbor* de *Sklearn*, mientras que la figura 9 presenta las

métricas de rendimiento del algoritmo de aprendizaje. En la figura 10 se verifica como a partir del primer vecino, el error en clasificación se dispara y no vuelve a bajar con los siguientes números de vecinos.

```
('PREDICCION', array(['Peter Rudenko', 'MURCIELAGO', 'minusbaby', ..., 'Kosta T',
       'Chamomile', 'EON'], dtype=object))
```

Fig 8 : Predicción por sklearn.KNearNeighbor

```
El numero optimo de vecinos es 1
('PREDICCION', array(['Peter Rudenko', 'MURCIELAGO', 'minusbaby', ..., 'Kosta T',
       'Chamomile', 'EON'], dtype=object))
('Rendimiento Accuracy 1: ', 0.99866657777185142)
('Rendimiento Accuracy Int: ', 4945)
('Rendimiento Accuracy Float: ', 0.99898989898989898)
('Rendimiento Macro: ', 0.99889910470440868)
('Rendimiento Micro: ', 0.99898989898989898)
('Rendimiento weighted: ', 0.99921677088343752)
('Rendimiento None: ', array([ 1.,  1.,  1., ...,  1.,  1.,  1.]))
```

Fig 9 : Métricas de Rendimientos del método clasificatorio kNN con el dataset 1.

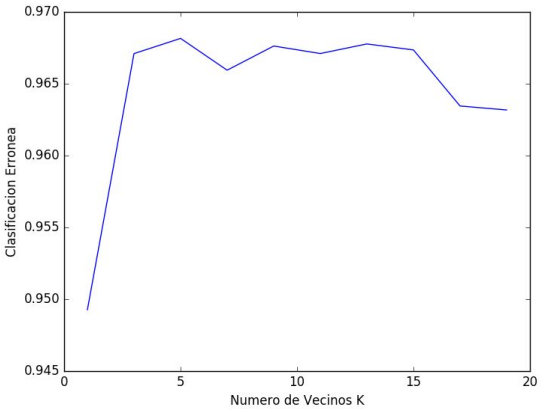

Fig 10. : Gráfico Clasificación Errónea vs k Vecinos con un dataset de 15000 datos y cross-validation de 10 subconjuntos.

Al repetir la práctica empleando un dataset de 1000 datos e igual número de atributos, se obtuvo un rendimiento mayor y la disminución de errores de clasificación, como se puede apreciar en la figura 11 el nivel de precisión(accuracy) alcanza el 36% aproximadamente.

```
('Rendimiento Accuracy 1: ', 0.377)
('Rendimiento Accuracy Int: ', 122)
('Rendimiento Accuracy Float: ', 0.36969696969696969)
('Rendimiento Macro: ', 0.15277716458482904)
('Rendimiento Micro: ', 0.36969696969696969)
('Rendimiento weighted: ', 0.29549912229451297)
('Rendimiento None: ', array([ 0.        ,  0.4       ,  0.        ,  0.        ,  0.18181818,
        0.        ,  0.        ,  0.        ,  0.        ,  0.4       ,
        0.66666667,  0.44444444,  0.54545455,  0.66666667,  0.        ,
        0.26666667,  0.        ,  0.        ,  0.        ,  0.30434783,
        0.        ,  0.52631579,  0.28571429,  0.        ,  0.        ,
        0.66666667,  0.        ,  0.        ,  0.        ,  0.        ,
        0.4       ,  0.        ,  0.        ,  0.        ,  0.36363636,
        0.        ,  0.31578947,  0.        ,  0.        ,  0.25      ,
        0.        ,  0.8       ,  0.        ,  0.57142857,  0.25      ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.8       ,  0.        ,
        0.        ,  0.        ,  0.4       ,  0.        ,  0.        ,
        0.        ,  0.53164557,  0.        ,  0.        ,  0.33333333,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.25      ,
        0.        ,  0.        ,  0.        ,  0.5       ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.5       ,  0.25      ,
        0.        ,  0.48648649,  0.85714286,  0.44444444,  0.66666667,
        0.        ,  0.28571429,  0.        ,  0.        ,  0.66666667]))
```

Figura 11: Métricas de Rendimiento para el dataset de 1000 datos

Para la práctica se varió el número de subconjuntos a utilizar en la función cross-validation. Las figuras 12 y 13 muestran la variación del error al realizar un cross-validation de 10 subconjuntos para la validación y un cross-validation de 40 subconjuntos respectivamente. En ambas pruebas se llegó determinar el valor óptimo de k como k=17.

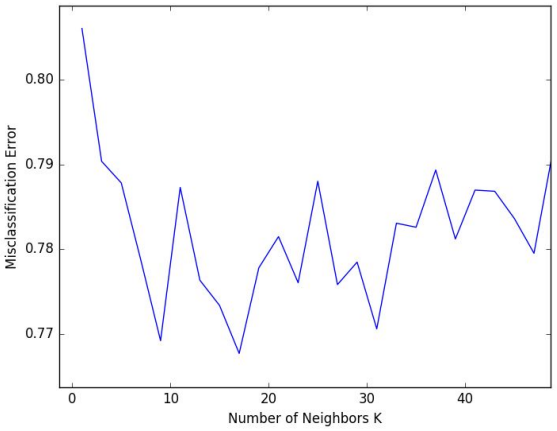

Fig 12: Gráfico Clasificación Errónea vs k Vecinos con un dataset de 1000 datos y cross validation de 10 subconjuntos.

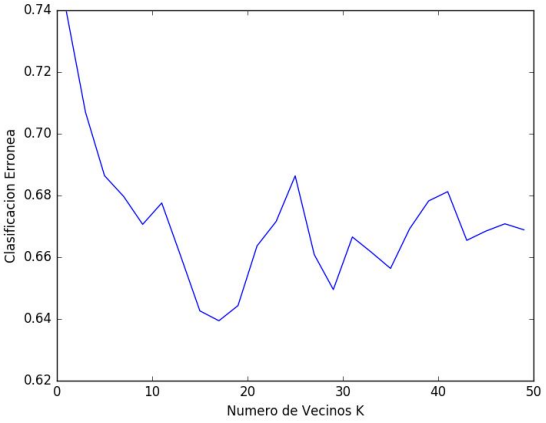

Fig 13 : Gráfico Clasificación Errónea vs k Vecinos con un dataset de 1000 datos y cross validation de 40 subconjuntos.

6. RECOMENDACIONES

El proceso de clasificación por artistas es una tarea complicada debido a que los atributos de una pista de audio no son suficientemente pertinentes para identificar claramente a un determinado artista. Por lo que se recomienda la búsqueda de otras variables que den soporte al proceso de clasificación.

7. CONCLUSIONES

- La principal ventaja de este enfoque basado en la memoria es que el clasificador se adapta de inmediato a medida que se recopilan nuevos datos de entrenamiento. Sin embargo, la

desventaja es que la complejidad computacional para clasificar nuevas muestras crece linealmente con el número de muestras en el training dataset.

- Una gran cantidad de atributos, no mejora el rendimiento ni la predicción por el método clasificatorio kNN, ya que el número de k vecinos disminuye y los errores aumentan.
- Una gran cantidad de atributos puede entorpecer la correcta clasificación, debido a lo irrelevantes que son determinados atributos ante ciertos casos.
- EL entrenamiento de un kNN clasificatorio es muy rápido, sin tener pérdida de datos.
- El método de validación cruzada (cross-validation) facilita y mejora obtener un muestreo representativo de todo el dataset utilizado.
- Por medio de la aplicación de knn para este tipo de dataset complejo se puede obtener una guía para aplicarlo en trabajos relacionados con grandes fuentes de datos.

7. REFERENCIAS

[1] Colaboradores UCI, *UCI Machine Learning Repository* Available online: https://goo.gl/PhtzjZ.

[2] Raschka Sebastian- 2016 - *Python Machine Learning*

[3] Gravano, Agustin - 2015 - *Aprendizaje Automatico* Available online: http://www.dc.uba.ar/materias/aa/2015/cuat2/ibl

[4] Colaboradores Scikit, *Scikit-Learn Documentation.* Available online: http://scikit-learn.org/stable/

[5] Microsoft Excel https://products.office.com/es/excel

[6] IBM SPSS Statistics 23 Available https://www.ibm.com/mx-es/marketplace/spss-statistics