Laboratorio 2: Nearest Neighbor

León Felipe Dueñas González
Posgrado en Ciencia e Ingeniería de la Computación
UNAM
Ciudad de México, México
leoactuario@ciencias.unam.mx

***Abstract*—Este artículo muestra el estudio e implementación del código KNN con la idención de agregar mejoras a la paqueteria y poder implementar en sobre las bases de datos proporcionadas en clase.**

***Index Terms*—Nearest neighbor methods; machine learning; KNN; Clasificación; Acurracy; clases**

I. Introduction

Dentro de todos los métodos de clasificación, los modelos "K Nearest Neighbors" (KNN) resaltan por su sencillez y efectividad. Este método se clasifica dentro de la familia de algoritmos de aprendizaje supervizado y puede usarse tanto para clasificación como para regresión. [2]

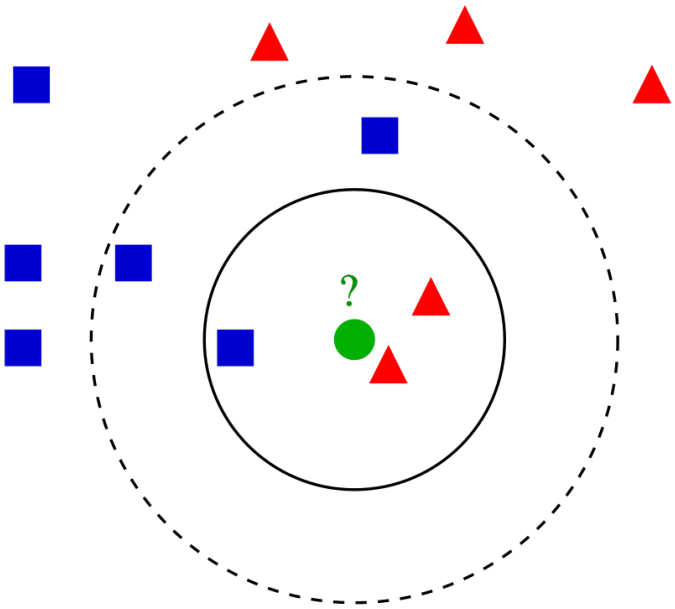

Fig. 1. Diagrama del funcionamiento del modelo KNN

II. Análisis

El *Notebook* compartido nos presenta desde el principio la clase que fue desarrollada.

```
class kNN():
    def __init__(self, k = 3, exp = 2):
        self.k = k
        self.exp = exp
```

Vemos como el método "init" nos da pauta a tener valores preestablecidos a la hora de ejecutar el código, uno de ellos es la "k", la cual representa el tamaño de la vecindad a utilizar y en esta ocasión vemos que se establece con tres parámetros a la redonda.

Adicionalmente vamos el parámetro "exp" el cual será quien nos determine el exponente de la distancia, cuando tenemos establecido el parámetro en dos, veremos la distancia euclidiana y si cambiamos ese valor por "1" estaríamos trabajando con la distancia Manhattan.

Sea p el valor dado al parámetro "exp" entonces tendremos la siguiente generalización de la métrica:

$$||x_n - y_n||_p$$

Posteriormente, el código pasa definir el método "fit" dentro de la misma clase.

```
    def fit(self, X_train, Y_train):
        self.X_train = X_train
        self.Y_train = Y_train
```

El objetivo del método es asignar en todo momento los valores que serán usados para los entrenamientos. Este método no genera cálculo alguno, simplemente asigna los datos.

```
def getDiscreteClassification(self,X_test):
    Y_pred_test = []
```

El método busca generar la predicción de clases:

- Inicializa la lista de predicciones

```
Y_pred_test = []
```

- Itera sobre todas las instancias de prueba

```
for i in range(len(X_test)):
    test_instance = X_test.iloc[i]
```

- Calcula las distancias con cada instancia de entrenamiento

```
for i in range(len(X_test)):
distances = []
for j in range(len(self.X_train)):
    train_instance = self.X_train.iloc[j]
```

```
    distance = self.Minkowski_distance(t
    distances.append(distance)
```

- Converte las distancias a un DataFrame y las ordena

```
df_dists = pd.DataFrame(data=distances,
df_nn = df_dists.sort_values(by=['dist']
df_knn = df_nn[:self.k]
```

- Determina la clase mayoritaria

```
predictions = self.Y_train[df_knn.index]
y_pred_test = predictions.index[0]
```

- Agrega la predicción al resultado final

```
Y_pred_test.append(y_pred_test)
```

- Devuelve el vector de predicciones

```
return Y_pred_test
```

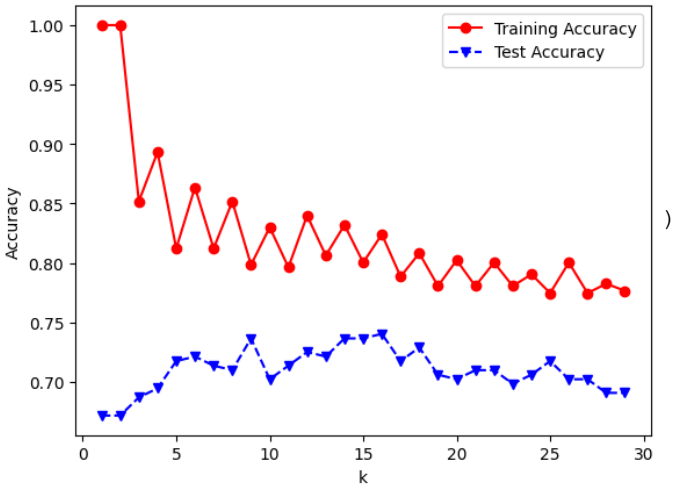

Fig. 2. Exactitud del modelo sobre la base diabetes variando k

Podemos notar que para $k = 1$ la exactitud del modelo es 1 con los datos de entrenamiento, pero es ínfimo con los datos de prueba. Tambien notamos que el mejor pronostico con los datos de entrenamiento se da con $k = 16$. Des pues de ese tamaño la exactitud comienza a disminuir.

posteriormente probamos normalizando los datos.

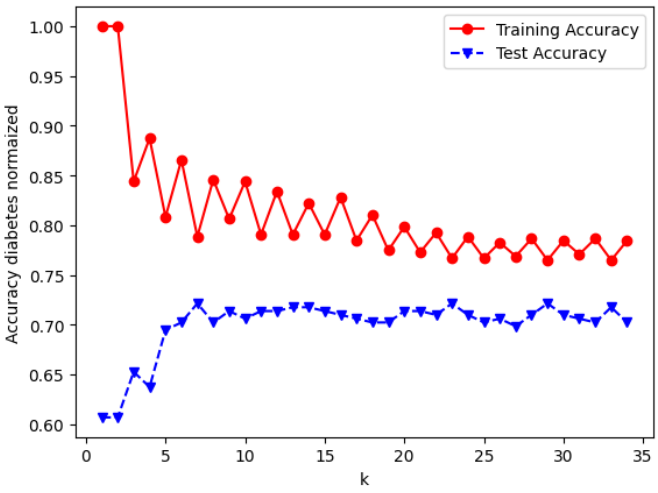

Fig. 3. Exactitud del modelo sobre la base diabetes variando k con datos normalizados.

A. *Etapas de la investigación*

1) **Extracción y consulta de la información**:
   - Se recuperó el archivo desde la tarea asignada por la Dra. Jimena Olveres Montiel en el grupo de *Classroom* "Aprendizaje de Máquina para VC 2025-2"
   - Esta información fue almacenada en un *dataframe* de *Pandas*, librería altamente utilizada para el manejo de la información.
2) **Implementación**:
   - Se agregaron tres métodos al código los cuales fueron probados en distintas etapas con las bases de datos proporcionadas por la Dra.

B. *Resultados*

Se implemento el código proporcionado en la base de datos de diabetes y se comparan las exactitudes a lo largo de distintos valores de k.

vemos como la exactitud con los datos normalizados converge de forma más rápida y no decrementa como con los datos no normalizados

Por lo que realizamos la misma prueba pero con la base de datos de glass.

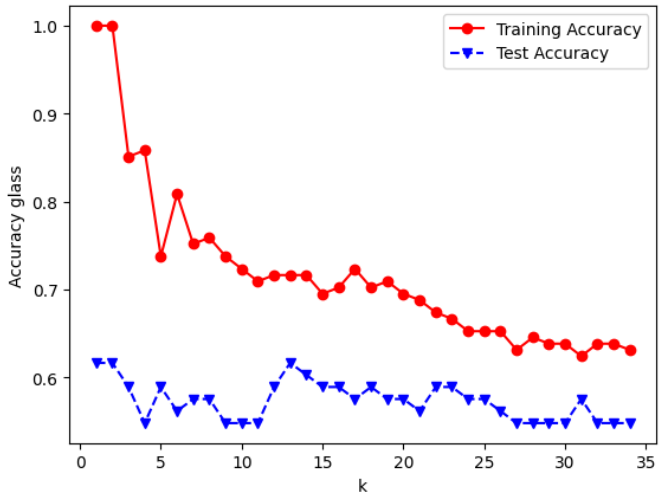

Fig. 4. Comparación de exactitud con la base glass.

Ahora normalizamos los datos.

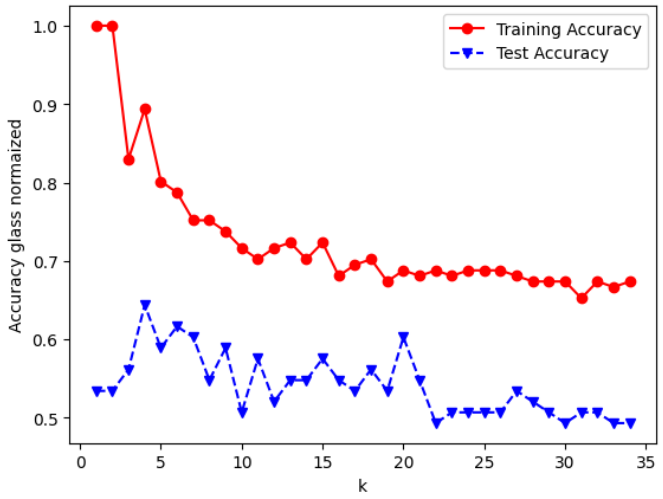

Fig. 5. Base de datos glass normalizada

De nueva cuenta encontramos que la base de datos normalizada llega de forma más rápida a la k optima dándonos mejores resultados con grupos pequeños.

Con respecto a la exactitud del modelo k-Nearest Neighbors (k-NN) en función del parámetro exp, que controla la métrica de distancia utilizada:

Línea roja (Training Accuracy): Exactitud del modelo en los datos de entrenamiento. Línea azul (Test Accuracy): Exactitud del modelo en los datos de prueba. Puntos clave observados en el gráfico:

Cuando exp=2 (distancia Euclidiana):

La exactitud en entrenamiento es alta (˜0.85). La exactitud en prueba es moderada (˜0.58). Cuando exp=10000 (aproximación a la distancia del máximo absoluto, o Chebyshev):

La exactitud en entrenamiento disminuye significativamente (˜0.70). La exactitud en prueba también disminuye (˜0.54). Tendencia general: A medida que exp aumenta, la exactitud del modelo disminuye tanto en entrenamiento como en prueba.

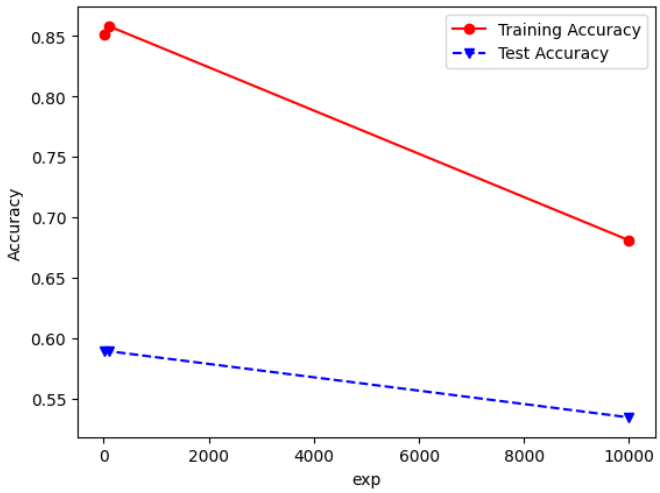

Fig. 6. Tipo de distancia utilizada en un modelo.

Procedemos a ver el error absoluto medio

```
X = autoprice.drop(columns=['class'])
Y = autoprice['class']


X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size=0.2,
random_state=10)

knn = kNN(k=3, exp=2)
knn.fit(X_train, Y_train)

predictions_df = knn.getPrediction(X_test)

mae = mean_absolute_error(Y_test, predictions_df['
print(f'Error Absoluto Medio (MAE): {mae}')

Error Absoluto Medio (MAE): 1456.0
```

El Error Absoluto Medio (MAE) de 1456.0 indica que, en promedio, la predicción del modelo k-NN se desvía 1456 unidades de los valores reales de la variable objetivo class.

```
class_probs_df = knn.getClassProbs(X_test)


print(class_probs_df.head())
print("Suma de probabilidades por fila:", class_pr
```

Este DataFrame contiene, para cada instancia en X test, la probabilidad de pertenecer a cada clase posible según el modelo k-NN.

```
      6095      5195      6695      7957      7689      7609      28248  \
0  0.333333  0.333333  0.333333       NaN       NaN       NaN       NaN
1       NaN       NaN       NaN  0.333333  0.333333  0.333333       NaN
2       NaN       NaN       NaN       NaN       NaN       NaN  0.333333
3       NaN       NaN       NaN       NaN       NaN       NaN       NaN
4       NaN       NaN       NaN       NaN       NaN       NaN       NaN

      25552     28176     7295   ...  12964  23875  21485  6989   8499   \
0       NaN       NaN       NaN  ...    NaN    NaN    NaN    NaN    NaN
1       NaN       NaN       NaN  ...    NaN    NaN    NaN    NaN    NaN
2  0.333333  0.333333       NaN  ...    NaN    NaN    NaN    NaN    NaN
3       NaN       NaN  0.333333  ...    NaN    NaN    NaN    NaN    NaN
4       NaN       NaN       NaN  ...    NaN    NaN    NaN    NaN    NaN

   10898  11248  8948   6692   6669
0    NaN    NaN    NaN    NaN    NaN
1    NaN    NaN    NaN    NaN    NaN
2    NaN    NaN    NaN    NaN    NaN
3    NaN    NaN    NaN    NaN    NaN
4    NaN    NaN    NaN    NaN    NaN

[5 rows x 60 columns]
Suma de probabilidades por fila: 0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
dtype: float64
```

Fig. 7. Salida del código.

Vemos que el número de clases no es la adecuada para el problema y tendríamos que aumentarlas para lograr una precisión adecuada.

III. Conclusiones

Se encontró que el modelo tiene un alto desempeño cuando k se ajusta adecuadamente, pero su precisión se ve afectada cuando k es muy pequeño, lo que causa sobreajuste, o cuando es muy grande, lo que introduce sesgo. En particular, al evaluar la base de datos de diabetes, se observó que la exactitud óptima se obtenía con k=16. También se evidenció que la normalización de los datos mejora significativamente el desempeño del modelo, ya que permite una convergencia más rápida y evita la degradación de la precisión cuando k aumenta.

Posteriormente, al aplicar k-NN en la base de datos auto-price, el cálculo del Error Absoluto Medio (MAE) resultó en 1456.0, indicando que el modelo tenía dificultades para hacer predicciones precisas en este contexto. Esto sugiere que k-NN podría no ser la mejor opción para este problema o que es necesario mejorar el preprocesamiento de los datos, ajustar k o explorar otros modelos.

Finalmente, al analizar la distribución de probabilidades de clasificación generadas por el modelo, se identificó que la asignación de clases no era adecuada, lo que indica la necesidad de aumentar el número de clases o ajustar la estructura del modelo para mejorar la precisión.

References

[1] H. A. Nurqamaradillah, L. Novamizanti and D. R. Wijaya, "Comparing KNN and SVM for Aroma-Based Tuna Freshness Detection," 2024 IEEE 2nd International Conference on Electrical Engineering, Computer and Information Technology (ICEECIT), Jember, Indonesia, 2024, pp. 123-128

[2] Z. Cao, Y. Shen and M. Tian, "Optimization of Weighted KNN Algorithm Based on Transition Probability," 2024 20th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Guangzhou, China, 2024, pp. 1-6, doi: 10.1109/ICNC-FSKD64080.2024.10702238.