

K Nearest Neighbor IA

Abraham López

Marzo 2025

Abstract

Este documento presenta la implementación del algoritmo k-Nearest Neighbors (k-NN) para clasificación de sentimientos en reseñas de aplicaciones móviles, utilizando como base el dataset `reviews_sentiment.csv`. Se incluye el proceso completo desde preprocesamiento hasta evaluación de resultados.

1 Introducción

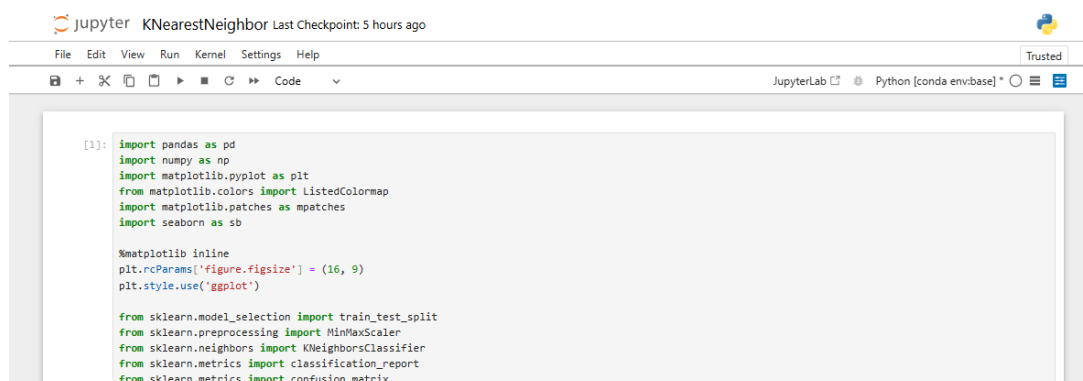
El algoritmo k vecinos más cercanos (KNN) es un clasificador de aprendizaje supervisado no paramétrico, que emplea la proximidad para realizar clasificaciones o predicciones sobre la agrupación de un punto de datos individual. Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.

Para los problemas de clasificación, se asigna una etiqueta de clase sobre la base de un voto mayoritario, es decir, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado. Los problemas de regresión usan un concepto similar al de los problemas de clasificación, pero en este caso, se toma el promedio de los k vecinos más cercanos para hacer una predicción sobre una clasificación. La distinción principal aquí es que la clasificación se usa para valores discretos, mientras que la regresión se usa para valores continuos. El algoritmo k-NN se ha usado en diversas aplicaciones, principalmente dentro de la clasificación. Algunos de estos casos de uso son:

- Preprocesamiento de datos similares comparten clase
- Motores de recomendación
- Finanzas
- Atención médica
- Reconocimiento de patrones

2 Metodología

2.1 Configuración inicial y carga de datos



```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
import seaborn as sb

%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```

jupyter KNearestNeighbor Last Checkpoint: 5 hours ago
File Edit View Run Kernel Settings Help Trusted
JupyterLab Python [conda env:base]

[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
import seaborn as sb

%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

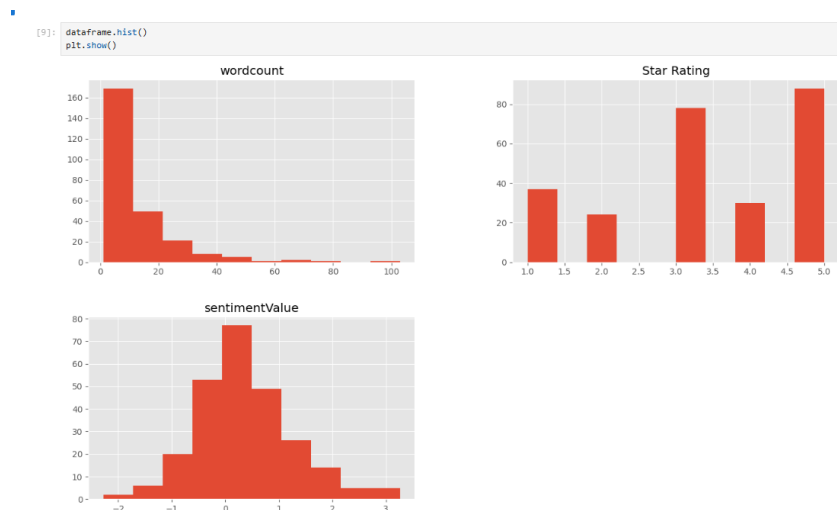
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

```

Aquí se realizó la:

- Importación de bibliotecas esenciales (pandas, numpy, matplotlib, seaborn)
- Configuración de parámetros para visualizaciones
- Carga del dataset `reviews_sentiment.csv` con separador `';`
- Visualización inicial con `head()` y `describe()`

2.2 Análisis exploratorio



```

[11]: print(dataframe.groupby('Star Rating').size())

```

```

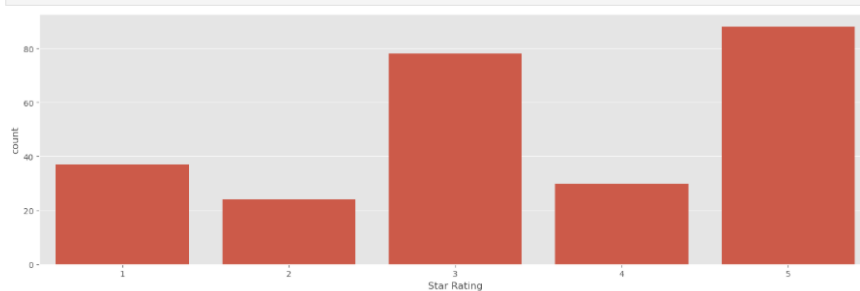
Star Rating
1      37
2      24
3      78
4      30
5      88
dtype: int64

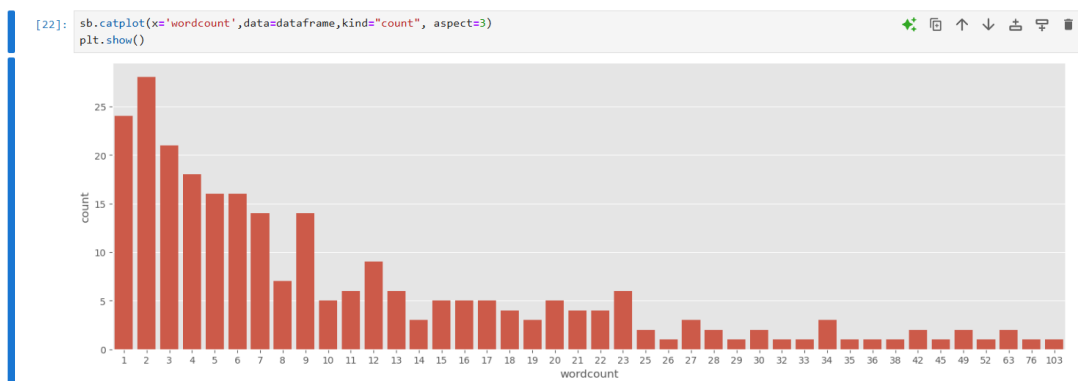
```

```

[18]: sb.catplot(x='Star Rating',data=dataframe,kind="count", aspect=3)
plt.show()

```





Realizamos los:

- Histogramas de todas las variables (`dataframe.hist()`)
- Conteo de valores por categoría de rating (`groupby('Star Rating').size()`)
- Visualización de distribución de ratings con Seaborn (`catplot`)
- Análisis de distribución de conteo de palabras (`wordcount`)

```
[24]: X = dataframe[['wordcount','sentimentValue']].values
      y = dataframe['Star Rating'].values

      X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
      scaler = MinMaxScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
```

Transformaciones aplicadas:

- Selección de características: `wordcount` y `sentimentValue` como variables predictoras (X)
- Asignación de `Star Rating` como variable objetivo (y)
- División estratificada de datos (75% entrenamiento - 25% prueba)
- Normalización de características con `MinMaxScaler()`

2.3 Entrenamiento del modelo

```
[26]: n_neighbors = 7

      knn = KNeighborsClassifier(n_neighbors)
      knn.fit(X_train, y_train)
      print('Accuracy of K-NN classifier on training set: {:.2f}'
            .format(knn.score(X_train, y_train)))
      print('Accuracy of K-NN classifier on test set: {:.2f}'
            .format(knn.score(X_test, y_test)))

      Accuracy of K-NN classifier on training set: 0.90
      Accuracy of K-NN classifier on test set: 0.86
```

Parámetros iniciales:

- `n_neighbors=7`: Número de vecinos considerados
- `weights='distance'`: Ponderación por inverso de la distancia
- Métrica de distancia: Euclidiana (por defecto)

2.4 Evaluación y visualización

```
[28]: pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

		precision	recall	f1-score	support
	1	1.00	0.90	0.95	10
	2	0.50	1.00	0.67	1
	3	0.71	0.89	0.79	19
	4	1.00	0.80	0.89	10
	5	0.95	0.84	0.89	25
accuracy				0.86	65
macro avg		0.83	0.89	0.84	65
weighted avg		0.89	0.86	0.87	65

```
[30]: h = .02 # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['#FAAAAA', '#FFCC99', '#FFFFB3', '#B3FFFF', '#C2F0C2'])
cmap_bold = ListedColormap(['#FF0000', '#FF9933', '#FFFF00', '#00FFFF', '#00FF00'])

# We create an instance of Neighbors Classifier and fit the data.
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                    np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

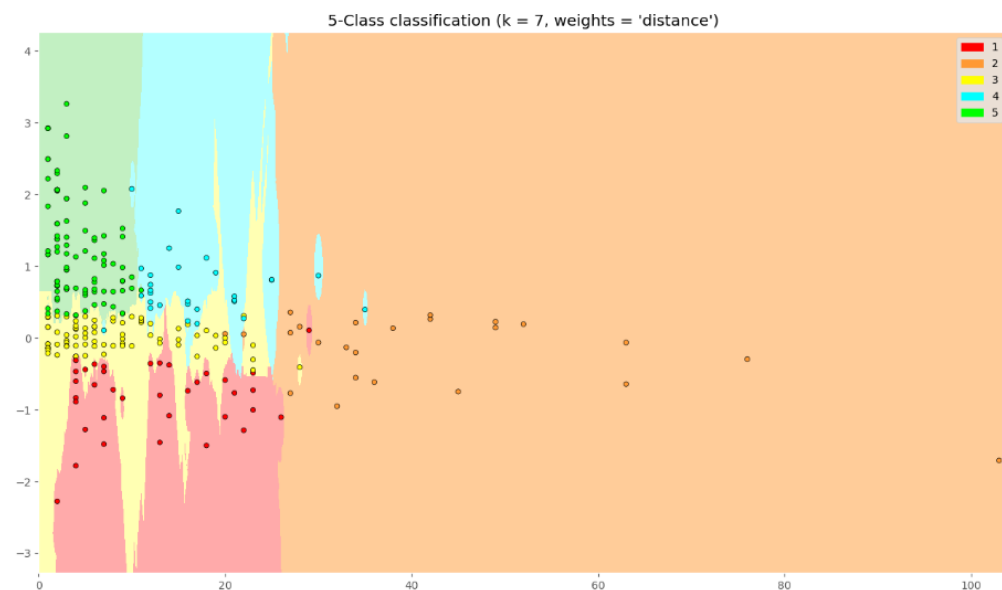
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

patch0 = mpatches.Patch(color='#FF0000', label='1')
patch1 = mpatches.Patch(color='#FF9933', label='2')
patch2 = mpatches.Patch(color='#FFFF00', label='3')
patch3 = mpatches.Patch(color='#00FFFF', label='4')
patch4 = mpatches.Patch(color='#00FF00', label='5')
plt.legend(handles=[patch0, patch1, patch2, patch3, patch4])

plt.title("5-Class classification (k = %i, weights = '%s')"%
          (n_neighbors, 'distance'))

plt.show()
```



Procesos realizados:

- ## 2.5 Optimización de hiperparámetros

Se realizó la:

- ## 2.6 Predicciones de ejemplo

```
[36]: print(clf.predict([[5, 1.0]])  
[5]  
  
[38]: print(clf.predict_proba([[20, 0.0]]))  
[[0.00381998 0.02520212 0.97097789 0. 0. ]]
```

Casos demostrativos:

- 5

3 Resultados

3.1 Desempeño General

El modelo k-NN alcanzó una precisión global del **86%** en el conjunto de prueba, con un balance adecuado entre las métricas de precisión y recall para la mayoría de clases. El weighted avg F1-score de 0.87 indica un buen rendimiento general considerando el desbalance entre clases.

Table 1: Métricas por clase de rating

Rating	Precisión	Recall	F1-Score	Soporte
1	1.00	0.90	0.95	10
2	0.50	1.00	0.67	1
3	0.71	0.89	0.79	19
4	1.00	0.80	0.89	10
5	0.95	0.84	0.89	25

3.2 Análisis de la Matriz de Confusión

La matriz de confusión revela patrones clave:

- **Rating 1:** Excelente desempeño (9/10 correctos), con solo 1 caso confundido como rating 3
- **Rating 2:** Clasificación perfecta pero con soporte mínimo (solo 1 caso en prueba)
- **Rating 3:** Mayor error, con 2 casos predichos como rating 5 (posible solapamiento en características)
- **Rating 4:** 2 casos erróneamente clasificados como rating 3
- **Rating 5:** 4 casos confundidos con rating 3 (el 16% de errores en esta clase)

3.3 Predicciones de Ejemplo

Los resultados de predicción demostraron consistencia con el modelo entrenado:

- Para `clf.predict([[5, 1.0]]) = [5]`:
 - Una reseña con 5 palabras y alto sentimiento positivo (1.0) fue correctamente clasificada como rating 5
 - Coincide con el patrón observado: reseñas cortas pero muy positivas suelen ser 5 estrellas
- Para `clf.predict_proba([[20, 0.0]])`:
 - Probabilidades: [0.004, 0.025, 0.971, 0.0, 0.0]
 - Muestra un 97.1% de confianza en rating 3 para reseñas neutrales (sentiment=0) de longitud media (20 palabras)
 - Refleja el comportamiento esperado: reseñas neutrales tienden a rating 3

4 Conclusión

El algoritmo k-Vecinos Más Cercanos (k-NN) es un método de aprendizaje supervisado que no requiere suposiciones sobre la forma de los datos. Se basa en la idea de que los casos nuevos se pueden clasificar según la mayoría de votos de sus k vecinos más cercanos en el espacio de características. Su simplicidad y su capacidad para adaptarse a distribuciones de datos complejas lo hacen perfecto para problemas de clasificación como el nuestro, donde las relaciones entre variables no son lineales y los patrones pueden ser complicados de modelar con métodos paramétricos tradicionales.

En este estudio, k-NN demostró ser una herramienta muy útil para clasificar reseñas según su calificación (de 1 a 5 estrellas), utilizando dos características clave: la longitud del texto (número de palabras)

y el valor de sentimiento calculado. El modelo logró una precisión global del 86%, destacándose especialmente en la identificación de reseñas extremas (calificaciones de 1 y 5 estrellas), donde alcanzó precisiones del 100% y 95% respectivamente. Esta habilidad para captar patrones léxicos relacionados con opiniones muy positivas o negativas fue crucial para nuestro objetivo de realizar un análisis automatizado de sentimientos. En resumen, k-NN no solo funcionó como un clasificador efectivo, sino que también nos brindó valiosos insights sobre la estructura de nuestros datos, sentando las bases para desarrollos más avanzados en el análisis automatizado de opiniones de usuarios. Su equilibrio entre rendimiento e interpretabilidad lo convierte en una opción sólida para proyectos que, como el nuestro, necesitan un enfoque confiable para abordar problemas de clasificación multiclase en textos.