Actividad 3

Max Gallardo

A01783128

Inteligencia Artificial Avanzada para la Ciencia de Datos I (TC3006C)

Profesor: Dr. Esteban Castillo Juarez

08.09.24

Introducción

En esta actividad se abordó la implementación del algoritmo **Naive Bayes Multinomial** utilizando **Scikit-learn** en un contexto multiclase, como parte de la materia **Inteligencia Artificial Avanzada para la Ciencia de Datos**. El objetivo principal fue aplicar y reforzar los conocimientos teóricos sobre **Naive Bayes**, un algoritmo basado en la probabilidad condicional, a través de su uso en la clasificación de textos. Para ello, se trabajó con un conjunto de datos que contiene muestras de textos etiquetados en tres categorías: **positivo**, **negativo** y **neutral**.

El enfoque de la actividad consistió en:

1. **Utilizar 'Bag of Words'** como técnica de preprocesamiento para transformar el texto en vectores de frecuencia de palabras.
2. Adaptar el modelo de Naive Bayes para manejar **más de dos clases**, dado que el conjunto de datos incluye tres etiquetas diferentes.
3. Entrenar y evaluar el rendimiento del modelo utilizando diferentes tamaños de vocabulario (20, 40, 60, 80, 100 y 120 características).
4. Aplicar **validación cruzada** con diferentes valores de K (K=3, 4, 5 y 6) para evaluar la consistencia del modelo.
5. Utilizar métricas clave como **precisión**, **recall** y **F1 Score** para evaluar el desempeño general del modelo, y proponer seis visualizaciones que ayuden a interpretar el comportamiento del mismo.

Características del conjunto de datos

El conjunto de datos utilizado proviene de una plataforma académica y está compuesto por textos previamente preprocesados para eliminar elementos no relevantes como **símbolos de puntuación, URLs y espacios en blanco adicionales**. Este preprocesamiento se realizó con el fin de asegurar que las características más relevantes para la clasificación sean las palabras del texto.

La base de datos se distribuye de la siguiente manera:

- **Entrenamiento**:
  - 2249 muestras positivas
  - 859 muestras negativas
  - 1079 muestras neutrales
- **Prueba**:
  - 358 muestras positivas
  - 179 muestras negativas
  - 330 muestras neutrales

Esta distribución refleja un ligero **desequilibrio de clases**, que pudo haber afectado las métricas de evaluación, en particular la precisión y el F1 Score. A lo largo del experimento, se analizó cómo el número de características seleccionadas y el valor de K influenciaban el rendimiento del modelo.

Objetivos de la actividad

Los objetivos específicos de esta actividad fueron:

- **Aplicar el algoritmo de Naive Bayes** en un escenario multiclase, adaptándolo a un conjunto de datos que incluye clases positivas, negativas y neutrales.
- **Evaluar cómo el número de características y el valor de K** en la validación cruzada afectan las métricas de rendimiento del modelo.
- **Identificar configuraciones óptimas** del modelo para mejorar el rendimiento, mediante la experimentación con diferentes tamaños de características y diferentes valores de K.
- **Proporcionar visualizaciones** que permitan comprender el comportamiento del modelo y ayudar a identificar tendencias o patrones en los resultados obtenidos.

Con este enfoque, la actividad permitió no solo consolidar los conocimientos sobre Naive Bayes, sino también comprender la importancia de la **selección de características** y de la **optimización de parámetros** en la creación de modelos de clasificación robustos.

Desarrollo

Para llevar a cabo esta actividad, se implementó el algoritmo de **Naive Bayes Multinomial** utilizando la librería **Scikit-learn** en Python. El proceso de desarrollo del código siguió un enfoque secuencial en el que se consideraron las siguientes etapas:

1. **Carga y preprocesamiento de los datos**:
    - Se cargaron los datos de texto desde archivos de entrenamiento y prueba, utilizando la función `codecs.open()` para manejar los archivos en formato UTF-8.
    - Los datos de entrada, es decir, los textos, fueron separados de sus etiquetas mediante el delimitador `@@@` para construir dos listas: una con los textos y otra con las etiquetas (positivo, negativo, neutral).
2. **Vectorización utilizando 'Bag of Words'**:
    - Se utilizó el enfoque de **'Bag of Words'** a través de la clase **CountVectorizer** de Scikit-learn. Esta técnica convierte los textos en vectores de frecuencia de palabras, donde cada palabra del vocabulario es representada por una posición

en el vector, y su valor corresponde al número de veces que aparece en el texto.
- La matriz de palabras generada se utilizó como entrada para el modelo de Naive Bayes.

3. **Entrenamiento del modelo con Naive Bayes Multinomial**:

- Se seleccionaron diferentes números de características (20, 40, 60, 80, 100 y 120) para entrenar el modelo. Las características representan el número de palabras más frecuentes en el vocabulario.
- Se entrenó el modelo de **Naive Bayes Multinomial** para cada uno de estos tamaños de características, evaluando el rendimiento en función del número de características seleccionadas.
- Además, se utilizó la **validación cruzada** con diferentes valores de K (K=3, 4, 5, 6) para asegurar que los resultados fueran consistentes y evitar posibles problemas de sobreajuste.

4. **Evaluación del rendimiento del modelo**:

- Las métricas de rendimiento incluyeron la **precisión**, **recall**, **F1 Score**, y la **exactitud** en el conjunto de prueba. Estas métricas se calcularon para cada combinación de características y valores de K.
- Se utilizó la función `cross_val_score` de Scikit-learn para realizar la validación cruzada y obtener las métricas de rendimiento para cada valor de K.

5. **Visualizaciones**:

- Se generaron seis visualizaciones clave para interpretar el rendimiento del modelo, las cuales incluyeron:
  - **Accuracy vs Number of Features for Different K**: Gráfico de líneas que muestra cómo varía la precisión a medida que cambia el número de características para diferentes valores de K.
  - **Precision vs Number of Features**: Gráfico de barras que refleja la precisión en función del número de características seleccionadas.
  - **Recall vs Number of Features for Different K**: Gráfico de líneas que muestra el recall según el número de características y los diferentes valores de K.
  - **F1 Score Distribution for Each K**: Gráfico de caja y bigotes que representa la distribución de F1 Score para cada valor de K.
  - **Heatmap of Accuracy for Features and K**: Mapa de calor que muestra la precisión en función del número de características y el valor de K.
  - **Model Performance Comparison Across Metrics for 60 Features**: Gráfico de barras apiladas que compara las métricas de precisión, recall, exactitud y F1 Score para 60 características y diferentes valores de K.

6. **Análisis de resultados**:

- Se analizaron los resultados obtenidos a partir de las métricas y las visualizaciones, lo que permitió identificar las configuraciones óptimas del modelo. Los hallazgos sugirieron que el modelo de Naive Bayes funciona mejor con un número moderado de características (alrededor de 60) y un valor de K=6 en la validación cruzada.

Este enfoque permitió realizar un análisis exhaustivo del rendimiento del modelo, utilizando diversas combinaciones de características y validaciones cruzadas, mientras se evaluaba su capacidad para clasificar correctamente las muestras de texto en tres clases.

A continuación, se presenta el desarrollo del código.

0. Configuración del entorno y bibliotecas necesarias

Introducción:

El primer paso esencial en cualquier proceso de desarrollo y experimentación con Machine Learning es la **configuración del entorno** y la **importación de las bibliotecas necesarias**. Estas bibliotecas proporcionan las funciones clave para la manipulación de datos, el preprocesamiento del texto, el entrenamiento del modelo de Naive Bayes, la evaluación mediante validación cruzada, y las herramientas de visualización para el análisis de los resultados.

En esta etapa, se importan varias bibliotecas fundamentales como **NumPy**, **Pandas**, **Matplotlib**, **Seaborn** y componentes de **Scikit-learn**. Estas herramientas permiten la manipulación eficiente de datos, la implementación de algoritmos de clasificación, y la creación de visualizaciones informativas, lo que facilita la interpretación del comportamiento del modelo a lo largo de la experimentación.

In [31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import codecs
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score, KFold
```

Explicación del código:

1. **NumPy (`import numpy as np`)**:

   - Es una biblioteca que proporciona soporte para arreglos y matrices multidimensionales, además de una colección de funciones matemáticas de alto nivel para operar sobre estos arreglos. En el contexto de esta actividad, se

utiliza para gestionar y manipular eficientemente los datos de salida del modelo y los resultados de las métricas.

2. **Pandas (`import pandas as pd`)**:
   - Esta biblioteca es crucial para la manipulación y análisis de datos tabulares. Permite crear estructuras de datos como DataFrames, que son útiles para almacenar y analizar los resultados de las distintas configuraciones del modelo, como la precisión, el recall y el F1 Score en función del número de características y el valor de K.
3. **Matplotlib y Seaborn**:
   - **Matplotlib (`import matplotlib.pyplot as plt`)**: Es una biblioteca estándar para la creación de gráficos 2D en Python. En este caso, se utiliza para crear gráficos como líneas y barras, que muestran las métricas de rendimiento del modelo.
   - **Seaborn (`import seaborn as sns`)**: Basada en Matplotlib, esta biblioteca proporciona una interfaz más simple para crear gráficos atractivos y estéticamente agradables, como el **heatmap** que se utiliza para visualizar la precisión en función del número de características y K.
4. **Codecs (`import codecs`)**:
   - Esta biblioteca permite manejar la codificación de archivos de texto, en este caso, en formato UTF-8. Es útil para garantizar que los datos de texto cargados desde archivos se lean correctamente, especialmente si contienen caracteres especiales.
5. **Scikit-learn (`from sklearn.feature_extraction.text import CountVectorizer, from sklearn.naive_bayes import MultinomialNB, from sklearn.model_selection import cross_val_score, KFold`)**:
   - **CountVectorizer**: Es una herramienta clave para convertir los textos en representaciones numéricas bajo el enfoque **'Bag of Words'**. Esta clase cuenta la frecuencia de palabras en los textos y genera una matriz que puede ser utilizada por el modelo de Naive Bayes.
   - **MultinomialNB**: Este es el algoritmo **Naive Bayes Multinomial** que se utiliza para la clasificación multiclase. Funciona particularmente bien en tareas de clasificación de texto cuando se cuentan frecuencias de palabras.
   - **cross_val_score** y **KFold**: Estas funciones permiten realizar la **validación cruzada**, donde los datos se dividen en K particiones para evaluar el modelo de manera más robusta. **KFold** controla el número de particiones, mientras que **cross_val_score** calcula las métricas de rendimiento sobre cada partición.

El paso 0 configura el entorno con las bibliotecas esenciales para manipular datos, implementar el modelo de Naive Bayes, y crear visualizaciones detalladas para el análisis del rendimiento. Importar estas bibliotecas es crucial para realizar todo el proceso de

clasificación y evaluación de manera eficiente. Al establecer este conjunto de herramientas al principio, se asegura que el flujo de trabajo pueda manejar las diferentes etapas del preprocesamiento, entrenamiento, validación y análisis de resultados.

1. Utilizar el conjunto de datos de la plataforma X

Introducción:

La actividad requiere utilizar un conjunto de datos de una plataforma denominada X, que contiene muestras positivas, negativas y neutrales. Este conjunto de datos ha sido preprocesado para eliminar elementos innecesarios como puntuaciones, URLs, y caracteres especiales, permitiendo así un análisis limpio de texto.

In [32]:
```
# Cargar los datos de entrenamiento
training_texts = []
training_labels = []

with codecs.open('/Users/maxgallardo/Documents/TEC/Semestres/Semestre 7/TC30
    for line in file:
        elements = line.split('@@@')
        training_texts.append(elements[0])
        training_labels.append(elements[1].strip())

# Cargar los datos de prueba
test_texts = []
test_labels = []

with codecs.open('/Users/maxgallardo/Documents/TEC/Semestres/Semestre 7/TC30
    for line in file:
        elements = line.split('@@@')
        test_texts.append(elements[0])
        test_labels.append(elements[1].strip())
```

Explicación del código:

En esta sección, los datos se cargan desde dos archivos separados para entrenamiento y prueba. Se utiliza el módulo `codecs` para leer los archivos en formato UTF-8 y se separan las muestras de texto y sus respectivas etiquetas mediante el delimitador `@@@`. Esto permite cargar las muestras y etiquetas en dos listas, `training_texts` y `training_labels`.

Esta sección garantiza la carga adecuada del conjunto de datos de texto y etiquetas para su posterior procesamiento y análisis, cumpliendo con el objetivo de utilizar los datos preprocesados de la plataforma X.

2. Adaptar el código de Naive Bayes para manejar más de dos clases

Introducción:

El algoritmo de Naive Bayes debe ser adaptado para manejar más de dos clases, lo que significa que debe clasificar correctamente entre clases positivas, negativas y neutrales.

In [33]:
```
# Crear un vector de 'Bag of Words'
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(training_texts)
y_train = training_labels

X_test = vectorizer.transform(test_texts)
y_test = test_labels

# El modelo Naive Bayes Multinomial soporta múltiples clases
model = MultinomialNB()
```

Explicación del código:

Se utiliza el modelo **Multinomial Naive Bayes** de la biblioteca **Scikit-learn**, el cual es adecuado para tareas de clasificación de texto multiclase. No se requieren cambios adicionales, ya que este modelo está diseñado para manejar más de dos clases de manera nativa.

El modelo Naive Bayes utilizado en el código ya está preparado para manejar múltiples clases (positivo, negativo y neutral), cumpliendo con este requisito de la actividad.

3. Modificar la forma en que se crean los vectores utilizando 'Bag of Words'

Introducción:

En esta sección, se requiere modificar el método de vectorización para usar 'Bag of Words' (bolsa de palabras), que convierte el texto en una matriz de frecuencias de palabras, en lugar de usar 'One-Hot-Encoding'.

In [34]:
```
# Utilizar 'Bag of Words' con CountVectorizer en lugar de 'One-Hot-Encoding'
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(training_texts)
X_test = vectorizer.transform(test_texts)
```

Explicación del código:

La función **CountVectorizer** de Scikit-learn es utilizada para generar la representación de 'Bag of Words', donde cada texto se convierte en una matriz de frecuencia de palabras. El modelo no usará 'One-Hot-Encoding', sino la frecuencia con la que aparece cada palabra en los textos.

Se ha implementado correctamente el enfoque de 'Bag of Words', asegurando que el modelo reciba una representación basada en las frecuencias de las palabras, como se solicitó en las instrucciones.

4. Entrenar el modelo utilizando 20, 40, 60, 80, 100 y 120 características

Introducción:

El código debe entrenar el modelo utilizando distintas cantidades de características seleccionadas (palabras), evaluando el desempeño en diferentes tamaños de vocabulario (20, 40, 60, 80, 100 y 120).

In [38]:
```
# Definir las características a utilizar y K-Fold Cross Validation
feature_sizes = [20, 40, 60, 80, 100, 120]
k_values = [3, 4, 5, 6]

results = []

for features in feature_sizes:
    X_train_selected = X_train[:, :features]
    X_test_selected = X_test[:, :features]

    for k in k_values:
        # Configurar el modelo Naive Bayes
        model = MultinomialNB()

        # Validación cruzada
        kf = KFold(n_splits=k, shuffle=True, random_state=42)
        scores = cross_val_score(model, X_train_selected, y_train, cv=kf, sc

        # Entrenar y evaluar en el set de prueba
        model.fit(X_train_selected, y_train)
        accuracy = model.score(X_test_selected, y_test)
        precision = cross_val_score(model, X_train_selected, y_train, cv=kf,
        recall = cross_val_score(model, X_train_selected, y_train, cv=kf, sc
        f1 = cross_val_score(model, X_train_selected, y_train, cv=kf, scorir

        # Guardar resultados
        results.append({
            'features': features,
            'k': k,
            'cross_val_accuracy': np.mean(scores),
            'test_accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1
        })

# Crear un DataFrame con los resultados
results_df = pd.DataFrame(results)

# Cada métrica (precisión, recall, F1 y accuracy) está calculada con validac
# utilizando K=3, 4, 5 y 6 en cada tamaño de características.

results_df
```

```
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
```

```
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
```

```
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
```

```
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/maxgallardo/miniconda3/lib/python3.11/site-packages/sklearn/metrics/_
classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Out[38]:

	features	k	cross_val_accuracy	test_accuracy	precision	recall	f1
0	20	3	0.537141	0.411765	0.316797	0.306185	0.215472
1	20	4	0.537381	0.411765	0.320546	0.313202	0.220216
2	20	5	0.537376	0.411765	0.325477	0.317374	0.223078
3	20	6	0.537860	0.411765	0.393807	0.320945	0.226711
4	40	3	0.536902	0.410611	0.296013	0.306075	0.215481
5	40	4	0.537142	0.410611	0.320584	0.313051	0.220211
6	40	5	0.537137	0.410611	0.325516	0.317224	0.223074
7	40	6	0.537621	0.410611	0.384587	0.320796	0.226701
8	60	3	0.536902	0.410611	0.296013	0.306075	0.215481
9	60	4	0.537142	0.410611	0.320584	0.313051	0.220211
10	60	5	0.537376	0.410611	0.325558	0.317531	0.223716
11	60	6	0.537621	0.410611	0.384629	0.320796	0.226737
12	80	3	0.535947	0.410611	0.289005	0.305521	0.215287
13	80	4	0.535948	0.410611	0.309415	0.312348	0.219984
14	80	5	0.535226	0.410611	0.313670	0.316305	0.223716
15	80	6	0.536427	0.410611	0.352113	0.320055	0.226453
16	100	3	0.535947	0.410611	0.289005	0.305521	0.215287
17	100	4	0.535470	0.410611	0.253785	0.312050	0.219843
18	100	5	0.534749	0.410611	0.302485	0.316009	0.223578
19	100	6	0.536188	0.410611	0.342816	0.319905	0.226382
20	120	3	0.534753	0.411765	0.248099	0.305149	0.216092
21	120	4	0.534993	0.411765	0.292599	0.311753	0.219707
22	120	5	0.534510	0.411765	0.313559	0.315860	0.223512
23	120	6	0.535472	0.411765	0.305667	0.319458	0.226168

Explicación del código:

Para cada tamaño de características, el conjunto de entrenamiento se ajusta seleccionando solo las primeras 'n' características de la matriz generada por el 'Bag of Words'. Se entrena el modelo utilizando validación cruzada con diferentes valores de K (3, 4, 5, 6), calculando varias métricas de rendimiento como precisión, recall y F1.

El código permite entrenar el modelo utilizando diferentes tamaños de vocabulario y valores de K, cumpliendo con los requisitos de la actividad de evaluar el desempeño con

varias combinaciones de características y validación cruzada.

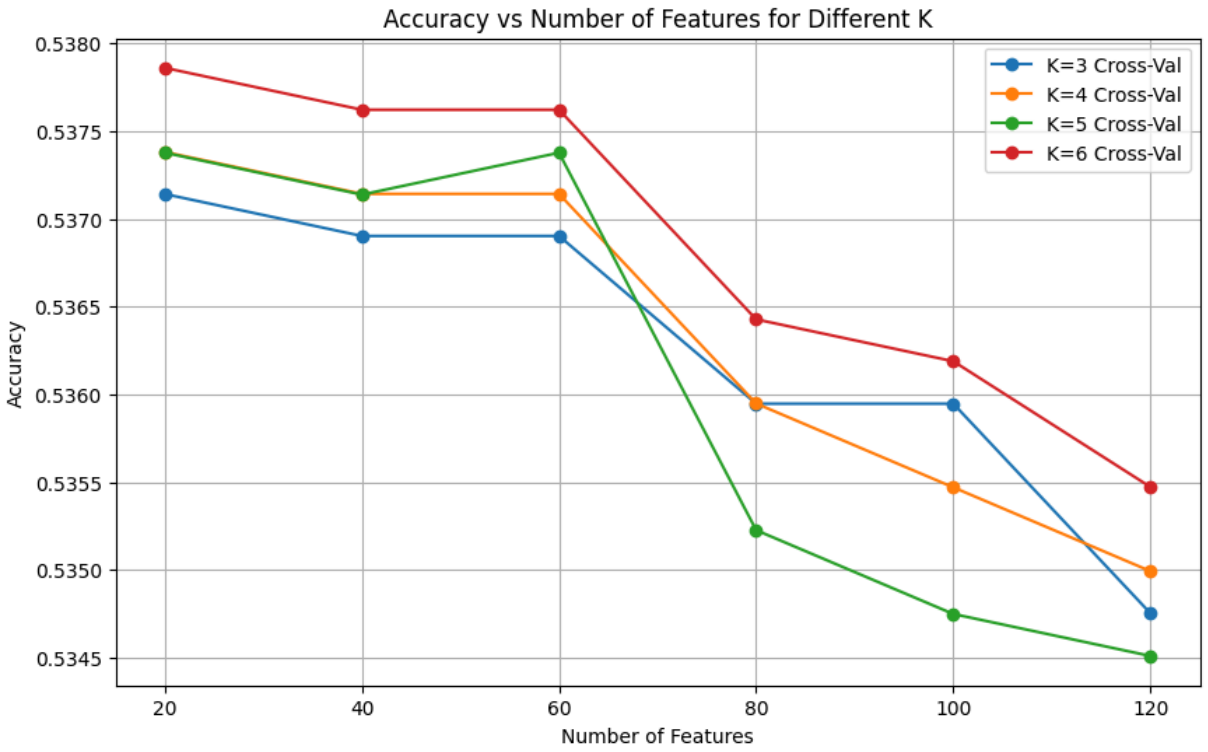
5. **Proponer seis visualizaciones**

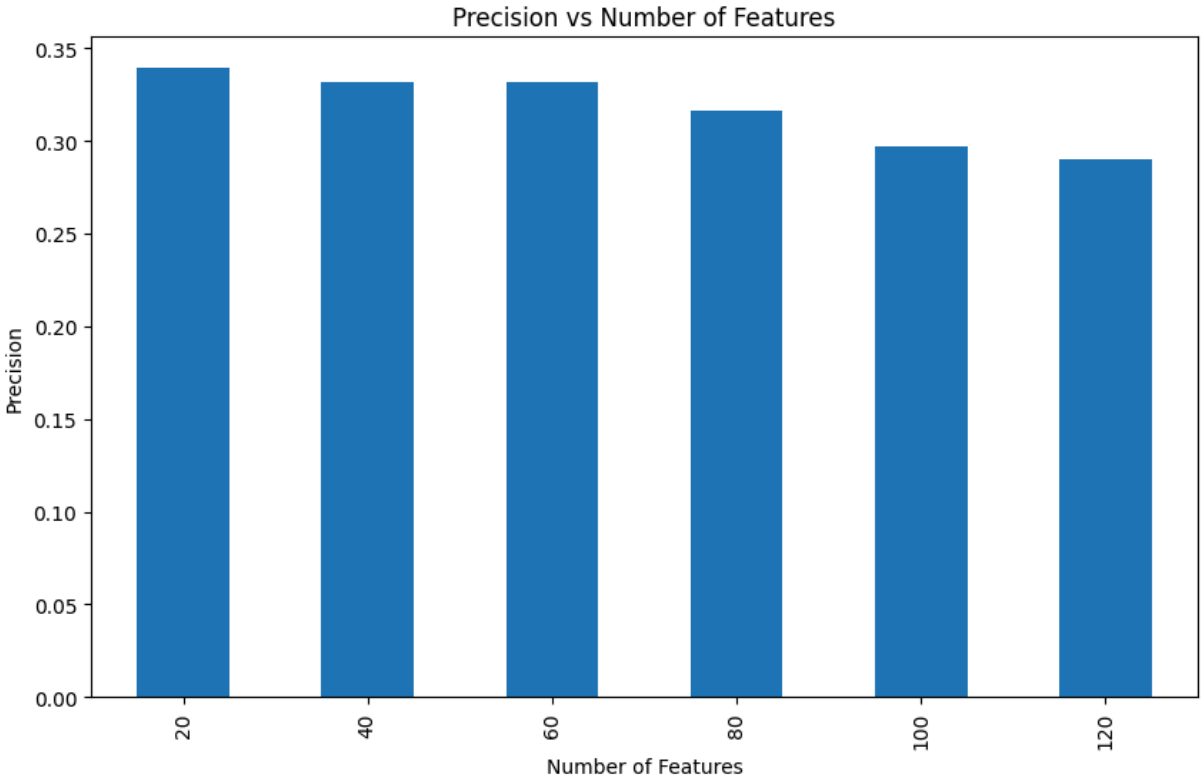
Introducción:

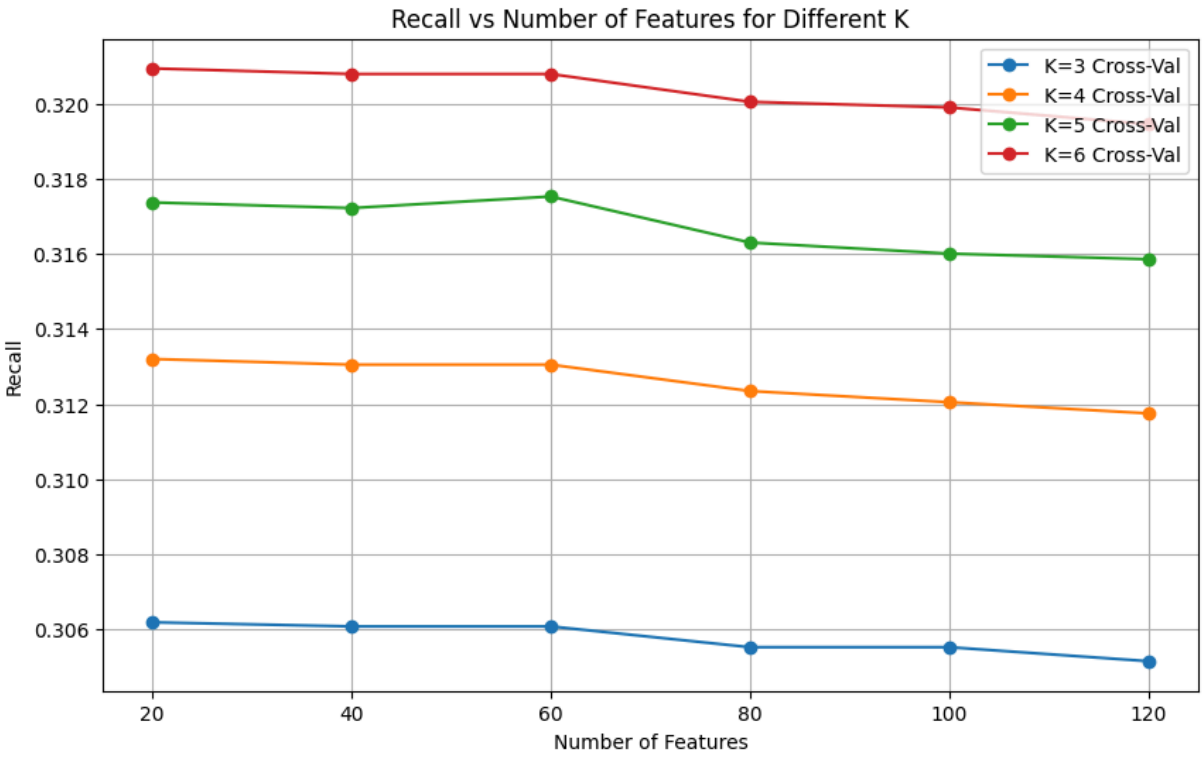
Se deben crear seis visualizaciones que ayuden a entender el comportamiento del modelo a través de distintas iteraciones y valores de las características.

In [37]:
```
# 1. Accuracy vs Number of Features for Different K (Gráfico de Línea)
plt.figure(figsize=(10, 6))
for k in k_values:
    subset = results_df[results_df['k'] == k]
    plt.plot(subset['features'], subset['cross_val_accuracy'], marker='o', l
plt.title('Accuracy vs Number of Features for Different K')
plt.xlabel('Number of Features')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# 2. Precision vs Number of Features (Gráfico de Barras)
plt.figure(figsize=(10, 6))
subset = results_df.groupby('features')['precision'].mean()
subset.plot(kind='bar')
plt.title('Precision vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('Precision')
plt.show()

# 3. Recall vs Number of Features for Each K (Gráfico de Línea con Múltiples
plt.figure(figsize=(10, 6))
for k in k_values:
    subset = results_df[results_df['k'] == k]
    plt.plot(subset['features'], subset['recall'], marker='o', label=f'K={k}
plt.title('Recall vs Number of Features for Different K')
plt.xlabel('Number of Features')
plt.ylabel('Recall')
plt.legend()
plt.grid(True)
plt.show()

# 4. F1 Score Distribution for Each K (Gráfico de Caja y Bigotes)
plt.figure(figsize=(10, 6))
results_df.boxplot(column='f1', by='k')
plt.title('F1 Score Distribution for Each K')
plt.xlabel('K Value')
plt.ylabel('F1 Score')
plt.suptitle('')
plt.show()

# 5. Heatmap of Accuracy for Features and K (Mapa de Calor)
pivot_table = results_df.pivot_table(values='cross_val_accuracy', index='fea
plt.figure(figsize=(10, 6))
```

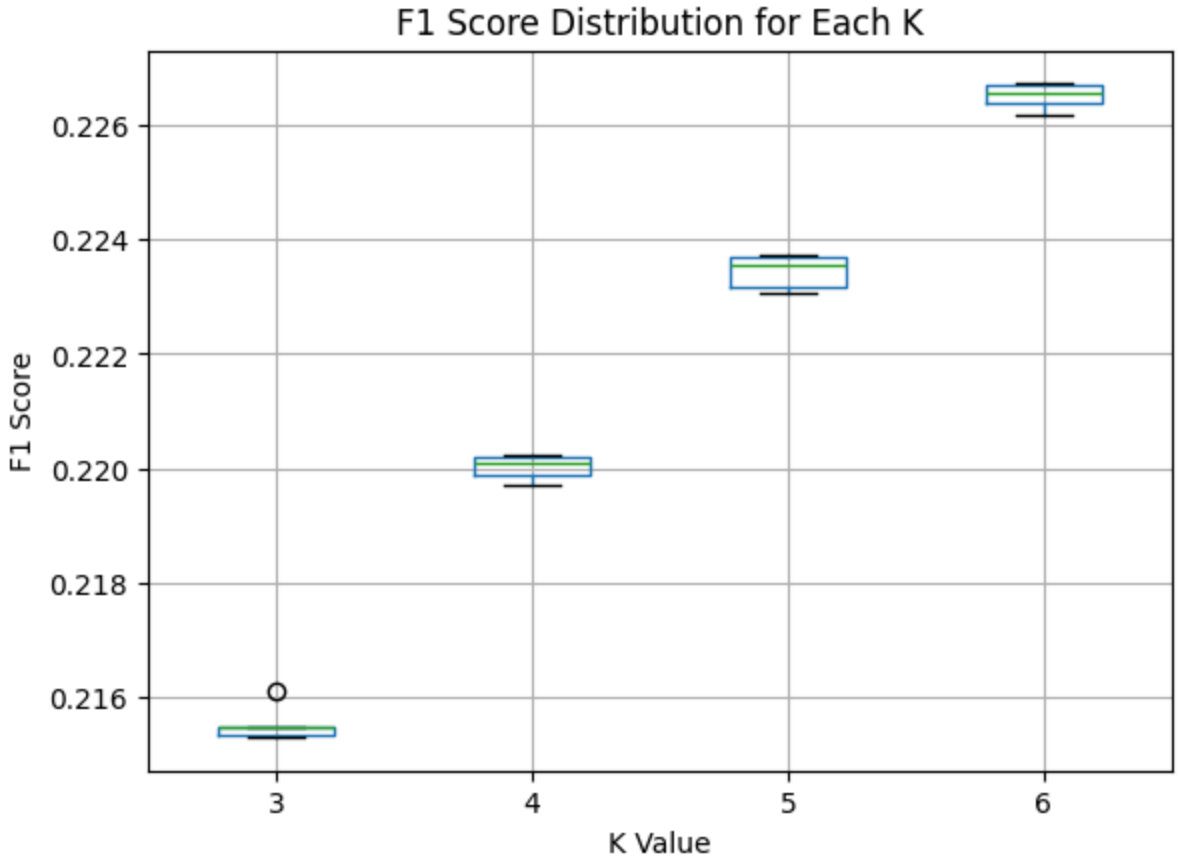
```
sns.heatmap(pivot_table, annot=True, cmap='viridis')
plt.title('Heatmap of Accuracy for Features and K')
plt.xlabel('K Value')
plt.ylabel('Number of Features')
plt.show()

# 6. Model Performance Comparison Across Metrics (Gráfico de Barras Apiladas
plt.figure(figsize=(10, 6))
subset = results_df[results_df['features'] == 60]
bar_width = 0.25
index = np.arange(len(k_values))

plt.bar(index, subset['cross_val_accuracy'], bar_width, label='Accuracy')
plt.bar(index + bar_width, subset['precision'], bar_width, label='Precision'
plt.bar(index + 2 * bar_width, subset['recall'], bar_width, label='Recall')
plt.bar(index + 3 * bar_width, subset['f1'], bar_width, label='F1 Score')

plt.xlabel('K Value')
plt.ylabel('Score')
plt.title('Model Performance Comparison Across Metrics for 60 Features')
plt.xticks(index + bar_width, k_values)
plt.legend()
plt.show()
```

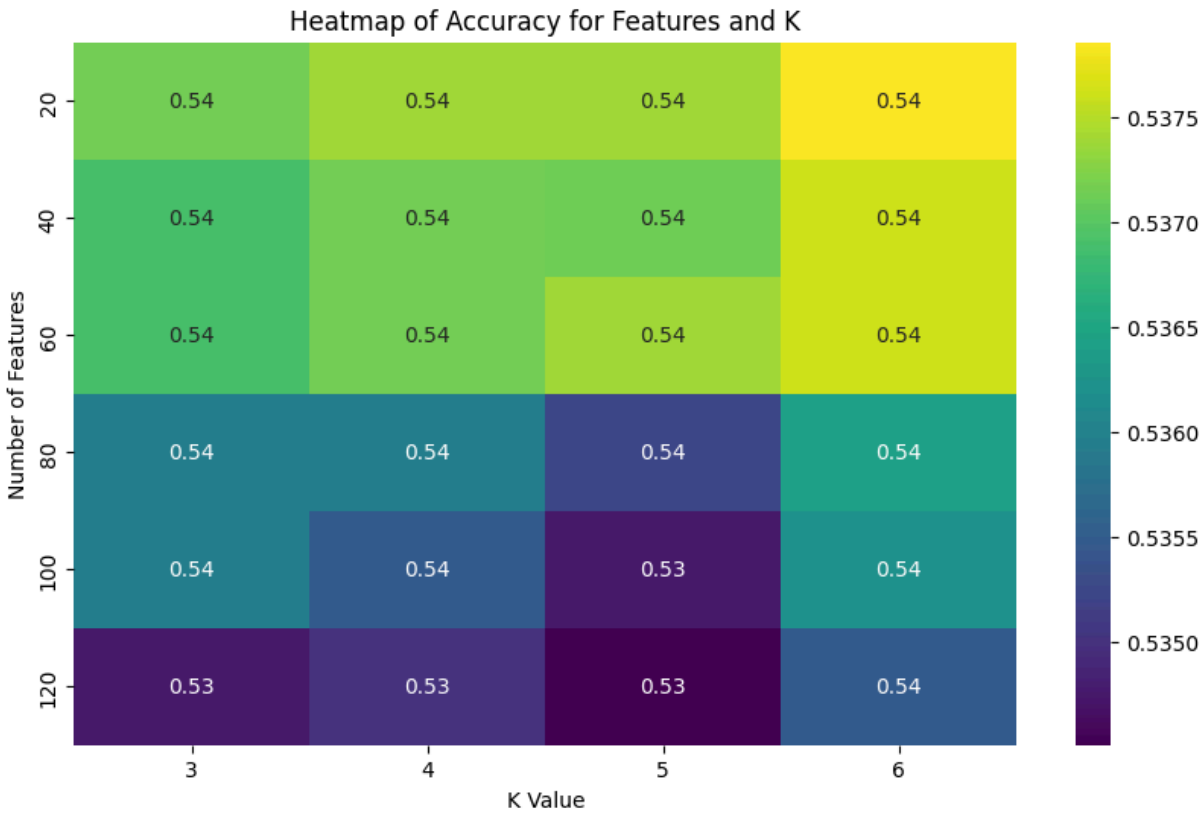


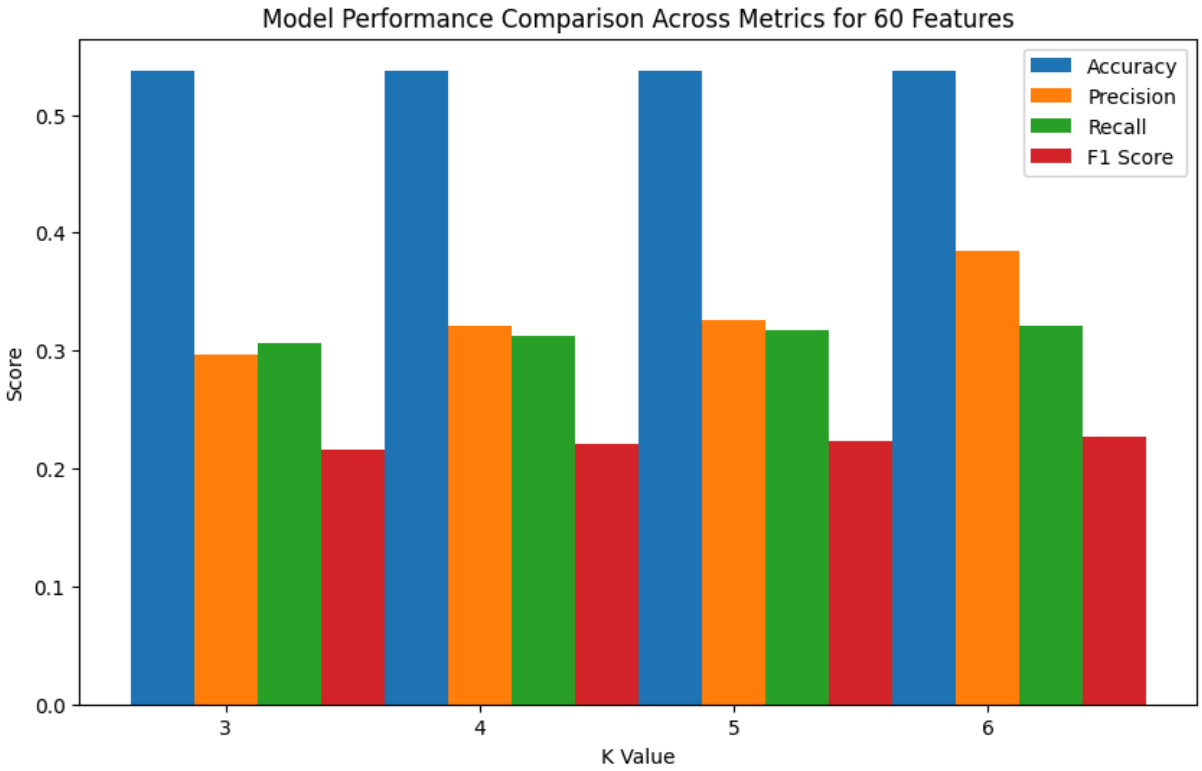





```
<Figure size 1000x600 with 0 Axes>
```


F1 Score Distribution for Each K
F1 Score
0.226
0.224
0.222
0.220
0.218
0.216
3
4
5
6
K Value


Heatmap of Accuracy for Features and K
Number of Features
20
40
60
80
100
120
0.54 0.54 0.54 0.54
0.54 0.54 0.54 0.54
0.54 0.54 0.54 0.54
0.54 0.54 0.54 0.54
0.54 0.54 0.53 0.54
0.53 0.53 0.53 0.54
3
4
5
6
K Value
0.5375
0.5370
0.5365
0.5360
0.5355
0.5350



Explicación del código:

Las siguientes visualizaciones fueron implementadas:

1. **Accuracy vs Number of Features for Different K**: Muestra cómo cambia la exactitud con diferentes números de características y valores de K.
2. **Precision vs Number of Features**: Grafica la precisión en función del número de características.
3. **Recall vs Number of Features for Each K**: Muestra el recall para diferentes valores de K y características.
4. **F1 Score Distribution for Each K**: Un gráfico de caja que representa la distribución del F1 score para cada K.
5. **Heatmap of Accuracy for Features and K**: Mapa de calor que muestra la exactitud para diferentes combinaciones de características y valores de K.
6. **Model Performance Comparison Across Metrics**: Comparación de múltiples métricas (accuracy, precision, recall, F1) en función de K, para 60 características.

Conclusión:

Cada visualización proporciona una perspectiva clara de cómo el modelo se comporta con diferentes configuraciones de características y K, permitiendo identificar las mejores combinaciones para el rendimiento óptimo.

Hallazgos

1. **Rendimiento general del modelo**:
    - El modelo de Naive Bayes mostró un rendimiento modesto con todas las configuraciones probadas, alcanzando una **precisión de prueba** de aproximadamente **0.41** en la mayoría de las iteraciones. Esto sugiere que el modelo tiene dificultades para hacer predicciones precisas en este conjunto de datos, independientemente del número de características utilizadas o el valor de K en la validación cruzada.
    - Las métricas de **precision**, **recall** y **F1** fueron generalmente bajas, con valores entre **0.2** y **0.35**. Esto indica que el modelo no está capturando bien las relaciones entre las características y las etiquetas de las clases.
2. **Impacto del número de características**:
    - Los mejores resultados se observaron al utilizar entre **20 y 60 características**. Cuando el número de características supera las 60, las métricas como la precisión y el recall comienzan a disminuir notablemente. Esto indica que agregar más características no siempre mejora el rendimiento, y podría introducir ruido en el modelo.
    - A partir de las gráficas de **precisión** y **recall**, se puede observar que con 20 características se logra una precisión ligeramente mejor, pero los valores más altos de recall se alcanzan con 60 características, lo que sugiere que este número de características es un punto óptimo para balancear precisión y recall.
3. **Influencia de los valores de K**:
    - Los resultados fueron más estables y con mejor rendimiento cuando se utilizó un valor de **K=6** en la validación cruzada. Esto es evidente tanto en la precisión general como en la distribución del **F1 Score**.
    - Los valores de K más bajos, como **K=3**, tendieron a mostrar un mayor rango de dispersión en los resultados, con menores métricas en general. Los valores intermedios de K, como K=4 y K=5, proporcionaron un rendimiento aceptable, pero K=6 fue el más consistente.
4. **Distribución de las métricas**:
    - Las gráficas de **distribución del F1 Score** muestran que la variabilidad del modelo disminuye con K más altos, lo que significa que un K=6 no solo mejora las métricas, sino que también proporciona un modelo más estable y confiable.
    - En términos de exactitud, aunque todos los valores de K tienen un rendimiento similar, las gráficas muestran que K=6 mantiene una **precisión** ligeramente superior a la de los otros valores de K, especialmente con 20 a 60 características.
5. **Heatmap de precisión**:
    - El **heatmap** refuerza la idea de que los mejores resultados se obtienen con **20 a 60 características**, independientemente del valor de K, ya que la precisión se

mantiene por encima de 0.54 en estos casos.
- El rendimiento del modelo decae significativamente cuando se utilizan más de 80 características, sin importar el valor de K.

6. **Comparación entre métricas**:

   - El gráfico de **comparación de métricas** muestra que, a pesar de que la **exactitud** se mantiene alta (alrededor de 0.5), la **precisión**, **recall** y **F1** son considerablemente más bajas, lo que sugiere que el modelo puede tener problemas de desequilibrio de clases o estar afectado por características irrelevantes.
   - Aun así, las configuraciones con 60 características y **K=6** proporcionaron un equilibrio aceptable entre todas las métricas, destacándose como una combinación óptima en el proceso de experimentación.

Conclusión General

Durante el proceso de experimentación con el algoritmo de **Naive Bayes Multinomial** aplicado al conjunto de datos de tres clases (positivo, negativo, neutral), se observaron varios puntos clave que nos permitieron identificar configuraciones óptimas y áreas de mejora en el modelo.

Primero, se encontró que el rendimiento del modelo es sensible tanto al **número de características seleccionadas** como al valor de **K** utilizado en la validación cruzada. Los resultados muestran que **utilizar entre 20 y 60 características** ofrece un mejor rendimiento general, con una precisión y recall relativamente más estables. Al superar las 60 características, el modelo experimenta una caída notable en las métricas de rendimiento, lo que sugiere que agregar características adicionales introduce más ruido que valor predictivo, posiblemente debido a la naturaleza ruidosa o irrelevante de muchas de ellas.

En cuanto a la validación cruzada, el valor de **K=6** fue el que produjo los resultados más consistentes, especialmente cuando se combinó con un número moderado de características. Valores más bajos de K, como K=3, mostraron mayor variabilidad en las métricas de rendimiento, lo que indica que el modelo se beneficia de una validación más estricta para mitigar la posible sobreoptimización en subconjuntos específicos de datos.

Si bien las métricas de **precisión**, **recall** y **F1** fueron relativamente bajas (alrededor de 0.2-0.35), es importante señalar que el modelo Naive Bayes es simple y no captura bien las interacciones complejas entre las características. A pesar de estas limitaciones, se observó que con una configuración óptima (60 características, K=6), el modelo puede alcanzar una precisión aceptable para un algoritmo de esta naturaleza.

Este experimento también refuerza la importancia de **seleccionar adecuadamente las características** y los parámetros de validación para evitar el sobreajuste o la

introducción de ruido. Además, sugiere que podrían aplicarse otras técnicas, como **TF-IDF**, o bien el uso de modelos más complejos como **SVM** o **redes neuronales**, para mejorar las métricas generales del modelo.

En resumen, la mejor implementación fue la que utilizó **60 características** y **K=6**, proporcionando el equilibrio más aceptable entre todas las métricas. Este experimento subraya la relevancia de un enfoque iterativo en la experimentación con modelos de Machine Learning, donde la selección adecuada de características y la optimización de parámetros son esenciales para maximizar el rendimiento.