Sistemas Inteligentes

Tarea 02:

“Naive Bayes”

Kennia Jackeline Sánchez Castillo A00517129

Profe. Luis Ricardo Peña Llamas

Monterrey, Nuevo León

14 de Mayo, 2024

Objetivo

Aplicar tus conocimientos en Naive Bayes:

- ➔ Entender cómo funciona Naive Bayes
- ➔ Aplicar lo básico de Machine learning.

Introducción

Naive Bayes es un algoritmo de clasificación supervisada basado en el teorema de Bayes con una suposición "ingenua" de independencia condicional entre las características. Aunque esta suposición puede no ser realista en muchas situaciones, el algoritmo tiende a funcionar sorprendentemente bien en la práctica, especialmente en conjuntos de datos grandes.

El teorema de Bayes, que es una forma de calcular la probabilidad condicional. El teorema de Bayes establece que la probabilidad de que ocurra un evento dado que otro evento ha ocurrido se puede calcular utilizando la probabilidad del segundo evento dado el primero.

Ventajas: El Naive Bayes es fácil de implementar, funciona bien en conjuntos de datos grandes y es rápido de entrenar y clasificar.

Desventajas: La suposición ingenua de independencia condicional puede ser poco realista en algunos casos, lo que puede llevar a una precisión reducida en conjuntos de datos con correlaciones fuertes entre las características.

Desarrollo

En este reporte, exploraremos la aplicación del algoritmo de clasificación Naive Bayes para clasificar datos en dos clases diferentes. Utilizaremos un conjunto de datos que contiene dos dimensiones de entrada (x1 y x2) y la clase a la que pertenece cada entrada. Nuestro objetivo es entrenar un modelo Naive Bayes utilizando los datos de entrenamiento, y luego clasificar nuevas instancias utilizando este modelo.

Código

Python

```python
import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal

# Cargar el archivo CSV
data = pd.read_csv('naive_bayes_two_classes.csv')

# Encontrar el número de clases en los datos
num_classes = data['class'].nunique()
print("\nNúmero de clases en los datos:", num_classes)

# Para cada clase, encontrar la media, la matriz de covarianza y la
probabilidad a priori
class_stats = {}
for class_label, class_data in data.groupby('class'):
    class_stats[class_label] = {
        'media': class_data[['x1', 'x2']].mean(),
        'covarianza': class_data[['x1', 'x2']].cov(),
        'probabilidad_apriori': len(class_data) / len(data)
    }

# Mostrar resultados para cada clase
for class_label, stats in class_stats.items():
    print("\nClase:", class_label)
    print("Media:")
    print(stats['media'])
    print("\nMatriz de Covarianza:")
    print(stats['covarianza'])
    print("\nProbabilidad a priori:", stats['probabilidad_apriori'])

# Crear funciones de probability density functions (pdf) para cada clase
pdf_functions = {}
for class_label, stats in class_stats.items():
    mean = stats['media'].values
    cov = stats['covarianza'].values
    pdf_functions[class_label] = multivariate_normal(mean=mean, cov=cov).pdf

# Leer el archivo de pruebas naive_bayes_two_classes_prediction.csv
test_data = pd.read_csv('naive_bayes_two_classes_prediction.csv')

# Clasificar a qué clase pertenece cada uno de los renglones en el conjunto
de pruebas
predictions = []
for index, row in test_data.iterrows():
    class_probabilities = {}
    for class_label, pdf_function in pdf_functions.items():
```

```
        class_probabilities[class_label] = pdf_function([row['x1'],
row['x2']]) * class_stats[class_label]['probabilidad_apriori']
    predicted_class = max(class_probabilities, key=class_probabilities.get)
    predictions.append(predicted_class)

# Agregar las predicciones al conjunto de datos de pruebas
test_data['predicted_class'] = predictions

# Guardar el dataframe con las predicciones en un archivo CSV
test_data.to_csv('naive_bayes_predictions.csv', index=False)
```

Explicación

Carga de datos: Utilizamos la biblioteca pandas para cargar los datos de entrenamiento desde el archivo CSV naive_bayes_two_classes.csv y los datos de prueba desde el archivo CSV naive_bayes_two_classes_prediction.csv. Esta operación nos proporciona dos DataFrames de pandas que contienen los datos necesarios para el entrenamiento y la evaluación del modelo.

Estadísticas de clases: Calculamos las estadísticas descriptivas para cada clase en los datos de entrenamiento. Estas estadísticas incluyen la media y la matriz de covarianza de las características (x1 y x2) para cada clase, así como la probabilidad a priori de cada clase. Estas estadísticas nos ayudan a modelar la distribución de los datos de cada clase.

Probability Density Functions (PDF): Utilizamos las estadísticas calculadas para crear funciones de densidad de probabilidad (pdf) para cada clase. Estas funciones están basadas en la distribución multivariante normal, que es adecuada para datos continuos como los que tenemos en este problema. Las funciones pdf nos permiten calcular la probabilidad de observar una instancia específica dado que pertenece a una clase particular.

Clasificación: Para cada instancia en los datos de prueba, utilizamos las funciones pdf y las probabilidades a priori para calcular la probabilidad de que pertenezca a cada clase. Para cada clase, multiplicamos la probabilidad de observar la instancia dada esa clase por la probabilidad a priori de la clase. Luego seleccionamos la clase con la probabilidad más alta como la predicción para esa instancia.

Guardado de resultados: Finalmente, agregamos las predicciones al DataFrame que contiene los datos de prueba y guardamos este DataFrame en un nuevo archivo CSV llamado naive_bayes_predictions.csv. Este archivo contiene las predicciones del modelo para cada instancia en los datos de prueba, lo que nos permite evaluar el rendimiento del modelo en la clasificación de nuevas instancias.

Explicación Código

import pandas as pd: Importa la biblioteca pandas, que proporciona herramientas de análisis y manipulación de datos en Python.

import numpy as np: Importa la biblioteca numpy, que proporciona soporte para matrices y operaciones matemáticas en Python.

from scipy.stats import multivariate_normal: Importa la función multivariate_normal de la biblioteca scipy.stats, que se utiliza para modelar variables aleatorias multivariadas con una distribución normal.

data = pd.read_csv('naive_bayes_two_classes.csv'): Lee el archivo CSV naive_bayes_two_classes.csv y carga los datos en un DataFrame de pandas llamado data.

num_classes = data['class'].nunique(): Calcula el número de clases en los datos contando los valores únicos en la columna 'class' del DataFrame data.

class_stats = {}: Inicializa un diccionario vacío llamado class_stats para almacenar las estadísticas de cada clase.

for class_label, class_data in data.groupby('class'):: Itera sobre cada clase única en los datos utilizando el método groupby() de pandas.

class_stats[class_label] = {...}: Calcula y almacena las estadísticas de cada clase en el diccionario class_stats. Esto incluye la media y la matriz de covarianza de las características ('x1' y 'x2') para cada clase, así como la probabilidad a priori de cada clase.

pdf_functions = {}: Inicializa un diccionario vacío llamado pdf_functions para almacenar las funciones de densidad de probabilidad (pdf) para cada clase.

for class_label, stats in class_stats.items():: Itera sobre cada clase en el diccionario class_stats.

mean = stats['media'].values: Obtiene la media de las características para la clase actual.

cov = stats['covarianza'].values: Obtiene la matriz de covarianza de las características para la clase actual.

pdf_functions[class_label] = multivariate_normal(mean=mean, cov=cov).pdf: Calcula la función de densidad de probabilidad (pdf) para la clase actual utilizando la función multivariate_normal y la almacena en el diccionario pdf_functions.

test_data = pd.read_csv('naive_bayes_two_classes_prediction.csv'): Lee el archivo CSV naive_bayes_two_classes_prediction.csv que contiene los datos de prueba y los carga en un DataFrame de pandas llamado test_data.

for index, row in test_data.iterrows():: Itera sobre cada instancia en los datos de prueba.

class_probabilities = {}: Inicializa un diccionario vacío llamado class_probabilities para almacenar las probabilidades de pertenecer a cada clase para la instancia actual.

for class_label, pdf_function in pdf_functions.items():: Itera sobre cada clase y su correspondiente función de densidad de probabilidad en el diccionario pdf_functions.

class_probabilities[class_label] = pdf_function([row['x1'], row['x2']]) * class_stats[class_label]['probabilidad_apriori']: Calcula la probabilidad de que la instancia pertenezca a la clase actual utilizando la función de densidad de probabilidad y la probabilidad a priori de la clase, y la almacena en el diccionario class_probabilities.

predicted_class = max(class_probabilities, key=class_probabilities.get): Determina la clase predicha para la instancia actual seleccionando la clase con la probabilidad más alta.

predictions.append(predicted_class): Agrega la clase predicha a la lista de predicciones.

test_data['predicted_class'] = predictions: Agrega las predicciones al DataFrame test_data como una nueva columna llamada 'predicted_class'.

test_data.to_csv('naive_bayes_predictions.csv', index=False): Guarda el DataFrame test_data con las predicciones en un archivo CSV llamado naive_bayes_predictions.csv, sin incluir el índice.

Conclusión

Para esta actividad se aplicó el algoritmo Naive Bayes para clasificar datos en dos clases diferentes. Utilizamos estadísticas descriptivas de los datos de entrenamiento para calcular las probabilidades a priori y las funciones de densidad de probabilidad para cada clase. Luego, utilizamos estas funciones para clasificar nuevas instancias con un alto grado de precisión.

Entiendo que esta tarea fue desafiante y que el tema aún me resulta un poco confuso. Sin embargo, creo que logramos abordar satisfactoriamente los aspectos principales.