prac6

October 18, 2024

Aim: Implementing Naïve Bayes Classifier

Theory: ### Naïve Bayes Classifier:

The **Naïve Bayes classifier** is a probabilistic algorithm used for classification tasks. It is called "naïve" because it assumes that all features (or variables) are **independent** of each other when predicting the class, which is a strong assumption that is rarely true in real-world data. Despite this, Naïve Bayes often performs well in practice, especially for text-based tasks like spam filtering, sentiment analysis, and document classification.

Key Concepts:

1. **Bayesian Approach**:
    - Naïve Bayes is based on **Bayesian probability**, which provides a way to update the probability estimate of a class label based on new evidence (features).
    - The algorithm calculates the likelihood of a particular class label given the input features and selects the class with the highest probability.
2. **Conditional Independence**:
    - Naïve Bayes assumes that all features are **conditionally independent**, meaning the presence or absence of a feature does not influence the presence or absence of any other feature, given the class label.
    - In practice, this assumption is often violated, but the algorithm can still produce good results because it simplifies computations and works well when certain feature combinations are more informative than others.
3. **Types of Naïve Bayes Classifiers**: There are different versions of the Naïve Bayes classifier depending on the type of data:
    - **Gaussian Naïve Bayes**: Assumes that continuous data follows a Gaussian (normal) distribution. It is used for continuous features like temperature or height.
    - **Multinomial Naïve Bayes**: Typically used for discrete data, especially in text classification where features represent the frequency of words in a document.
    - **Bernoulli Naïve Bayes**: Best suited for binary/Boolean features, often used in scenarios where features represent whether a word occurs in a document (yes/no).
4. **Prediction Process**:
    - During training, the Naïve Bayes classifier learns the probability distribution of features for each class.

- For a given test instance, it calculates the probability of the instance belonging to each possible class, based on its features, and then assigns the class with the highest probability.

5. **Advantages**:
   - **Fast and efficient**: Since the Naïve Bayes algorithm involves straightforward computations, it is fast and works well even with large datasets.
   - **Works with small datasets**: It performs well even when the dataset is small because it relies on probabilities rather than complex pattern recognition.
   - **Handles multi-class problems**: Naïve Bayes naturally supports multi-class classification, unlike some other algorithms that focus on binary classification.
6. **Disadvantages**:
   - **Independence assumption**: The assumption that features are independent can lead to inaccurate results in cases where features are highly correlated.
   - **Data scarcity**: If the model encounters a feature combination it hasn't seen during training (i.e., a feature with zero probability), it might incorrectly assume that class is impossible. Techniques like **Laplace smoothing** are used to handle this issue.
7. **Applications**:
   - **Spam detection**: It can classify emails as spam or not spam based on the occurrence of specific words.
   - **Sentiment analysis**: Naïve Bayes is used to analyze text and determine if the sentiment is positive or negative.
   - **Document classification**: It can categorize text documents into different topics, such as politics, sports, or technology, based on word frequency.

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
```

```
# You can also write temporary files to /kaggle/temp/, but they won't be saved
↪outside of the current session
```

/kaggle/input/drug-classification/drug200.csv

[ ]:
```
df = pd.read_csv("/kaggle/input/drug-classification/drug200.csv")
```

[ ]:
```
df.head(5)
```

[ ]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

[ ]:
```
df.isnull().sum()
```

[ ]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

[ ]:
```
df.value_counts('BP')
```

[ ]:
```
BP
HIGH      77
LOW       64
NORMAL    59
Name: count, dtype: int64
```

[ ]:
```
df.value_counts('Cholesterol')
```

[ ]:
```
Cholesterol
HIGH      103
NORMAL     97
Name: count, dtype: int64
```

[ ]:
```
df.value_counts('Drug')
```

[ ]:
```
Drug
DrugY    91
drugX    54
drugA    23
```

```
drugB    16
drugC    16
Name: count, dtype: int64
```

[ ]:
```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(categories=[['LOW','NORMAL', 'HIGH']])
df['BP'] = encoder.fit_transform(df[['BP']])
```

[ ]:
```
encoder = OrdinalEncoder(categories=[['NORMAL', 'HIGH']])
df['Cholesterol'] = encoder.fit_transform(df[['Cholesterol']])
```

[ ]:
```
encoder = OrdinalEncoder(categories=[['DrugY', 'drugX', 'drugA', 'drugB',␣
↪'drugC']])
df['Drug'] = encoder.fit_transform(df[['Drug']])
```

[ ]:
```
encoder = OrdinalEncoder(categories=[['F', 'M']])
df['Sex'] = encoder.fit_transform(df[['Sex']])
```

[ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

[ ]:
```
X = df.drop('Drug', axis=1)
y = df['Drug']
```

[ ]:
```
df.shape
```

[ ]:
```
(200, 6)
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
gnb = GaussianNB()

param_grid = {
    'var_smoothing': np.logspace(0, -9, num=10)
}

grid_search = GridSearchCV(estimator=gnb, param_grid=param_grid, cv=5,␣
↪scoring='accuracy')

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

best_model = grid_search.best_estimator_
```

```python
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print("Test Set Accuracy:", test_accuracy)
```

```
Best Parameters: {'var_smoothing': 0.0001}
Best Cross-Validation Score: 0.94375
Test Set Accuracy: 1.0
```

Conclusion: Naïve Bayes is a simple yet powerful algorithm for classification, especially when dealing with high-dimensional data such as text. While the assumption of feature independence is rarely true, the algorithm's efficiency and strong performance on certain types of tasks make it a popular choice in the machine learning field.