w7 naive bayes

September 4, 2024

[1]:
```python
import numpy as np
import pandas as pd
```

some theory

- At the heart of the Naive Bayes classifier is Bayes' theorem, a fundamental concept in probability theory
- There are different variants of Naive Bayes classifiers, including Multinomial and Gaussian Naive Bayes. Multinomial Naive Bayes is commonly used for text data, while Gaussian Naive Bayes is suitable for continuous data with a normal distribution.
- Step 1: Install scikit-learn If you haven't already installed scikit-learn, you can do so using pip: !pip install scikit-learn

Step 2: Import Necessary Libraries from sklearn.feature_extraction.text import CountVectorizer from sklearn.naive_bayes import MultinomialNB from sklearn.metrics import accuracy_score, classification_report from sklearn.model_selection import train_test_split

Step 3: Prepare Your Data Assuming you have a dataset with text samples and corresponding labels (e.g., positive or negative sentiment), you should split the data into a training set and a test set. Here's an example: # Sample data texts = ["This is a positive review.", "Negative sentiment detected.", "A very positive experience.", "I didn't like this at all."] # Corresponding labels (1 for positive, 0 for negative) labels = [1, 0, 1, 0] # Split the data into a training set and a test set X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)

Step 4: Feature Extraction You need to convert the text data into numerical features. One common approach is to use the CountVectorizer, which counts the frequency of words in the text. Here's how to do it: vectorizer = CountVectorizer() X_train_vec = vectorizer.fit_transform(X_train) X_test_vec = vectorizer.transform(X_test)

Step 5: Train the Naïve Bayes Classifier Next, create and train the Naïve Bayes classifier. For text classification, the Multinomial Naïve Bayes classifier is commonly used: clf = MultinomialNB() clf.fit(X_train_vec, y_train)

Step 6: Make Predictions Once the classifier is trained, you can use it to make predictions on new data: y_pred = clf.predict(X_test_vec)

Step 7: Evaluate the Model Evaluate the model's performance using appropriate metrics: accuracy = accuracy_score(y_test, y_pred) report = classification_report(y_test, y_pred) print(f"Accuracy: {accuracy}") print(report)

This code demonstrates a basic implementation of the Naïve Bayes Classifier in Python using scikit-learn. Depending on your specific task and dataset, you may need to fine-tune the pre-processing steps, hyper parameters, and model selection to achieve the best performance.

0.0.1 q1

1. Implement in python program of the following problems using Bayes Theorem.

a) Of the students in the college, 60% of the students reside in the hostel and 40% of the students are day scholars. Previous year results report that 30% of all students who stay in the hostel scored A Grade and 20% of day scholars scored A grade. At the end of the year, one student is chosen at random and found that he/she has an A grade. What is the probability that the student is a hosteler?

b) Suppose you're testing for a rare disease, and you have the following information: The disease has a prevalence of 0.01 (1% of the population has the disease). The test is not perfect: The test correctly identifies the disease (true positive) 99% of the time (sensitivity).

The test incorrectly indicates the disease (false positive) 2% of the time (1 - specificity). Calculate the probability of having the disease given a positive test result using Bayes' theorem.

[3]:
```python
####a
p_hostel = 0.6
p_daySchol = 0.4
p_A_hostel = 0.3
p_A_daySchol = 0.2

p_A = p_A_hostel*p_hostel + p_A_daySchol*p_daySchol
print(p_A)

P_H_given_A = (p_A_hostel * p_hostel) / p_A
print("p_h_given A: ", P_H_given_A)
```

```
0.26
p_h_given A:  0.6923076923076923
```

[4]:
```python
#### b
# Given probabilities
P_D = 0.01  # Prevalence of the disease
P_T_given_D = 0.99  # Sensitivity (true positive rate)
P_T_given_not_D = 0.02  # False positive rate
P_not_D = 1 - P_D  # Probability of not having the disease

# Total probability of a positive test result
P_T = (P_T_given_D * P_D) + (P_T_given_not_D * P_not_D)

# Probability of having the disease given a positive test result
P_D_given_T = (P_T_given_D * P_D) / P_T
```

```
print(f"Probability of having the disease given a positive test result:␣
↪{P_D_given_T:.4f}")
```

```
Probability of having the disease given a positive test result: 0.3333
```

0.0.2 q2

Develop a function python code for Naïve Bayes classifier from scratch without using scikit-learn library, to predict whether the buyer should buy computer or not. Consider a following sample training dataset stored in a CSV file containing information about following buyer conditions (such as "<=30," "medium," "Yes," and "fair") and whether the player played golf ("Yes" or "No")

[26]:
```
import pandas as pd

# Define the dataset as a dictionary
data_dict = {
    'age': ['<=30', '<=30', '31…40', '>40', '>40', '>40', '31…40', '<=30',␣
↪'<=30', '>40', '<=30', '31…40', '>40'],
    'income': ['high', 'high', 'high', 'medium', 'low', 'low', 'low', 'medium',␣
↪'low', 'medium', 'medium', 'high', 'medium'],
    'student': ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes', 'no', 'yes',␣
↪'yes', 'yes', 'yes', 'no'],
    'credit_rating': ['fair', 'excellent', 'fair', 'fair', 'fair', 'excellent',␣
↪'excellent', 'fair', 'fair', 'fair', 'excellent', 'fair', 'excellent'],
    'buys_computer': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no',␣
↪'yes', 'yes', 'yes', 'yes', 'no']
}

df = pd.DataFrame(data_dict)
df.to_csv("buyDF.csv")
print("DataFrame:\n", df)

# Compute prior probabilities
prior_probs = df['buys_computer'].value_counts(normalize=True).to_dict()

print("\nPrior Probabilities:\n", prior_probs)
```

```
DataFrame:
       age  income student credit_rating buys_computer
0    <=30    high      no          fair            no
1    <=30    high      no     excellent            no
2   31…40    high      no          fair           yes
3     >40  medium      no          fair           yes
4     >40     low     yes          fair           yes
5     >40     low     yes     excellent            no
6   31…40     low     yes     excellent           yes
7    <=30  medium      no          fair            no
8    <=30     low     yes          fair           yes
```

```
9     >40  medium     yes        fair           yes
10   <=30  medium     yes   excellent           yes
11  31…40    high     yes        fair           yes
12    >40  medium      no   excellent            no

Prior Probabilities:
 {'yes': 0.6153846153846154, 'no': 0.38461538461538464}
```

[27]:
```
p_age = df['age'].value_counts(normalize=True).to_dict()
print(p_age)
p_age_lessThan30 = (p_age['<=30'])
p_age_31to40 = (p_age['31…40'])
p_age_moreThan40 = (p_age['>40'])
```

```
{'<=30': 0.38461538461538464, '>40': 0.38461538461538464, '31…40':
0.23076923076923078}
```

[28]:
```
p_income = df['income'].value_counts(normalize=True).to_dict()
p_income_high = p_income['high']
p_income_medium = p_income['medium']
p_income_low = p_income['low']
```

[29]:
```
p_student = df['student'].value_counts(normalize=True).to_dict()
p_student_y = p_student['yes']
p_student_n = p_student['no']
```

[49]:
```
#fn to get each features likelihoods given target class
def compute_likelihoods(df, feature, target):
    likelihoods = {}
    target_classes = df[target].unique() #yes / no in our case (buys_comp)
    print(feature)

    for cls in target_classes: #ones of yes/no
        print("Class:",cls)
        cls_data = df[df[target] == cls] ##all data with a given target
        print("class data for ", cls, ": ",cls_data, '\n')
        feature_counts = cls_data[feature].value_counts(normalize=True).
 ↪to_dict() # prob of feature=x given target
        print(feature_counts)
        likelihoods[cls] = feature_counts
        #print(likelihoods[cls])

    return likelihoods

features = ['age', 'income', 'student', 'credit_rating']
likelihoods = {}

for feat in features:
```

```
    likelihoods[feat] = compute_likelihoods(df, feat, 'buys_computer')

#BASICALLY : P(FEATURE1=V1 | TARGET = YES/NO)
for feat in likelihoods:
    print(feat,':', likelihoods[feat], end = '\n')
    print('\n')
```

```
age
Class: no
class data for  no :       age  income student credit_rating buys_computer
0   <=30    high      no          fair            no
1   <=30    high      no     excellent            no
5    >40     low     yes     excellent            no
7   <=30  medium      no          fair            no
12   >40  medium      no     excellent            no

{'<=30': 0.6, '>40': 0.4}
Class: yes
class data for  yes :        age  income student credit_rating buys_computer
2   31…40    high      no          fair           yes
3     >40  medium      no          fair           yes
4     >40     low     yes          fair           yes
6   31…40     low     yes     excellent           yes
8    <=30     low     yes          fair           yes
9     >40  medium     yes          fair           yes
10   <=30  medium     yes     excellent           yes
11  31…40    high     yes          fair           yes

{'31…40': 0.375, '>40': 0.375, '<=30': 0.25}
income
Class: no
class data for  no :       age  income student credit_rating buys_computer
0   <=30    high      no          fair            no
1   <=30    high      no     excellent            no
5    >40     low     yes     excellent            no
7   <=30  medium      no          fair            no
12   >40  medium      no     excellent            no

{'high': 0.4, 'medium': 0.4, 'low': 0.2}
Class: yes
class data for  yes :        age  income student credit_rating buys_computer
2   31…40    high      no          fair           yes
3     >40  medium      no          fair           yes
4     >40     low     yes          fair           yes
6   31…40     low     yes     excellent           yes
8    <=30     low     yes          fair           yes
9     >40  medium     yes          fair           yes
10   <=30  medium     yes     excellent           yes
```

```
11  31…40    high     yes          fair           yes

{'medium': 0.375, 'low': 0.375, 'high': 0.25}
student
Class: no
class data for  no :       age  income student credit_rating buys_computer
0   <=30    high      no          fair            no
1   <=30    high      no     excellent            no
5    >40     low     yes     excellent            no
7   <=30  medium      no          fair            no
12   >40  medium      no     excellent            no

{'no': 0.8, 'yes': 0.2}
Class: yes
class data for  yes :        age  income student credit_rating buys_computer
2   31…40    high      no          fair           yes
3     >40  medium      no          fair           yes
4     >40     low     yes          fair           yes
6   31…40     low     yes     excellent           yes
8    <=30     low     yes          fair           yes
9     >40  medium     yes          fair           yes
10   <=30  medium     yes     excellent           yes
11  31…40    high     yes          fair           yes

{'yes': 0.75, 'no': 0.25}
credit_rating
Class: no
class data for  no :       age  income student credit_rating buys_computer
0   <=30    high      no          fair            no
1   <=30    high      no     excellent            no
5    >40     low     yes     excellent            no
7   <=30  medium      no          fair            no
12   >40  medium      no     excellent            no

{'excellent': 0.6, 'fair': 0.4}
Class: yes
class data for  yes :        age  income student credit_rating buys_computer
2   31…40    high      no          fair           yes
3     >40  medium      no          fair           yes
4     >40     low     yes          fair           yes
6   31…40     low     yes     excellent           yes
8    <=30     low     yes          fair           yes
9     >40  medium     yes          fair           yes
10   <=30  medium     yes     excellent           yes
11  31…40    high     yes          fair           yes

{'fair': 0.75, 'excellent': 0.25}
age : {'no': {'<=30': 0.6, '>40': 0.4}, 'yes': {'31…40': 0.375, '>40': 0.375,
```

'<=30': 0.25}}

income : {'no': {'high': 0.4, 'medium': 0.4, 'low': 0.2}, 'yes': {'medium': 0.375, 'low': 0.375, 'high': 0.25}}

student : {'no': {'no': 0.8, 'yes': 0.2}, 'yes': {'yes': 0.75, 'no': 0.25}}

credit_rating : {'no': {'excellent': 0.6, 'fair': 0.4}, 'yes': {'fair': 0.75, 'excellent': 0.25}}

[50]:
```
def classify_instance(instance, prior_probs, likelihoods):
    posteriors = {}

    for cls, prior in prior_probs.items():
        posterior = prior

        for feature, value in instance.items():
            if feature in likelihoods and cls in likelihoods[feature]:
                posterior *= likelihoods[feature][cls].get(value, 1e-6)  # Add␣
↪small value to avoid zero probabilities

        posteriors[cls] = posterior

    # Normalize to get probabilities
    total = sum(posteriors.values())
    posteriors = {cls: prob / total for cls, prob in posteriors.items()}

    return posteriors
```

[51]:
```
new_instance = {
    'age': '<=30',
    'income': 'medium',
    'student': 'yes',
    'credit_rating': 'fair'
}

posterior_probs = classify_instance(new_instance, prior_probs, likelihoods)
print("Posterior Probabilities for New Instance:", posterior_probs)

# Predict the class with the highest posterior probability
predicted_class = max(posterior_probs, key=posterior_probs.get)
print("Predicted Class:", predicted_class)
```

```
Posterior Probabilities for New Instance: {'yes': 0.8146270818247646, 'no':
0.18537291817523538}
Predicted Class: yes
```

0.0.3 Q3

Write a Python function to implement the Naive Bayes classifier without using the scikit-learn library for the following sample training dataset stored as a .CSV file. Calculate the accuracy, precision, and recall for your train/test dataset.

a. Build a classifier that determines whether a text is about sports or not.

b. Determine which tag the sentence “A very close game” belongs to

[52]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Sample data
data = {
    'Text': [
        "A great game", "The election was over", "A very close game",
        "Very clean match", "A clean but forgettable game", "It was a close␣
 ↪election"
    ],
    'Tag': ['Sports', 'Not sports', 'Sports', 'Sports', 'Sports', 'Not sports']
}

# Create DataFrame
df = pd.DataFrame(data)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df['Text'], df['Tag'],␣
 ↪test_size=0.5, random_state=42)
```

[53]:
```python
from collections import defaultdict
import numpy as np

class NaiveBayesTextClassifier:
    def __init__(self):
        self.class_priors = {}
        self.word_likelihoods = defaultdict(lambda: defaultdict(lambda: 1e-6)) ␣
 ↪# Smooth with a small value
        self.vocab = set()
        self.classes = []

    def fit(self, X, y):
        # Compute prior probabilities
        total_docs = len(y)
        class_counts = y.value_counts()
```

```
        self.classes = class_counts.index.tolist()
        self.class_priors = {cls: count / total_docs for cls, count in␣
↪class_counts.items()}

        # Count words per class
        word_counts = defaultdict(lambda: defaultdict(int))
        class_word_counts = defaultdict(int)

        for text, label in zip(X, y):
            words = text.lower().split()
            self.vocab.update(words)
            for word in words:
                word_counts[label][word] += 1
                class_word_counts[label] += 1

        # Compute likelihoods
        for cls in self.classes:
            total_words = class_word_counts[cls]
            for word in self.vocab:
                self.word_likelihoods[cls][word] = (word_counts[cls][word] + 1)␣
↪/ (total_words + len(self.vocab))

    def predict(self, X):
        predictions = []
        for text in X:
            words = text.lower().split()
            class_probs = {}

            for cls in self.classes:
                prob = np.log(self.class_priors[cls])
                for word in words:
                    prob += np.log(self.word_likelihoods[cls].get(word, 1e-6))
                class_probs[cls] = prob

            # Predict the class with the highest probability
            predictions.append(max(class_probs, key=class_probs.get))

        return predictions
```

[54]:
```
# Instantiate and train the classifier
nb_classifier = NaiveBayesTextClassifier()
nb_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = nb_classifier.predict(X_test)

# Evaluate performance
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪classification_report

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='Sports',␣
 ↪average='binary')
recall = recall_score(y_test, y_pred, pos_label='Sports', average='binary')
report = classification_report(y_test, y_pred, target_names=['Not sports',␣
 ↪'Sports'])

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.33
Precision: 0.33
Recall: 1.00
Classification Report:
              precision    recall  f1-score   support

  Not sports       0.00      0.00      0.00         2
      Sports       0.33      1.00      0.50         1

    accuracy                           0.33         3
   macro avg       0.17      0.50      0.25         3
weighted avg       0.11      0.33      0.17         3


/usr/lib/python3/dist-packages/sklearn/metrics/_classification.py:1509:
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))
/usr/lib/python3/dist-packages/sklearn/metrics/_classification.py:1509:
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))
/usr/lib/python3/dist-packages/sklearn/metrics/_classification.py:1509:
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))
```

[55]:

```
# Classify a new sentence
new_sentence = "A very close game"
```

```python
prediction = nb_classifier.predict([new_sentence])[0]
print(f"The sentence '{new_sentence}' belongs to the class '{prediction}'.")
```

```
The sentence 'A very close game' belongs to the class 'Sports'.
```

[ ]:

```python

```