naive_bayes

September 4, 2024

[64]:

```
## Q1 solve the question using bayes theorum and then code the same

# a part
# given
P_H = 0.60
P_D = 0.40
P_A_given_H = 0.30
P_A_given_D = 0.20

# probability of A grade
P_A = (P_A_given_H * P_H) + (P_A_given_D * P_D)

# bayes' theorem to find P(H | A)
P_H_given_A = (P_A_given_H * P_H) / P_A

print(f"Probability that a student with an A grade is a hosteler: {P_H_given_A:.
  ↪3f}")

# b part
#given
P_D = 0.01
P_not_D = 0.99
P_pos_given_D = 0.99
P_pos_given_not_D = 0.02


# prob of p(D|pos)
P_D_given_pos = (P_pos_given_D * P_D)/((P_pos_given_D * P_D) +␣
  ↪(P_pos_given_not_D * P_not_D))

print(f"Probability that a person has a disease given the test is positive is :␣
  ↪{P_D_given_pos:.3f}")
```

```
0.26
Probability that a student with an A grade is a hosteler: 0.692
0.029700000000000004
Probability that a person has a disease given the test is positive is : 0.333
```

[62]:

```
## Q2

import pandas as pd
import numpy as np

class NaiveBayesClassifier:
    def __init__(self):
        self.class_probs = {} # basically the output values probability
        self.feature_probs = {} # the input features
        self.classes = []
        self.features = []

    def fit(self, X, y):
        # here x is a dataframe and y is a series, hence we get all the unique␣
 ↪values for outputs
        self.classes = np.unique(y)
        self.features = X.columns  # get features

        # we take the average probability of the output
        self.class_probs = {cls: np.mean(y == cls) for cls in self.classes}

        # creating a suitable dict structure
        self.feature_probs = {cls: {feature: {} for feature in self.features}␣
 ↪for cls in self.classes}

        # creating the conditional probability
        for cls in self.classes:
            class_data = X[y == cls] # gets the features for a certain output␣
 ↪class
            for feature in self.features:
                feature_data = class_data[feature]
                values, counts = np.unique(feature_data, return_counts=True)#␣
 ↪gets all the unique values and returns an array of the frequncy
                probs = dict(zip(values, counts / len(feature_data)))# convert␣
 ↪to probability and place in a temp dict
                self.feature_probs[cls][feature] = probs


    def predict(self, X):
        predictions = []
        # the math behind the working, we get the proportionality not the␣
 ↪actual value
        # P(Ck F1,F2,…,Fn) P(Ck) [P(F1 Ck) P(F2 Ck) … P(Fn Ck)]
        for _, row in X.iterrows():
            class_probs = {}
            for cls in self.classes:
                prob = self.class_probs[cls]
                for feature in self.features:
                    value = row[feature]
                    prob *= self.feature_probs[cls].get(feature).get(value)
                class_probs[cls] = prob

        return class_probs

data = pd.read_csv('buy_computer.csv')

# split features
X = data.drop('Buys Computer', axis=1)
y = data['Buys Computer']

nb_classifier = NaiveBayesClassifier()
nb_classifier.fit(X, y)

# Example prediction
test_data = pd.DataFrame({
    'Age': ['<=30'],
    'Income': ['High'],
    'Student': ['Yes'],
    'Credit Rating': ['Fair']
})

# predict
# normalising the predictions we can get the actual probability
predictions = nb_classifier.predict(test_data)
total = 0
for i in predictions.keys():
    total += predictions.get(i)
res = {}
for i in predictions.keys():
    res[i] = predictions.get(i)/total

print(f"Predictions: {res}")
```

```
Predictions: {'No': 0.5714285714285714, 'Yes': 0.42857142857142866}
```

[65]:

```
import pandas as pd

# Create a DataFrame with sample data
data = {
    'text': [
        'The game was exciting and intense',
        'The match ended with a close score',
        'The team won the championship',
        'The weather was sunny and warm',
        'She went to the beach and relaxed',
        'They watched a movie together',
        'The player scored a last-minute goal',
        'He enjoyed a delicious meal'
    ],
    'label': [
        'sports',
        'sports',
        'sports',
        'not sports',
        'not sports',
        'not sports',
        'sports',
        'not sports'
    ]
}

df = pd.DataFrame(data)

# Save the DataFrame to a CSV file
df.to_csv('text_data.csv', index=False)
```

[86]:

```
# Q3

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

class NaiveBayesClassifier:
    def __init__(self):
        self.class_probs = {}
        self.feature_probs = {}
        self.classes = []
        self.feature_names = []

    def fit(self, X, y):
        self.classes = np.unique(y)
        self.feature_names = X.columns

        self.class_probs = {cls: np.mean(y == cls) for cls in self.classes}

        self.feature_probs = {cls: {feature: {} for feature in self.
 ↪feature_names} for cls in self.classes}

        for cls in self.classes:
            class_data = X[y == cls]
            for feature in self.feature_names:
                feature_data = class_data[feature]
                values, counts = np.unique(feature_data, return_counts=True)
                probs = dict(zip(values, counts / len(feature_data)))
                self.feature_probs[cls][feature] = probs

    def predict(self, X):
        predictions = []
        for _, row in X.iterrows():
            class_probs = {}
            for cls in self.classes:
                prob = self.class_probs[cls]
                for feature in self.feature_names:
                    value = row[feature]
                    prob *= self.feature_probs[cls].get(feature, {}).get(value, 1e-6)
                class_probs[cls] = prob
            predictions.append(max(class_probs, key=class_probs.get))
        return class_probs

data = pd.read_csv('text_data.csv')

# vectorize the text data
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['text']).toarray()
X = pd.DataFrame(X, columns=vectorizer.get_feature_names_out())

y = data['label']


nb_classifier = NaiveBayesClassifier()
nb_classifier.fit(X, y)

# new sentence
new_sentence = ["A very close game"]
X_new = vectorizer.transform(new_sentence).toarray()
X_new = pd.DataFrame(X_new, columns=vectorizer.get_feature_names_out())
predictions = nb_classifier.predict(X_new)
# print("cool till here")
res = {}
total = 0
for i in prediction.keys():
    total+=predictions.get(i)

for i in prediction.keys():
    res[i] = prediction.get(i)/total

print(f"Prediction for {new_sentence}: {res}")
```

```
Prediction for ['A very close game']: {'not sports': 2.999991000027e-06,
'sports': 0.999997000009}
```

[ ]:

```
```

[ ]:

```
```