NAivebayes

September 6, 2024

[21]:
```
P_H = 0.60
P_D = 0.40
P_A_given_H = 0.30
P_A_given_D = 0.20

P_A = (P_A_given_H * P_H) + (P_A_given_D * P_D)

P_H_given_A = (P_A_given_H * P_H) / P_A

print(f"The probability that the student is a hosteler given an A grade is:␣
 ↪{P_H_given_A:.2f}")
```

```
The probability that the student is a hosteler given an A grade is: 0.69
```

[22]:
```
P_D = 0.01
P_not_D = 0.99
P_T_given_D = 0.99
P_T_given_not_D = 0.02

P_T = (P_T_given_D * P_D) + (P_T_given_not_D * P_not_D)

P_D_given_T = (P_T_given_D * P_D) / P_T

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

```
The probability of having the disease given a positive test result is: 0.3333
```

[23]:
```
import pandas as pd
from collections import defaultdict
import numpy as np

data = {
    'age': ['<=30', '<=30', '31…40', '>40', '>40', '>40', '31…40', '<=30',␣
 ↪'<=30', '>40', '<=30', '31…40', '31…40', '>40'],
    'income': ['high', 'high', 'high', 'medium', 'low', 'low', 'low', 'medium',␣
 ↪'low', 'medium', 'medium', 'medium', 'high', 'medium'],
```

```
    'student': ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes', 'no', 'yes',␣
 ↪'yes', 'yes', 'no', 'yes', 'no'],
    'credit_rating': ['fair', 'excellent', 'fair', 'fair', 'fair', 'excellent',␣
 ↪'excellent', 'fair', 'fair', 'fair', 'excellent', 'excellent', 'fair',␣
 ↪'excellent'],
    'buys_computer': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no',␣
 ↪'yes', 'yes', 'yes', 'yes', 'yes', 'no']
}

df = pd.DataFrame(data)

df_encoded = pd.get_dummies(df, columns=['age', 'income', 'student',␣
 ↪'credit_rating'])

X = df_encoded.drop('buys_computer', axis=1)
y = df_encoded['buys_computer']

print(X.head())
print(y.head())
```

```
   age_31…40  age_<=30  age_>40  income_high  income_low  income_medium  \
0      False      True    False         True       False          False
1      False      True    False         True       False          False
2       True     False    False         True       False          False
3      False     False     True        False       False           True
4      False     False     True        False        True          False

   student_no  student_yes  credit_rating_excellent  credit_rating_fair
0        True        False                    False                True
1        True        False                     True               False
2        True        False                    False                True
3        True        False                    False                True
4       False         True                    False                True
0     no
1     no
2    yes
3    yes
4    yes
Name: buys_computer, dtype: object
```

[24]:
```
def calculate_probabilities(X, y):
    classes = y.unique()

    prior_probs = defaultdict(float)
    likelihoods = defaultdict(lambda: defaultdict(float))

    total_count = len(y)
```

```
    for cls in classes:
        prior_probs[cls] = np.mean(y == cls)

    for cls in classes:
        X_cls = X[y == cls]
        total_cls_count = len(X_cls)
        for column in X.columns:
            for value in X[column].unique():
                likelihoods[cls][(column, value)] = (X_cls[column] == value).
 ↪sum() / total_cls_count

    return prior_probs, likelihoods

prior_probs, likelihoods = calculate_probabilities(X, y)
```

[25]:
```
def predict(new_data, prior_probs, likelihoods):
    predictions = []

    for _, row in new_data.iterrows():
        posteriors = defaultdict(float)
        for cls in prior_probs:
            posterior = prior_probs[cls]
            for column, value in row.items():
                posterior *= likelihoods[cls].get((column, value), 1e-6)
            posteriors[cls] = posterior

        total = sum(posteriors.values())
        if total > 0:
            posteriors = {cls: p / total for cls, p in posteriors.items()}

        predictions.append(max(posteriors, key=posteriors.get))

    return predictions

new_data = pd.DataFrame({
    'age_<=30': [1, 0],
    'age_31…40': [0, 1],
    'age_>40': [0, 0],
    'income_high': [1, 0],
    'income_medium': [0, 1],
    'income_low': [0, 0],
    'student_no': [1, 0],
    'student_yes': [0, 1],
    'credit_rating_fair': [1, 0],
    'credit_rating_excellent': [0, 1]
})
```

```
predictions = predict(new_data, prior_probs, likelihoods)

print("Predictions:", predictions)
```

```
Predictions: ['no', 'yes']
```

[26]:
```
import pandas as pd
from collections import defaultdict
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

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

df = pd.DataFrame(data)

vectorizer = CountVectorizer()
X_encoded = vectorizer.fit_transform(df['Text']).toarray()

X_df = pd.DataFrame(X_encoded, columns=vectorizer.get_feature_names_out())

X = X_df
y = df['Tag']
print(X_df.head())
```

```
   but  clean  close  election  forgettable  game  great  it  match  over  \
0    0      0      0         0            0     1      1   0      0     0
1    0      0      0         1            0     0      0   0      0     1
2    0      1      0         0            0     0      0   0      1     0
3    1      1      0         0            1     1      0   0      0     0
4    0      0      1         1            0     0      0   1      0     0

   the  very  was
0    0     0    0
1    1     0    1
2    0     1    0
3    0     0    0
4    0     0    1
```

[27]:
```
def calculate_probabilities(X, y):
    # Get the classes
    classes = y.unique()

    prior_probs = defaultdict(float)
```

```
    likelihoods = defaultdict(lambda: defaultdict(float))

    total_count = len(y)
    for cls in classes:
        prior_probs[cls] = np.mean(y == cls)

    for cls in classes:
        X_cls = X[y == cls]
        total_cls_count = len(X_cls)
        for column in X.columns:
            for value in [0, 1]:
                likelihoods[cls][(column, value)] = (X_cls[column] == value).
 ↪sum() / total_cls_count

    return prior_probs, likelihoods

prior_probs, likelihoods = calculate_probabilities(X, y)
```

[28]:
```
def predict(new_data, prior_probs, likelihoods):
    predictions = []

    for _, row in new_data.iterrows():
        posteriors = defaultdict(float)
        for cls in prior_probs:
            posterior = prior_probs[cls]
            for column, value in row.items():
                posterior *= likelihoods[cls].get((column, value), 1e-6)
            posteriors[cls] = posterior

        total = sum(posteriors.values())
        if total > 0:
            posteriors = {cls: p / total for cls, p in posteriors.items()}

        predictions.append(max(posteriors, key=posteriors.get))

    return predictions

y_pred = predict(X, prior_probs, likelihoods)

accuracy = accuracy_score(y, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y, y_pred,␣
 ↪average='binary', pos_label='Sports')

print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
```

```
def predict_new_text(texts, vectorizer, prior_probs, likelihoods):
    X_new = vectorizer.transform(texts).toarray()
    X_new_df = pd.DataFrame(X_new, columns=vectorizer.get_feature_names_out())

    return predict(X_new_df, prior_probs, likelihoods)

new_texts = ["A very close game", "The election was over",]

predicted_tags = predict_new_text(new_texts, vectorizer, prior_probs,␣
 ↪likelihoods)
print("Predicted tags for new texts:", predicted_tags)
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
Predicted tags for new texts: ['Sports', 'Not sports']
```