02 - Naive Bayesian - Multinomial - Lab

August 26, 2021

0.0.1 === Task ===

1) Learn about TFidVectorizer and replace CountVectorizer with TFIDVectorizer (Explanation Provided in the Lecture)
2) Put Multinomial Naive Classification into a class that can transform the data, fit the model and do prediction.
    - In the class, allow users to choose whether to use CountVectorizer or TFIDVectorizer to transform the data.

[1]:
```
from sklearn.datasets import fetch_20newsgroups
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import label_binarize
from sklearn.metrics import average_precision_score, classification_report

data = fetch_20newsgroups()
```

[2]:
```
categories = ['talk.religion.misc', 'soc.religion.christian',
              'sci.space', 'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)

train_data = train.data
test_data = test.data
```

[3]:
```
def likelihood(X_class, laplace=1):
    return (np.sum(X_class, axis=0) + laplace) / (np.sum(np.sum(X_class,␣
 ↪axis=1)) + laplace)

def prior(X_class, m):
    return X_class.shape[0] / m
```

1 Put Multinomial Naive Classification into a class that can transform the data, fit the model and do prediction.

[4]:
```
class MultinomialNB():

    def __init__(self):
        self.priors = None
        self.likelihoods = None

    def transform_data(self, train_data, test_data, method):
        if method == 'CountVectorizer':
            vectorizer = CountVectorizer()
            X_train = vectorizer.fit_transform(train_data)
            X_test = vectorizer.transform(test_data)
            X_test = X_test.toarray()
            y_train = train.target
            y_test = test.target
            return X_train, y_train, X_test, y_test

        elif method == 'TFidVectorizer':
            vectorizer = TfidfVectorizer()
            X_train = vectorizer.fit_transform(train_data)
            X_test = vectorizer.transform(test_data)
            X_test = X_test.toarray()
            y_train = train.target
            y_test = test.target
            return X_train, y_train, X_test, y_test

    def fit(self, X_train, y_train):
        m, n = X_train.shape
        classes = np.unique(y_train)  #list of class
        k = len(classes) #number of class

        priors = np.zeros(k) #prior for each classes
        likelihoods = np.zeros((k, n)) #likehood for each class of each feature

        for idx, label in enumerate(classes):
            X_train_c = X_train[y_train==label]
            priors[idx] = prior(X_train_c, m)
            likelihoods[idx, :] = likelihood(X_train_c)

        self.priors = priors
        self.likelihoods = likelihoods

    def predict(self, X_test):
        yhat = np.log(self.priors) + X_test @ np.log(self.likelihoods.T)
        return np.argmax(yhat, axis=1)
```

[5]:
```
# TFidVectorizer
model = MultinomialNB()
X_train, y_train, X_test, y_test = model.transform_data(train_data, test_data,␣
 ↪method='TFidVectorizer')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
```

[6]:
```
n_classes = len(np.unique(y_test))

print("Accuracy: ", np.sum(yhat == y_test)/len(y_test))

print("=========Average precision score=======")
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2, 3])
yhat_binarized = label_binarize(yhat, classes=[0, 1, 2, 3])

for i in range(n_classes):
    class_score = average_precision_score(y_test_binarized[:, i],␣
 ↪yhat_binarized[:, i])
    print(f"Class {i} score: ", class_score)

print("=========Classification report=======")
print("Report: ", classification_report(y_test, yhat))
```

```
Accuracy:  0.630586592178771
=========Average precision score=======
Class 0 score:  0.8428735046171963
Class 1 score:  0.6681589584508366
Class 2 score:  0.42851478060694537
Class 3 score:  0.3326836615720188
=========Classification report=======
Report:                precision    recall  f1-score   support

           0       0.89      0.93      0.91       389
           1       1.00      0.55      0.70       394
           2       1.00      0.21      0.35       398
           3       0.34      0.98      0.50       251

    accuracy                           0.63      1432
   macro avg       0.81      0.66      0.61      1432
weighted avg       0.85      0.63      0.62      1432
```

[7]:
```
# CountVectorizer
model = MultinomialNB()
X_train, y_train, X_test, y_test = model.transform_data(train_data, test_data,␣
 ↪method='CountVectorizer')
model.fit(X_train, y_train)
```

```
yhat = model.predict(X_test)
```

[8]:
```
n_classes = len(np.unique(y_test))

print("Accuracy: ", np.sum(yhat == y_test)/len(y_test))

print("=========Average precision score=======")
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2, 3])
yhat_binarized = label_binarize(yhat, classes=[0, 1, 2, 3])

for i in range(n_classes):
    class_score = average_precision_score(y_test_binarized[:, i],␣
 ↪yhat_binarized[:, i])
    print(f"Class {i} score: ", class_score)

print("=========Classification report=======")
print("Report: ", classification_report(y_test, yhat))
```

```
Accuracy:  0.9287709497206704
=========Average precision score=======
Class 0 score:  0.9023786499946543
Class 1 score:  0.9192582932480275
Class 2 score:  0.8866233445509671
Class 3 score:  0.7813002245920824
=========Classification report=======
Report:                precision    recall  f1-score   support

           0       0.93      0.96      0.94       389
           1       0.96      0.94      0.95       394
           2       0.95      0.91      0.93       398
           3       0.86      0.88      0.87       251

    accuracy                           0.93      1432
   macro avg       0.92      0.92      0.92      1432
weighted avg       0.93      0.93      0.93      1432
```