Multinomial Naive Bayesian

=== Task ===

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

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

In [2]:
```
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups()

#Check target names
data.target_names
```

Out[2]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [3]:
```
# Select just a few categories
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)
```

Put in class

In [4]:
```
class MultiNommialClassification():

    def __init__(self):
        self.laplace = 1

    def transform(self, train, test, method):
        if method == 'TfidVectorizer':
            vectorizer = TfidfVectorizer()
        elif method == 'CountVectorizer':
            vectorizer = CountVectorizer
        else:
            raise ValueError("You must use a Vectorizer! method= TfidVectorizer | CountVectorizer")

         # count the number of unique words
        X_train = vectorizer.fit_transform(train)
        X_test = vectorizer.transform(test)
        X_test = X_test.toarray()
        return X_train, X_test

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

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

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

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

    def likelihood(self, X_class):
        return ((X_class.sum(axis=0)) + self.laplace) / (np.sum(X_class.sum(axis=0) + self.laplace))

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

Predict

In [5]:
```
y_train = train.target
y_test = test.target

model = MultiNommialClassification()
X_train, X_test = model.transform(train.data, test.data, method='TfidVectorizer')
model.fit(X_train, y_train)

yhat = model.predict(X_test)
```

Classification Report

In [6]:
```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import average_precision_score, classification_report

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:  0.8016759776536313
=========Average precision score=======
Class 0 score:  0.888341920518241
Class 1 score:  0.8744630809734135
Class 2 score:  0.6122064043881043
Class 3 score:  0.332994836297269
=========Classification report=======
Report:                precision    recall  f1-score   support

           0       0.97      0.88      0.92       389
           1       0.92      0.92      0.92       394
           2       0.62      0.98      0.76       398
           3       1.00      0.19      0.32       251

    accuracy                           0.80      1432
   macro avg       0.88      0.75      0.73      1432
weighted avg       0.86      0.80      0.77      1432
```