Natural Language Processing

Lab #4

Rhichard Koh

Multinomial Naive Bayes from SKlearn

In [1]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load the 20 Newsgroups dataset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups_data = fetch_20newsgroups(subset='all', categories=categories, shuffle=True

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    newsgroups_data.data, newsgroups_data.target, test_size=0.2, random_state=42)

# Create a CountVectorizer to convert text into token counts
vectorizer = CountVectorizer(stop_words='english')

# Fit and transform the training data
X_train_counts = vectorizer.fit_transform(X_train)

# Transform the testing data
X_test_counts = vectorizer.transform(X_test)

# Initialize and train a Naive Bayes classifier (MultinomialNB)
classifier = MultinomialNB()
classifier.fit(X_train_counts, y_train)

# Predict the labels of the test set
y_pred = classifier.predict(X_test_counts)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.976063829787234
```

Multinomial Naive Bayes from scratch

In [28]:
```
import numpy as np

class RK_Naive_Bayes:
```

```
    def __init__(self):
        self.class_log_prior_ = None
        self.feature_log_prob_ = None
        self.classes_ = None

    def fit(self, X, y):
        # Count the number of documents in each class
        unique_y = np.unique(y)
        self.classes_ = unique_y
        class_counts = np.array([len(y[y == yi]) for yi in unique_y])
        self.class_log_prior_ = np.log(class_counts / len(y))

        # Count the frequency of each feature in each class
        feature_counts = np.zeros((len(unique_y), X.shape[1]))
        for i, yi in enumerate(unique_y):
            feature_counts[i, :] = X[y == yi].sum(axis=0)
        smoothed_fc = feature_counts + 1  # Laplace smoothing
        smoothed_cc = smoothed_fc.sum(axis=1).reshape(-1, 1)
        self.feature_log_prob_ = np.log(smoothed_fc / smoothed_cc)

    def predict_log_proba(self, X):
        return (X @ self.feature_log_prob_.T) + self.class_log_prior_

    def predict(self, X):
        log_probs = self.predict_log_proba(X)
        return self.classes_[np.argmax(log_probs, axis=1)]

    def score(self, X_test, y_test):
        y_pred = self.predict(X_test)
        return np.sum(y_pred == y_test)/len(y_test)
```

In [29]:
```
model = RK_Naive_Bayes()
model.fit(X_train_counts, y_train)

accuracy = model.score(X_test_counts, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.976063829787234
```

Logistic Regression from SKlearn using Count Vectorizer

In [30]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the 20 Newsgroups dataset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups_data = fetch_20newsgroups(subset='all', categories=categories, shuffle=True

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    newsgroups_data.data, newsgroups_data.target, test_size=0.2, random_state=42)

# Create a CountVectorizer to convert text into token counts
vectorizer = CountVectorizer(stop_words='english')
```

```
# Fit and transform the training data
X_train_counts = vectorizer.fit_transform(X_train)

# Transform the testing data
X_test_counts = vectorizer.transform(X_test)

# Initialize and train a classifier (e.g., Logistic Regression)
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X_train_counts, y_train)

# Predict the labels of the test set
y_pred = classifier.predict(X_test_counts)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9694148936170213
```

Logistic Regression from SKlearn using Tfidf Vectorizer

In [31]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# Initialize and train a classifier (e.g., Logistic Regression)
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X_train_counts, y_train)

# Predict the labels of the test set
y_pred = classifier.predict(X_test_counts)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.964095744680851
```