Programming Assignment - 3

ADA University | Ahmadbey Aghaoghlu str. 61, Baku, Azerbaijan, AZ1008

Fall 2023 Artificial Intelligence (CSCI-3613 - 10349)

Gabil Gurbanov

2023

Naive Bayes Algorithm for Sentiment Analysis

Gabil Gurbanov (ADA University)

ID: 12098 ([ggurbanov12098@ada.edu.az](mailto:ggurbanov12098@ada.edu.az))

Fall 2023 Artificial Intelligence (CSCI-3613 - 10349)

Naive Bayes Algorithm (NBA) is a classification algorithm based on Bayes' theorem with the assumption of feature independence. In other words, NBA assumes that the presence of any attribute in a class is not associated with the presence of any other attribute.

Pros and cons of Naive Bayes Algorithm (NBA)

Positive sides:

1. Classification, including multi-class classification, is easy and fast.
2. When the independence assumption is met, NBA outperforms other algorithms such as logistic regression while requiring less training data.

Negative sides:

1. If there is some categorical feature value in the test dataset that was not found in the training dataset, then the model will assign a probability of zero to that value and will not be able to make a prediction. This phenomenon is known as “zero frequency”. This problem can be solved using smoothing. One of the simplest methods is Laplace smoothing.
2. Another limitation of NBA is the assumption of independence of features. In reality, sets of completely independent features are extremely rare.

**Part 1: Importing the Dataset to DataFrame**

* Imported the necessary libraries (pandas and os).
* Defined a function (readFile) to read the content of a file.
* Defined a function (createDataframe) to create a DataFrame from a folder of text files with a specified label.
* Obtained the file paths for positive and negative reviews.
* Created DataFrames for positive and negative reviews (pos\_df and neg\_df).
* Combined positive and negative DataFrames into one (combined\_df).
* Displayed the combined DataFrame.

import pandas as pd

import os

# Define a function to read the content of a file

def readFile(filePath):

    with open(filePath, 'r', encoding='utf-8') as file:

        return file.read()

# Define a function to create a DataFrame from a folder of text files with a specified label

def createDataframe(folderPath, label):

    files = os.listdir(folderPath)                  # List all files in the given folder

    data = {'text': [], 'label': []}                # Initialize a dictionary to store data

    # Loop through each file in the folder

    for file in files:

        filePath = os.path.join(folderPath, file)

        content = readFile(filePath)                # Read the content of the file

        # Append content and label to the data dictionary

        data['text'].append(content)

        data['label'].append(label)

    return pd.DataFrame(data)                       # Create a DataFrame from the data dictionary

# Define folder paths for positive and negative reviews

posFolderPath = os.path.join('./', 'txt\_sentoken', 'pos')

negFolderPath = os.path.join('./', 'txt\_sentoken', 'neg')

# Create DataFrames for positive and negative reviews

pos\_df = createDataframe(posFolderPath, 'pos')

neg\_df = createDataframe(negFolderPath, 'neg')

# Combine positive and negative DataFrames into one

combined\_df = pd.concat([pos\_df, neg\_df], ignore\_index=True)

combined\_df

| **text** | **label** |
| --- | --- |
| 0 | films adapted from comic books have had plenty... | pos |
| 1 | every now and then a movie comes along from a ... | pos |
| 2 | you've got mail works alot better than it dese... | pos |
| 3 | " jaws " is a rare film that grabs your atten... | pos |
| 4 | moviemaking is a lot like being the general ma... | pos |
| ... | ... | ... |
| 1995 | if anything , " stigmata " should be taken as ... | neg |
| 1996 | john boorman's " zardoz " is a goofy cinematic... | neg |
| 1997 | the kids in the hall are an acquired taste . \... | neg |
| 1998 | there was a time when john carpenter was a gre... | neg |
| 1999 | two party guys bob their heads to haddaway's d... | neg |

2000 rows × 2 columns

**Part 2: Tokenization**

* Imported the CountVectorizer from sklearn.feature\_extraction.text.
* Defined a list of stop words (stopWords).
* Defined a function (getMostCommonWords) to get the most common words in a list of texts.
* Obtained the most common words in positive and negative reviews (posWords and negWords).
* Created DataFrames for the most common words in positive and negative reviews (posDataFrame and negDataFrame).
* Printed the most common words in positive and negative reviews.

from sklearn.feature\_extraction.text import CountVectorizer

stopWords = ['would', 'characters', 'like', 'character', 'time', 'story', 'one', 'film', 'movie', 'above', 'here', 'too', 'ours', 'same', 'no', 'some', 'whom', 'over', 'does', 'the', 'through', 'am', 'those', 'at', 'having', 'she', 'again', 'which', 'until', 'while', 'don', 'to', 'this', 'other', 'after', 'my', 'each', 'such', 'them', 'it', 'should', 'her', 'herself', 'how', 'has', 'any', 's', 'then', 'can', 'but', 'there', 'they', 'most', 'our', 'him', 'these', 'himself', 'yourselves', 'itself', 'of', 'are', 'he', 'you', 'only', 'because', 'from', 'had', 'who', 'own', 'if', 'yours', 'during', 'below', 'his', 'nor', 'themselves', 'between', 'more', 'when', 'where', 'being', 'few', 'with', 'be', 'hers', 'for', 'out', 'doing', 'into', 'ourselves', 'down', 'do', 'off', 'their', 't', 'all', 'that', 'will', 'about', 'been', 'as', 'before', 'is', 'now', 'we', 'or', 'by', 'than', 'an', 'and', 'once', 'just', 'up', 'against', 'not', 'on', 'its', 'very', 'did', 'yourself', 'both', 'so', 'your', 'were', 'i', 'in', 'a', 'further', 'was', 'me', 'myself', 'what', 'theirs', 'under', 'have', 'why']

# Define a function to get the most common words in a list of texts

def getMostCommonWords(texts, n=10):

    vectorizer = CountVectorizer(stop\_words = stopWords)                                # Create a CountVectorizer with custom stop words

    X = vectorizer.fit\_transform(texts)                                                 # Transform the input texts into a document-term matrix

    words = vectorizer.get\_feature\_names\_out()                                          # Get the feature names (words)

    wordCounts = X.sum(axis=0).A1                                                       # Calculate the sum of word occurrences across all documents

    wordFrequency = dict(zip(words, wordCounts))                                        # Create a dictionary of word frequencies

    sortedWordFreq = sorted(wordFrequency.items(), key=lambda x: x[1], reverse=True)    # Sort the word frequencies in descending order

    return sortedWordFreq[:n]                                                           # Return the top 'n' most common words

# Get the most common words in positive and negative reviews

posWords = getMostCommonWords(pos\_df['text'])

negWords = getMostCommonWords(neg\_df['text'])

# Create DataFrames for the most common words in positive and negative reviews

posDataFrame = pd.DataFrame(posWords, columns=['Word', 'Frequency'])

negDataFrame = pd.DataFrame(negWords, columns=['Word', 'Frequency'])

# Print the most common words in positive and negative reviews

print("Most common words in positive reviews:")

print(posDataFrame)

print("\nMost common words in negative reviews:")

print(negDataFrame)

Most common words in positive reviews:

Word Frequency

0 good 1248

1 also 1200

2 even 1179

3 well 1123

4 life 1057

5 much 1038

6 first 1004

7 two 999

8 see 965

9 way 929

Most common words in negative reviews:

Word Frequency

0 even 1386

1 good 1163

2 get 1052

3 bad 1034

4 much 1011

5 plot 917

6 two 912

7 make 851

8 first 832

9 could 791

**Subpart 2.5: Visualization of most common words in each class**

import matplotlib.pyplot as plt

def plotMostCommonWords(data\_frame, title):

    plt.figure(figsize=(12, 5))

    plt.bar(data\_frame['Word'], data\_frame['Frequency'], color='blue')

    plt.xlabel('Words')

    plt.ylabel('Frequency')

    plt.title(title)

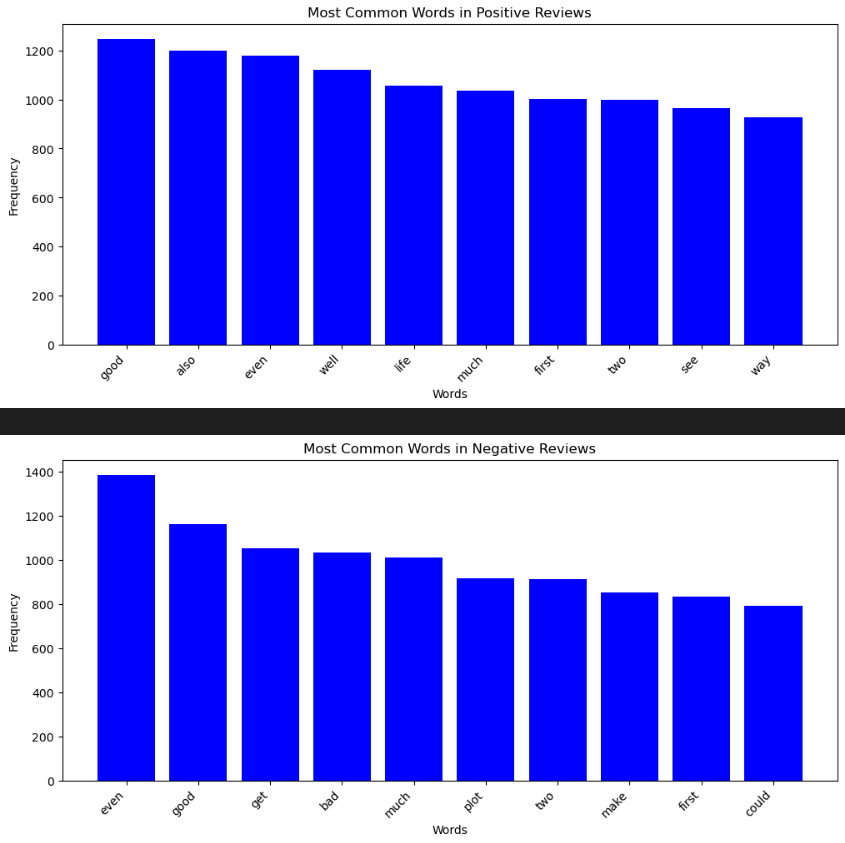
    plt.xticks(rotation=45, ha='right')

    plt.show()

# Visualize most common words in positive reviews

plotMostCommonWords(posDataFrame, 'Most Common Words in Positive Reviews')

plotMostCommonWords(negDataFrame, 'Most Common Words in Negative Reviews')



**Part 3: Naive Bayes Algorithm for Sentiment Analysis without Library**

* Imported numpy for numerical operations.
* Defined a class (NaiveBayesClassifier) for a Naive Bayes classifier.
* Implemented methods in the class for calculating class probabilities, word probabilities, training, prediction, and evaluation.
* Defined a function (trainTestSplit) for splitting data into training and testing sets.
* Prepared data for training and testing.
* Trained the Naive Bayes classifier using training data.
* Performed predictions on sample test documents.
* Evaluated the classifier's performance using the test data.
* Printed the predictions and evaluation metrics.

import numpy as np

from numpy.random import shuffle, seed

class NaiveBayesClassifier:

    # Initialize class probabilities and word probabilities dictionaries

    def \_\_init\_\_(self):

        self.classProbs = {}

        self.wordProbs = {}

    # Calculate class probabilities based on the input labels

    def calcClassProb(self, labels):

        uniqueLabels, counts = np.unique(labels, return\_counts=True)

        totalSamples = len(labels)

        for label, count in zip(uniqueLabels, counts):

            self.classProbs[label] = count / totalSamples

    # Calculate word probabilities for each class based on the input documents and labels

    def calcWordProb(self, documents, labels):

        combinedData = list(zip(documents, labels))

        for label in set(labels):

            labelData = [data[0] for data in combinedData if data[1] == label]

            labelWordCounts = {}

            totalWords = 0

            for document in labelData:

                words = document.split()

                totalWords += len(words)

                for word in words:

                    if word in labelWordCounts:

                        labelWordCounts[word] += 1

                    else:

                        labelWordCounts[word] = 1

            wordProbs = {word: (count) / totalWords for word, count in labelWordCounts.items()}

            self.wordProbs[label] = wordProbs

    # Train the Naive Bayes Classifier by calculating class and word probabilities

    def train(self, documents, labels):

        self.calcClassProb(labels)

        self.calcWordProb(documents, labels)

    # Predict the class of a given document

    def predict(self, document):

        words = document.split()

        maxProb = float('-infinity')

        predictedClass = None

        # Calculate the probability of each class for the given document

        for label, classProb in self.classProbs.items():

            wordProb = self.wordProbs[label]

            documentProb = np.log(classProb)

            # Calculate the document probability based on word probabilities

            for word in words:

                if word in wordProb:

                    documentProb += np.log(wordProb[word])

            # Update the predicted class if the current probability is higher

            if documentProb > maxProb:

                maxProb = documentProb

                predictedClass = label

        return predictedClass

    # Evaluate the classifier's performance using test documents and labels

    def evaluate(self, testDocuments, testLabels):

        predictions = [self.predict(document) for document in testDocuments]

        truePositive, falsePositive, trueNegative, falseNegative = 0, 0, 0, 0

        # Calculate metrics such as accuracy, precision, recall, and F1 score

        for predictedLabel, trueLabel in zip(predictions, testLabels):

            if predictedLabel == trueLabel and trueLabel == 'pos':

                truePositive += 1

            elif predictedLabel == trueLabel and trueLabel == 'neg':

                trueNegative += 1

            elif predictedLabel != trueLabel and trueLabel == 'pos':

                falseNegative += 1

            elif predictedLabel != trueLabel and trueLabel == 'neg':

                falsePositive += 1

        accuracy = (truePositive + trueNegative) / len(testLabels)

        precision = truePositive / (truePositive + falsePositive) if (truePositive + falsePositive) != 0 else 0

        recall = truePositive / (truePositive + falseNegative) if (truePositive + falseNegative) != 0 else 0

        f1 = 2 \* (precision \* recall) / (precision + recall) if (precision + recall) != 0 else 0

        return accuracy, precision, recall, f1

# Define a function for training and test data split

def trainTestSplit(data, labels, testSize, randomState=None):

    if randomState:

        seed(randomState)

    indices = np.arange(len(data))

    shuffle(indices)

    splitIndex = int((1 - testSize) \* len(data))

    trainIndices, testIndices = indices[:splitIndex], indices[splitIndex:]

    trainData, testData = [data[i] for i in trainIndices], [data[i] for i in testIndices]

    trainLabels, testLabels = [labels[i] for i in trainIndices], [labels[i] for i in testIndices]

    return trainData, testData, trainLabels, testLabels

# Prepare data for training and testing

documents = combined\_df['text'].tolist()

labels = combined\_df['label'].tolist()

# Split the data into training and testing sets

trainDocuments, testDocuments, trainLabels, testLabels = trainTestSplit(documents, labels, testSize=0.25, randomState=40)

# Create an instance of the NaiveBayesClassifier class

NaiveBayesClassifierInstance = NaiveBayesClassifier()

# Train the classifier using the training data

NaiveBayesClassifierInstance.train(trainDocuments, trainLabels)

# Perform predictions on sample test documents

sampleTest = ["I didn't like the movie, total trash", "This is really nice movie, I enjoyed it"]

for document in sampleTest:

    prediction = NaiveBayesClassifierInstance.predict(document)

    print(f"Prediction for '{document}':    {prediction}")

# Evaluate the classifier's performance using the test data

accuracy, precision, recall, f1 = NaiveBayesClassifierInstance.evaluate(testDocuments, testLabels)

print("\nEvaluation Metrics:")

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

Prediction for 'I didn't like the movie, total trash': neg

Prediction for 'This is really nice movie, I enjoyed it': pos

Evaluation Metrics:

Accuracy: 0.41

Precision: 0.34355828220858897

Recall: 0.22950819672131148

F1 Score: 0.2751842751842752

**Part 4: Naive Bayes Algorithm for Sentiment Analysis with Library (sklearn)**

* Imported necessary libraries for model evaluation and training (sklearn.metrics, sklearn.model\_selection, sklearn.naive\_bayes).
* Split the data into training and testing sets.
* Created a CountVectorizer with custom stop words.
* Vectorized the training and testing data.
* Initialized a Multinomial Naive Bayes classifier.
* Trained the classifier on the vectorized training data.
* Made predictions on the vectorized testing data.
* Calculated accuracy, confusion matrix, and classification report.
* Printed evaluation metrics.
* Performed custom predictions on sample input and displayed the results.
* from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
* from sklearn.model\_selection import train\_test\_split
* from sklearn.naive\_bayes import MultinomialNB
* # Split the data into training and testing sets
* xTrain, xTest, yTrain, yTest = train\_test\_split(combined\_df['text'], combined\_df['label'], test\_size=0.25, random\_state=40)
* # Create a CountVectorizer with custom stop words
* vectorizer = CountVectorizer(stop\_words=stopWords)
* # Vectorize the training and testing data
* xTrainVectorized = vectorizer.fit\_transform(xTrain)
* xTestVectorized = vectorizer.transform(xTest)
* classifier = MultinomialNB()                        # Initialize a Multinomial Naive Bayes classifier
* classifier.fit(xTrainVectorized, yTrain)            # Train the classifier on the vectorized training data
* yPrediction = classifier.predict(xTestVectorized)   # Make predictions on the vectorized testing data
* # Calculate accuracy, confusion matrix, and classification report
* accuracy = accuracy\_score(yTest, yPrediction)
* confMatrix = confusion\_matrix(yTest, yPrediction)
* classificationReport = classification\_report(yTest, yPrediction)
* # Print evaluation metrics
* print(f"Accuracy: {accuracy}")
* print("\nConfusion Matrix:")
* print(confMatrix)
* print("\nClassification Report:")
* print(classificationReport)
* print("\n")
* # Perform custom predictions on some sample input
* customInput = ["I didn't like the movie, total trash", "This is really nice movie, I enjoyed it"]
* customPredictions = classifier.predict(vectorizer.transform(customInput))
* # Display custom predictions
* for text, prediction in zip(customInput, customPredictions):
* print(f"Text: {text}\nPrediction: {prediction}\n")

Accuracy: 0.814

Confusion Matrix:

[[214 47]

[ 46 193]]

Classification Report:

precision recall f1-score support

neg 0.82 0.82 0.82 261

pos 0.80 0.81 0.81 239

accuracy 0.81 500

macro avg 0.81 0.81 0.81 500

weighted avg 0.81 0.81 0.81 500

Text: I didn't like the movie, total trash

Prediction: neg

Text: This is really nice movie, I enjoyed it

Prediction: pos

Reference:  
<http://datascientist.one/naive-bayes/>

<https://scikit-learn.ru/1-9-naive-bayes/>

<https://www.ibm.com/topics/naive-bayes>