Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Part I: Naïve Bayes for Classification

Objective:

The main goal of this part is to implement the Naïve Bayes algorithm to create a basic classifier.

Background:

In this part of the project, we have implemented a Naïve Bayes algorithm that uses probabilities to perform classification. The probabilities are estimated based on training data which is given as a sample dataset named "fishing.data" for which the value of the classification is known. This dataset consists of the 14 data points (examples). Our target is to find the class of a new instance using the algorithm which has implemented based on the training data. To do this, our algorithm needs to follow the below operations:

Learn (applied to the Training Data)

1. Estimate the probability of each class:
   $P(c_j) = \#c_j \ / \ \#\text{training examples}$
2. Estimate the probability of each attribute value a_i, given a class of type j
   $P(a_i \mid c_j) = \#a_i / \#c_j$

Classify (applied to the new Instance)

1. Return classification C_{NB} for new instance

$$C_{NB} = \max_{c_j \in C} \left(P(c_j) \prod_i P(a_i|c_j) \right)$$

where c_j denotes the class, a_i denotes the value of i^{th} attribute

We have tried to create a generalized function for training with any type of textual dataset like .csv/.tsv and with/without header in file. It will also count first row as name of the attributes if the header is present in the dataset, otherwise it will count first row as instance.

```python
def train(self, class_at_column=1, header=True, filepath='', delimeter=','):
    """
    self.class_col = class_at_column - 1

    with open(filepath, mode='r', encoding='utf8') as fin:
        """
        Take the first line and based on header(True/False)
        Create Prior, Likelihood and Variable name lists
        """

        first_line = fin.readline()
        parts = first_line.strip().split(delimeter)

        if header:
            i = 1
            for col in parts:
                if col in self.attribute_names:
                    raise IOError("Column Names not Unique.")
                self.attribute_names.append(col.strip())
                i += 1
        else:
            self.total_instances += 1

            for i in range(1, len(parts) + 1, 1):
                self.attribute_names.append(f'column_{i}')

            for i in range(0, len(parts), 1):
                if i == self.class_col:
                    self.prior_counts[parts[i]] += 1
                    continue

                self.likelihood_counts[
                    f'{self.attribute_names[i]}|{parts[i]}|{parts[self.class_col]}'] += 1

        """
        Parse the full file and count all data
        """
        for line in fin:
            self.total_instances += 1
            parts = line.strip().split(delimeter)

            for i in range(0, len(parts), 1):
                if i == self.class_col:
                    self.prior_counts[parts[i]] += 1
                    continue

                self.likelihood_counts[
                    f'{self.attribute_names[i]}|{parts[i]}|{parts[self.class_col]}'] += 1

    for key in self.prior_counts:
        self.priors[key] = self.prior_counts[key] / self.total_instances

    for key in self.likelihood_counts:
        cls = key.split('|')[2]
        self.likelihoods[key] = self.likelihood_counts[key] / self.prior_counts[cls]
```

Visualization: [Display the Output]

Output generated using Implemented Algorithm:

```
Enter training dataset name: fishing.data
Enter new Instance, separated by comma: Strong,WarmAir,Cold,Sunny


**********************************

Training Instances: 14

Prior Probabilities
#Yes: 8     P(Yes): 0.5714
#No: 6      P(No): 0.4286

**********************************


New instance: ['Strong', 'WarmAir', 'Cold', 'Sunny']

P(Strong|Yes): 0.75 P(WarmAir|Yes): 0.62P(Cold|Yes): 0.12   P(Sunny|Yes): 0.75
P(Strong|No): 0.33  P(WarmAir|No): 0.33 P(Cold|No): 0.50    P(Sunny|No): 0.33

Class probabilities

Yes: 0.0251
No: 0.0079

Classify: Yes
Conditional Probability for class "Yes" : 75.98%


Continue?(Y/n): Y
Enter new Instance, separated by comma: Weak,WarmAir,Moderate,Rainy


**********************************

Training Instances: 14

Prior Probabilities
#Yes: 8     P(Yes): 0.5714
#No: 6      P(No): 0.4286

**********************************


New instance: ['Weak', 'WarmAir', 'Moderate', 'Rainy']

P(Weak|Yes): 0.25   P(WarmAir|Yes): 0.62P(Moderate|Yes): 0.50   P(Rainy|Yes): 0.12
P(Weak|No): 0.67    P(WarmAir|No): 0.33 P(Moderate|No): 0.33P(Rainy|No): 0.50

Class probabilities

Yes: 0.0056
No: 0.0159

Classify: No
Conditional Probability for class "No" : 73.99%


Continue?(Y/n): n
```

Output generated using Scikit-Learn Gaussian NB:

```
Enter new Instance, separated by comma: Strong,WarmAir,Cold,Sunny
Classify: Yes
Continue?(Y/n): Y
Enter new Instance, separated by comma: Weak,WarmAir,Moderate,Rainy
Classify: No
Continue?(Y/n): n

Process finished with exit code 0
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Analysis and Discussion:

We can see from the above visualization part that the output generated using the implemented algorithm and the output generated using the Naïve-Bayes module in Scikit-learn are same. So, we can say that our implemented algorithm is validated by a library package. To implement the algorithm, we have used the Python dictionary and there were few problems encountered during the whole implementation process like the dataset had no header whereas there might be header in average number of datasets. In that case we had to consider this to create the generalized function which can be used for any kind of dataset. Another problem was that the class was in the first column in our dataset but in general the class can be in the last column of the average number of datasets. To create the generalized function, we had to consider this issue too.

Future Work:

We can see from our sample dataset that it has only 14 instances which is too small. But if we use any larger dataset for this algorithm, we may get many small likelihoods. And if we multiply those likelihoods together as per our classifier model, arithmetic underflow can happen. Because, in that case, the result of the calculation will be a number smaller than absolute value than the computer can represent as a fixed length (fixed precision) binary digit. As a result, instead of returning the actual result of the calculation, the computer returns a zero. So, if we take the log of each likelihood and then add those instead of multiplying those together, it will increase the absolute value which will reduce the risk of arithmetic underflow.

[Note: You will find the source-code in the last part of this report]

Part II: Naïve Bayes for Spam Classification

Objective:

The main goal of this part is to implement the Naïve Bayes algorithm to create a spam classifier (filter).

Background:

In this part of the project, we have used a sample dataset (textMsgs.data) which consists of a collection of 5574 labeled SMS text messages where about 13% of the messages are spam. Basically, we have implemented a Naïve Bayes algorithm that uses probabilities to perform classification. In part 1 of this project, we used multiplication technique to compute the probability of each class, given the probabilities of the observed data. But, as the data set, we have used is larger in this part, we have used the logarithm technique to compute those probabilities to avoid arithmetic underflow. The probabilities are estimated based on training data that consists of labeled (Spam | Ham) text messages. Then the target is to learn how to classify (or tag) new messages correctly using this Naïve Bayes classifier. To do this, we have used 10-fold cross validation where 90% data will be training data and rest 10% will be test data. In addition, the sample data was not pre-processed. So, we have observed the prediction both before and after data pre-processing. Moreover, we have tried to characterize misclassification by using precision and recall analysis.

Data pre-processing:

To pre-process the dataset, we have converted all words to lowercase, removed all numbers, punctuations and all stopwords such as articles, adjectives, and pronouns.

Visualization: [Display the Output]

Output generated before data pre-processing:

Confusion Matrix		Predict	
		Positive (P)	Negative (N)
Actual	Positive (P)	4827	0
	Negative (N)	332	415

Accuracy	Precision	Recall	Misclassification Rate
94.044%	93.56%	100%	5.96%

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
C:\Users\shuvo\AppData\Local\Microsoft\WindowsApps\python3.9.exe "C:\Program Files\JetBrains\PyCharm
2021.3\plugins\python\helpers\pydev\pydevd.py" --multiproc --qt-support=auto --client 127.0.0.1 --port
64509 --file C:/Users/shuvo/Workspace/PyCharm/CIS678/main.py
Connected to pydev debugger (build 213.5744.248)
TP=4827, TN=415, FP=332, FN=0

Process finished with exit code 0

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Output generated after data pre-processing:

Confusion Matrix		Predict	
		Positive (P)	Negative (N)
Actual	Positive (P)	4825	2
	Negative (N)	181	566

Accuracy	Precision	Recall	Misclassification Rate
96.72%	96.38%	99.96%	10.19%

```
Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin
C:\Users\shuvo\AppData\Local\Microsoft\WindowsApps\python3.9.exe C:/Users/shuvo/Workspace/PyCharm/CIS678
/main.py
TP=4825, TN=566, FP=181, FN=2

Process finished with exit code 0

```

Analysis and Discussion:

We can see from the above visualization part that the output generated before data pre-processing and after data pre-processing are not the same. Though the accuracy of our spam classifier is not 100% but it has improved after data pre-processing and same goes for precision too. But, the interesting thing is the recall and misclassification rate have declined after data pre-processing.

Future Work:

There are a few other criteria to pre-process the given dataset. For instance, we can pre-process this dataset by substituting the abbreviations with their full forms, data stemming etc. If we can do so, the accuracy of the implemented classifier will definitely improve.

[Please turn the next page for the source-codes of both parts]

```python
from collections import defaultdict
from nltk import corpus
import math


class NB:
    __class_col = None
    __attribute_names = None
    __prior_counts = None
    __priors = None
    __likelihood_counts = None
    __likelihoods = None
    __total_instances = None

    def __init__(self):
        self.__class_col = None
        self.__attribute_names = []

        self.__prior_counts = defaultdict(int)
        self.__priors = defaultdict(float)

        self.__likelihood_counts = defaultdict(float)
        self.__likelihoods = defaultdict(float)

        self.__total_instances = 0

    def train(self, class_at_column=1, header=True, filepath='', delimeter=','):
        """
        Trying to create a Generalized function for training with any
        type of textual dataset. .csv/.tsv and with/without header in
        file support.

        :param class_at_column: The column that contains the class.
        All other columns are considered attributes.
        Default class at 1st column.
        :param header: If the attribute names present at the first row of the file.
        Default is True.
        :param filepath: The full/relative to .py file location of data set path.
        :param delimeter: The attribute separator for each line.
        :return: None
        """
        self.__class_col = class_at_column - 1

        with open(filepath, mode='r', encoding='utf8') as fin:
            """
            Take the first line and based on header(True/False)
            Create Prior, Likelihood and Variable name lists
            """

            lines = fin.readlines()
            parts = lines[0].strip().split(delimeter)

            if header:
                i = 1
                for col in parts:
                    if col in self.__attribute_names:
                        raise IOError("Column Names not Unique.")
                    self.__attribute_names.append(col.strip())
                    i += 1

            """
            Parse the full file and count all data
            """
            for line in lines:
                self.__total_instances += 1
                parts = line.strip().split(delimeter)

                cls = parts[self.__class_col].strip()

                for i in range(0, len(parts), 1):
```

```
                    if i == self.__class_col:
                        self.__prior_counts[cls] += 1
                        continue

                    self.__likelihood_counts[f'{parts[i].strip()}|{cls}'] += 1

        for cls in self.__prior_counts:
            self.__priors[cls] = self.__prior_counts[cls] / self.__total_instances

        for key in self.__likelihood_counts:
            cls = key.split('|')[-1]
            self.__likelihoods[key] = self.__likelihood_counts[key] / self.__prior_counts[cls]

        """
        For tabular data, if any attribute is previously unseen, we assign probability to 0.
        For textual data, if any word is previously unseen, we assign a
        probability using - 1/(total #word + size of vocab)

        This helps keep the test code same for both tabular and textual data.
        """
        self.__likelihoods['_any_|_any_'] = 0

    def test(self, new_instance=None, no_print=False):
        if new_instance is None:
            raise AttributeError(f"Invalid New Instance.")

        if not no_print: print('\n\n**********************************\n')
        if not no_print: print(f'Training Instances: {self.__total_instances}\n')
        if not no_print: print('Prior Probabilities')

        for cls in self.__priors:
            if not no_print: print(f'#{cls}: {self.__prior_counts[cls]}\t\t', end='')
            if not no_print: print(f'P({cls}): {self.__priors[cls]}\t\t', end='')
            if not no_print: print(f'log(P({cls})): {math.log(self.__priors[cls])}')

        if not no_print: print('\n**********************************\n\n')
        if not no_print: print(f'New instance: {str(new_instance)}\n')

        posteriors = {}

        for cls in self.__priors:
            posteriors[cls] = self.__priors[cls]
            posteriors[f'log({cls})'] = math.log(self.__priors[cls])

            for attr in new_instance:
                key = f'{attr}|{cls}'

                if key in self.__likelihoods:
                    lkh = self.__likelihoods[key]
                else:
                    lkh = self.__likelihoods['_any_|_any_']

                posteriors[cls] *= lkh
                posteriors[f'log({cls})'] += math.log(lkh)
                if not no_print: print(f'P({attr}|{cls}): {lkh}\t', end='')
                if not no_print: print(f'log(P({attr}|{cls})): {math.log(lkh)}\t\t', end='')

            if not no_print: print('')

        if not no_print: print('\nClass probabilities\n')
        max_prob = -999999999
        verdict = ''
        total = 0
        for cls in posteriors:
            if not cls.startswith('log'):
                total += posteriors[cls]
                if not no_print: print(f'{cls}: {posteriors[cls]}\t\t', end='')
                continue

            if not no_print: print(f'{cls}: {posteriors[cls]}')
```

```

            if max_prob < posteriors[cls]:
                max_prob = posteriors[cls]
                verdict = cls

        if not no_print: print(f'\nClassify: {verdict[4:-1]}')

        if total > 0:
            if not no_print: print(f'Conditional Probability for class "{verdict[4:-1]}" : {posteriors[verdict[4:-1]] * 100 / total:.2f}%\n\n')

        return verdict[4:-1]

    def __fix_likelihoods_for_text(self):
        size_of_vocab = len(self.__likelihood_counts)
        total_word_count = 0
        for value in self.__likelihood_counts.values():
            total_word_count += value

        for key in self.__likelihoods:
            self.__likelihoods[key] = (self.__likelihood_counts[key] + 1) / (total_word_count + size_of_vocab)

        self.__likelihoods['_any_|_any_'] = 1 / (total_word_count + size_of_vocab)

    def train_for_text(self, datafile):
        lines = []
        max_len = 0
        # total_instances_by_cls = defaultdict(int)
        class_at_column = 1
        stop_words = set(corpus.stopwords.words('english'))

        with open(datafile, mode='r', encoding='utf8') as fin:
            for line in fin:
                # case-insensitive prediction
                line = line.strip().lower()

                # removing stopwords
                # words = word_tokenize(line)
                # valid_words = []
                # for w in words:
                #     if w not in stop_words:
                #         valid_words.append(w)
                #
                # if max_len < len(valid_words):
                #     max_len = len(valid_words)

                line = line.replace(',', ' ')
                valid_words = line.split()

                # total_instances_by_cls[valid_words[class_at_column - 1]] += 1

                lines.append(valid_words)

        newfile = 'processed_texts.csv'
        with open(newfile, mode='w', encoding='utf8') as fout:

            # for cls in total_instances_by_cls:
            #     fout.write(f'#{cls}_{total_instances_by_cls[cls]}\n')

            for line in lines:
                length = len(line)
                for w in line:
                    fout.write(f'{w},')

                for i in range(0, max_len - length, 1):
                    fout.write(',')

                fout.write('\n')

```

```
        self.train(class_at_column=class_at_column, header=False, filepath=newfile, delimeter=',')
        self.__fix_likelihoods_for_text()

```

```python
from sklearn.naive_bayes import GaussianNB
import numpy as nb
from nltk import word_tokenize
from nltk import corpus
from project2.naive_bayes import NB
from collections import defaultdict

def sampling(filename):
    """
    10-fold cross-validation files. each training set will have 90% of each
    class.
    :param filename: dictionary of lists containing filenames for each class
    :return:
    """
    hams = []
    spams = []
    with open(filename, mode='r', encoding='utf8') as fin:
        for line in fin:
            if line.startswith('ham'):
                hams.append(line.strip())
            else:
                spams.append(line.strip())

    filenames = {
        'ham': [],
        'spam': []
    }

    len_train_ham = int(len(hams) * 0.9)
    len_test_ham = len(hams) - len_train_ham
    index = 0
    while index < len(hams):
        file_train = f'ham_train_{index}.txt'
        file_test = f'ham_test_{index}.txt'
        filenames['ham'].append((file_train, file_test))

        upto = index + len_test_ham
        if upto > len(hams):
            upto = len(hams)

        with open(file_train, mode='w', encoding='utf8') as fout_train, open(
                file_test, mode='w', encoding='utf8') as fout_test:
            for i in range(0, index, 1):
                fout_train.write(f'{hams[i]}\n')

            for i in range(index, upto, 1):
                fout_test.write(f'{hams[i]}\n')

            for i in range(upto, len(hams), 1):
                fout_train.write(f'{hams[i]}\n')

        index += len_test_ham

    len_train_spam = int(len(spams) * 0.9)
    len_test_spam = len(spams) - len_train_spam
    index = 0
    while index < len(spams):
        file_train = f'spam_train_{index}.txt'
        file_test = f'spam_test_{index}.txt'
        filenames['spam'].append((file_train, file_test))

        upto = index + len_test_spam
        if upto > len(spams):
            upto = len(spams)

        with open(file_train, mode='w', encoding='utf8') as fout_train, open(
                file_test, mode='w', encoding='utf8') as fout_test:
            for i in range(0, index, 1):
                fout_train.write(f'{spams[i]}\n')

```

```python
            for i in range(index, upto, 1):
                fout_test.write(f'{spams[i]}\n')

            for i in range(upto, len(spams), 1):
                fout_train.write(f'{spams[i]}\n')

        index += len_test_spam

    return filenames


def textual_data(filename):
    train_test_files = sampling(filename)

    TP = TN = FP = FN = 0

    for i in range(0, len(train_test_files['ham']), 1):
        train_ham, test_ham = train_test_files['ham'][i]
        train_spam, test_spam = train_test_files['spam'][i]
        train_filename = 'training_set.txt'
        test_filename = 'test_set.txt'
        with open(train_filename, mode='w', encoding='utf8') as fout_train, open(
                test_filename, mode='w', encoding='utf8') as fout_test, open(
                train_ham, mode='r', encoding='utf8') as fin_train_ham, open(
                train_spam, mode='r', encoding='utf8') as fin_train_spam, open(
                test_ham, mode='r', encoding='utf8') as fin_test_ham, open(
                test_spam, mode='r', encoding='utf8') as fin_test_spam:

            trains = fin_train_ham.readlines()
            trains.extend(fin_train_spam.readlines())
            test = fin_test_ham.readlines()
            test.extend(fin_test_spam.readlines())

            for line in trains:
                fout_train.write(f'{line.strip()}\n')

            for line in test:
                fout_test.write(f'{line.strip()}\n')

        nb = NB()
        nb.train_for_text(datafile=train_filename)

        with open(test_filename, mode='r', encoding='utf8') as fin:
            for line in fin:
                new_instance = line.strip()

                if len(new_instance) == 0:
                    continue

                # lowercase trained model, so lowercase test
                new_instance = new_instance.lower()

                words = new_instance.split()
                # do pre-processing

                actual_cls = words[0]
                prediction_cls = nb.test(words[1:], no_print=True)

                if actual_cls == 'ham' and prediction_cls == 'ham' :
                    TP += 1
                elif actual_cls == 'spam' and prediction_cls == 'spam':
                    TN += 1
                elif actual_cls == 'ham' and prediction_cls == 'spam':
                    FN += 1
                elif actual_cls == 'spam' and prediction_cls == 'ham':
                    FP += 1

    print(f'TP={TP}, TN={TN}, FP={FP}, FN={FN}')
```

```python

def tabular_data():
    filename = input("Enter training dataset name: ")
    nb = NB()
    nb.train(class_at_column=1, header=False, filepath=filename, delimeter=' ')

    while True:
        new_instance = input("Enter new Instance, separated by comma: ")
        nb.test(new_instance.split(','))

        s = input("Continue?(Y/n): ")
        if s == 'n':
            break


def gaussian_nb():
    X = []
    Y = []
    classes = {
        'Yes': 1,
        'No': 0
    }
    wind = {
        'Strong': 0,
        'Weak': 1
    }
    air = {
        'WarmAir': 0,
        'ColdAir': 1
    }
    water = {
        'Warm': 0,
        'Moderate': 1,
        'Cold': 2
    }
    sky = {
        'Sunny': 0,
        'Cloudy': 1,
        'Rainy': 2
    }
    with open('fishing.data', mode='r') as fin:
        for line in fin:
            parts = line.strip().split(' ')
            Y.append(classes[parts[0]])

            X.append([wind[parts[1]], air[parts[2]], water[parts[3]], sky[parts[4]]])

    classifier = GaussianNB()
    classifier.fit(nb.array(X), nb.array(Y))

    while True:
        new_instance = input("Enter new Instance, separated by comma: ")
        parts = new_instance.split(',')
        new_instance = [[wind[parts[0]], air[parts[1]], water[parts[2]], sky[parts[3]]]]
        c = classifier.predict(new_instance)

        if c == classes['Yes']:
            print(f'Classify: Yes')
        else:
            print(f'Classify: No')

        s = input("Continue?(Y/n): ")
        if s == 'n':
            break

```