Problem #2

2. Please write complete a first draft of a classifier in python and provide an analysis of how well it performs according to the following instructions:
    2.1. You are attempting to classify whether a given chunk of text describes a commercial establishment that offers office space for rent
    2.2. Download the data from this link: https://s3-us-west-1.amazonaws.com/ra-training/ra_data_classifier.csv
    2.3. Use any classification method you want on the data
    2.4. Write a short explanation of your methodology and how your classifier performs
    2.5. Beyond a minimal bar of technical competence, you will be assessed largely on the clarity of your write-up. Do not spend too much time perfecting your classifier

Problem statement :

Binary Classification Problem : To classify whether a given chunk of text describes a commercial establishment that offers office space for rent.

Methodology:

Language used: Python
Library used: pandas, numpy, sklearn, nltk

This problem is a binary classification problem where the model has to predict whether the given chunk of text describes a commercial establishment (that offers space for rent) or not.

The given data are english words. Our model or algorithm understands only real valued numbers so we need to convert these words to the numerical values. As, the data is raw so we need preprocessing of data before feeding it to the algorithm/model.We will be using several **data preprocessing** techniques **(see section 1.1)**.

Once the data is processed, we want to convert these words to real valued vectors so that it can be directly consumed by our algorithm. This process of mapping from textual data to real valued vectors is called feature extraction. There are so many techniques to do feature extraction but we would be using **TF-IDF technique (see section 1.2)** to solve this problem.

At this point, our chunk of text is converted to real valued vectors and is ready to be used in algorithm/model.

Next step is to use **ML algorithms** for training and prediction. We can use Naive Bayes algorithm, support vector machine, random forest or even neural network for the prediction. I am going to use Naive Bayes algorithm **(see section 1.3)** because the problem is about text classification and classical ML approaches like Naive Bayes seem to work better with quicker training time.

Once our model is ready, we need to have some mechanism to measure the **performance**; to see how well the model is doing in terms of accuracy, precision, recall etc. We are going to use cross-validation technique for measuring the performance **(see section 1.4)**.

Section 1.1 - Data Preprocessing

- **Tokenization** - convert sentences to words
- **Removing unnecessary punctuation, tags**
- **Removing stop words** - frequent words such as “the”,”are”, “is”, etc. that don’t have specific semantic
- **Stemming** - words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix
- **Lemmatization** - Another approach to remove inflection by determining the part of speech and utilizing a detailed database of the language.

We can use Natural Language ToolKit (NLTK) to do the text preprocessing operations. NLTK - It is one of the best-known and most-used NLP libraries, useful for all sorts of tasks from tokenization, stemming, tagging, parsing, and beyond.

Listing 1.1.1 - data preprocessing

```
In [ ]: df = pd.read_csv("ra_data_classifier.csv", encoding= 'unicode_escape')                    #1
        df['chunk'] = df['chunk'].apply(lambda x: re.sub(r"[^a-zA-Z0-9%\d\/\$\s]+", "", x))      #2
        df['chunk'].isnull().sum()                                                               #3
        dataX = df['chunk'].values.tolist()                                                      #4

        """
        #1 : Reading the data from a CSV file and converting to pandas dataframe.
        #2 : Removing unnecessary punctuation, tags and converting to words (tokenization)
        #3 : Check for any NULL/NaN values
        #4 : Convert dataframe into a list
        """
```

Section 1. 2 - Feature Extraction

In text processing, words of the text represent discrete, categorical features. How do we encode such data in a way which is ready to be used by the algorithms? The mapping from textual data to real valued vectors is called feature extraction. One of the simplest techniques to numerically represent text is Bag of Words (BOW).

Bag of Words (BOW): We make the list of unique words in the text corpus called vocabulary. Then we can represent each sentence or document as a vector with each word represented as 1 for present and 0 for absent from the vocabulary.
Another representation can be count the number of times each word appears in a document. The most popular approach is using the Term Frequency-Inverse Document Frequency (TF-IDF) technique.
Term Frequency (TF) = (Number of times term t appears in a document)/(Number of terms in the document)

Inverse Document Frequency (IDF) = log(N/n),
where, N is the number of documents and n is the number of documents a term t has appeared in.

The IDF of a rare word is high, whereas the IDF of a frequent word is likely to be low. Thus having the effect of highlighting words that are distinct.

We calculate TF-IDF value of a term as = TF * IDF

Python scikit-learn library provides efficient tools for text data mining and provides functions to calculate TF-IDF of text vocabulary given a text corpus.

Convert document into word vector and find TF-IDF

- stop words are removed
- lemmanize the words

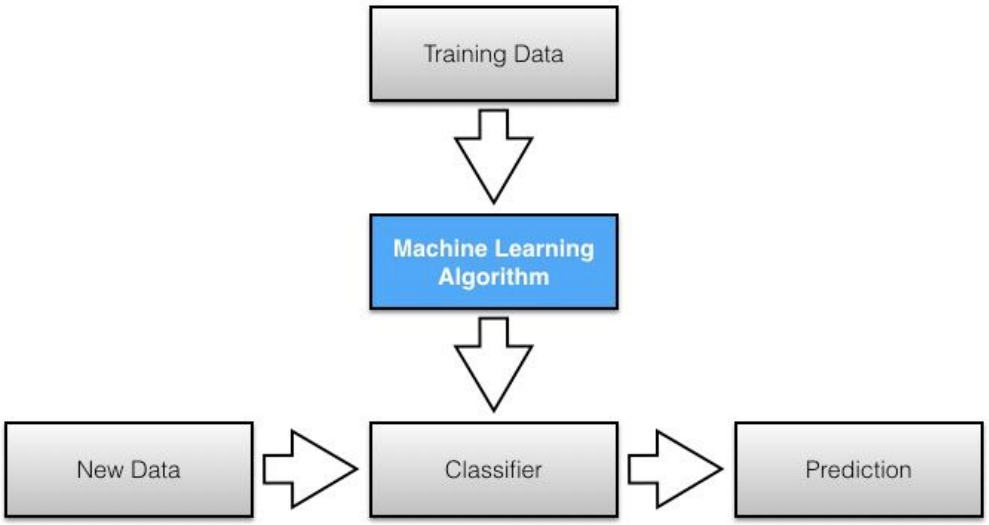
- Get the token
- find TF-IDF

Listing 1.2.1 -TF-IDF using Sklearn ML library

```
In [ ]: from nltk.stem import WordNetLemmatizer                                    #1
        from sklearn.feature_extraction.text import TfidfVectorizer                #1

        class LemmaTokenizer:
            def __init__(self):
                self.wnl = WordNetLemmatizer()                                     #2
            def __call__(self, doc):
                return [self.wnl.lemmatize(t) for t in word_tokenize(doc)]


        vectorizer = TfidfVectorizer(tokenizer=LemmaTokenizer(), stop_words='english')   #3
        X = vectorizer.fit_transform(dataX)                                        #4
        data = X.toarray()                                                         #5

        """
        #1 : import WordNetLemmatizer from nltk and TfidfVectorizer from sklearn library
        #2 : create a class LemmaTokenizer for doing Lemmanization and tokenization.
        #3 : Convert a collection of raw documents to a matrix of TF-IDF features.
        #4 : X is a sparse matrix of (n_samples, n_features)
        #5 : Converting sparse matrix to an array
        """
```

Section 1.3 - ML Algorithms

As mentioned above, we are going to use Naive Bayes algorithms for the prediction. Naive Bayes classifiers, a family of classifiers that are based on the popular Bayes' probability theorem, are known for creating simple yet well performing models, especially in the fields of document classification.



Also, we are going to use a multinomial Naive Bayes classifier. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Listing 1.3.1 showing Multinomial Naive Bayes classifier

```
In [ ]: from sklearn.naive_bayes import MultinomialNB
        naive_bayes = MultinomialNB()
```

Section 1.4 - Measuring Performance

We are using cross-validation techniques for measuring the performance.

As the data set given has very few documents (101 documents in the given csv file) so we can't simply divide the dataset into train and validation and measure the performance. It will lead to overfitting or memorization rather than generalization. We are using a cross-validation method which is a better technique to measure the performance; we get accuracy, precision, recall and confusion matrix to analyze the prediction.

Listing 1.4.1 showing cross-validation technique to measure the performance.

```
In [ ]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
        from sklearn.metrics import confusion_matrix

        naive_bayes = MultinomialNB()
        scoring = ['precision', 'recall', 'accuracy']
        score = cross_validate(naive_bayes, data, labels, cv=5, scoring=scoring)
```

Result

Listing showing the final result using Multinomial Naive Bayes classifier.

```
In [430]: score

Out[430]: {'fit_time': array([0.00243521, 0.00225306, 0.00193977, 0.00153899, 0.00281906]),
           'score_time': array([0.00439978, 0.002635  , 0.00242996, 0.00434995, 0.00320816]),
           'test_precision': array([1.  , 1.  , 0.75, 1.  , 1.  ]),
           'test_recall': array([0.71428571, 0.57142857, 0.42857143, 0.71428571, 0.57142857]),
           'test_accuracy': array([0.9 , 0.85, 0.75, 0.9 , 0.85])}

In [431]: print("Precision", sum(score['test_precision'])/5))
          print("Recall", sum(score['test_recall'])/5)
          print("Accuracy", sum(score['test_accuracy'])/5)

          Precision 0.95
          Recall 0.6
          Accuracy 0.85
```