**Task 2**

**Problem Description:**

The problem is to classify whether a chunk of text describes a commercial establishment that offers office space for rent.

**Methodology:**

This a text classification problem wherein based on the given text, the classifier should categorize it into a commercial establishment that offers office space for rent or not.

To do this, I have built a Naïve Bayes Classifier. Naïve Bayes is a classification technique based on Bayes’ theorem with an assumption of independence among predictors.

To complete this task there are three steps necessary:

* Dataset preparation
* Feature Engineering
* Model Training

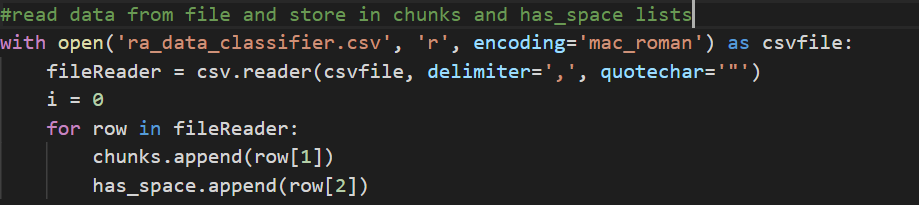
Before I start with these three steps, I need to download the required libraries in python.

For the classifier I have built here, I have used the scikit-learn library available in python. scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

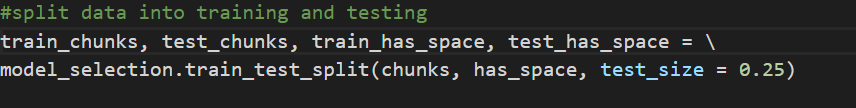


* Dataset Preparation:

As given in the problem statement, I am using the dataset provided, that is available at [https://s3-us-west-1.amazonaws.com/ra-training/ra\_data\_classifier.csv](https://urldefense.com/v3/__https:/s3-us-west-1.amazonaws.com/ra-training/ra_data_classifier.csv__;!!LIr3w8kk_Xxm!5gxOGg5QvO_VbZ3Z27eyuV787mQrvcJXp9I30-5vg_YiwcYLoIGRKuGE7WkJgI4$). This is a csv file with three columns: hid, chunk and has space. The first step in dataset preparation is to load the data from the csv into python. This can be done by using python’s in-built csv module



The next step would be to split the dataset into test data and train data. This can be done by the **train\_test\_split()** available in the model\_selection class in the sklearn module.

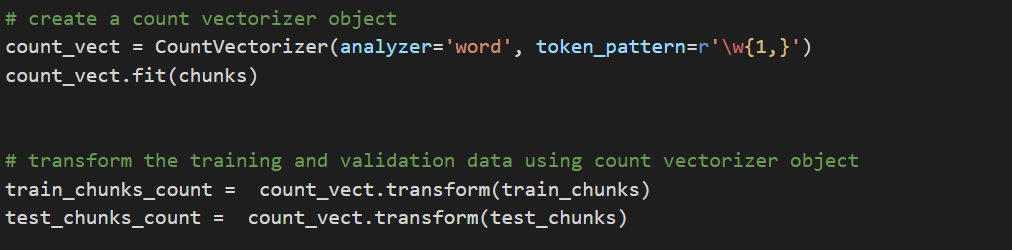


As seen here, the **chunks** and the **has\_space** (which is the label in our case) have been split into train and test sets. The parameter of **test\_size = 0.25** specifies that the test data set should be 1/4th the size of the full data set while the rest of it will be the training data.

* Feature Engineering

The next step is the feature engineering step. Here, I transform the raw text data into feature vectors and create new features using the existing dataset.

To achieve this, I have used a CountVectorizer. Count Vector is a matrix notation of the dataset in which every row represents a text chunk, every column represents a word in the dataset and every cell represents the frequency count of a particular term in a particular chunk of text.

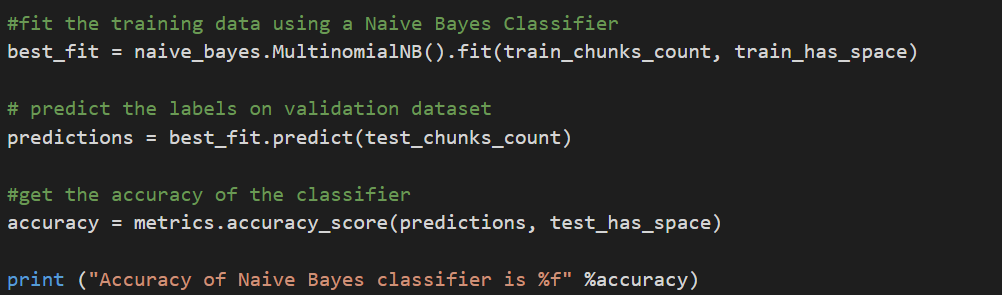


As we can see here, the **fit()** learns the vocabulary dictionary of all the words present in the chunks. The **transform()** then returns a document matrix for all the words in the given dataset.

[Why I chose CountVectorizer?](#CV)

* Model Training

The last step is the model training. In this step we train a classifier using the features we created in the previous step and then run the test data on it to get the accuracy of the model. I have used a Naïve Bayes classifier in this step. I choose a **MulinomialNB()** because the multinomial Naive Bayes classifier is suitable for classification with discrete features. The multinomial distribution normally requires integer feature counts.



The first step is to fit the training data to the MultinomialNB classfier. Here, the **fit()** takes in the training vectors i.e. **train\_chunks** and target values. i.e. **train\_has\_space** values and fits the Naïve Bayes classifier.

The **predict()** then uses the **best\_fit** obtained in the previous step to predict the values for the test chunks.

The values then obtained are cross matched with the test labels i.e. the **test\_has\_space** values to get the accuracy of the classifier.

[Why I chose Naïve Bayes?](#NB)

**Performance:**

The Naïve Bayes classifier I have developed gives an accuracy of **70-80%** on average. This accuracy can be improved by using TF-IDF Vectors, Word Embeddings etc as features or by implementing a Linear Classifier, Support Vector Machine or Deep Neural Networks model.

**Why I chose CountVectorizer?**

The most important problem in text classification is converting the words to numbers, since all classifiers take in numbers as input and train the model based on it. **CountVectorizer**converts the dataset to something called a **Bag-of-Words (BoW)**. It is one of the simplest methods of representing text data for Machine Learning algorithms. It basically puts all the words in the dataset together, and creates a matrix having the count of each word in each chunk of the dataset. Hence I chose CountVectorizer as it one of the simplest methods to assign an integer value to a word in the chunk based on its frequency.

**Why I chose Naïve Bayes?**

Naïve Bayes can provide accurate results without a large dataset. And since a Naïve Bayes classifier is based on Bayes theorem, it helps to calculate the probability of a word belonging to the particular category. Using this probability, its simple to calculate whether a particular chunk of data is more probable to be an ad for office space rent or not.

**Solution file and Execution Steps:**

File: Classifier.py

Execution Steps:

* Open your terminal
* Go the folder which contains Classifier.py
* Run the command: python Classifier.py

**New text test:**

In my code, I have commented an additional part wherein the user can give an input text of his choice, and after the model is trained using the dataset, it predicts whether this new text can be an ad for an office space for rent.

