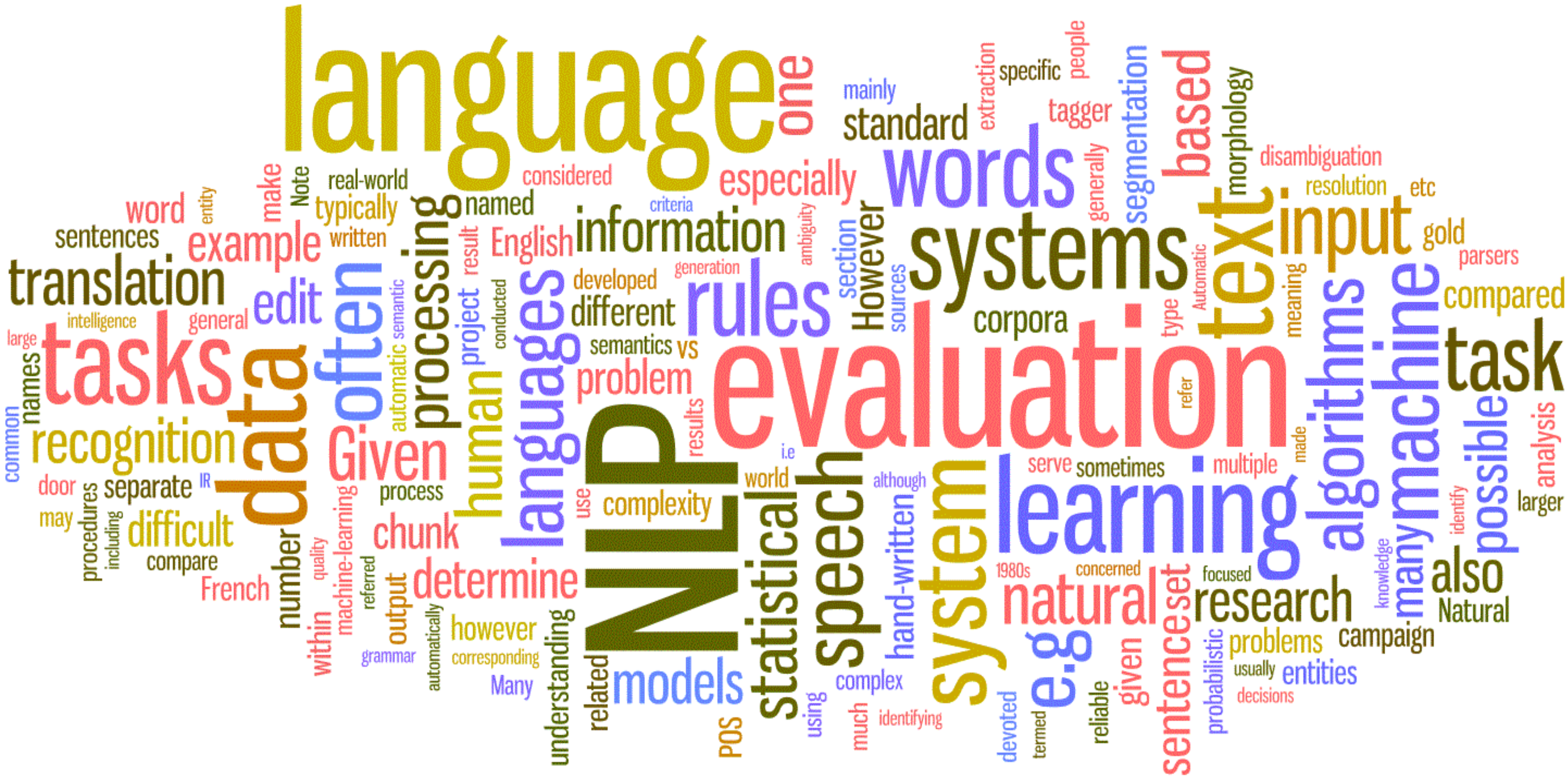


# NATURAL LANGUAGE PROCESSING

# TABLE OF CONTENTS

# 1. INTRODUCTION TO NLP

## Introduction

- Natural language processing (NLP) is an area of computer science and artificial intelligence that is concerned with the interaction between computers and humans in natural language. The goal of NLP is to enable computers to understand language as well as we do.

  It is the driving force behind things like virtual assistants, speech recognition, sentiment analysis, automatic text summarization, machine translation and much more.

- According to industry estimates, only 21% of the available data is present in structured form. Data is being generated as we speak, as we tweet, as we send messages on WhatsApp and in various other activities. Majority of this data exists in the textual form, which is highly unstructured in nature.

  Few notorious examples include – tweets / posts on social media, user to user chat conversations, news, blogs and articles, product or services reviews and patient records in the healthcare sector. A few more recent ones include chatbots and other voice driven bots.

  Despite having high dimension data, the information present in it is not directly accessible unless it is processed (read and understood) manually or analyzed by an automated system.

- The field is divided into the three following parts:

  **Speech Recognition**

  The translation of spoken language into text.

  **Natural Language Understanding:**

  The computer’s ability to understand what we say.

  **Natural Language Generation**

  The generation of natural language by a computer.

## Why NLP is difficult

- Human language is special for several reasons. It is specifically constructed to convey the speaker/writers meaning. It is a complex system, although little children can learn it pretty quickly. Another remarkable thing about human language is that it is all about symbols. According to Chris Manning (Machine Learning Professor at Stanford University), it is a discrete, symbolic, categorical signaling system. This means that you can convey the same meaning by using different ways, like speech, gesture, signs etc. The encoding of these by the human brain is a continuous pattern of activation, where the symbols are transmitted via continuous signals of sound and vision.

- Understanding human language is considered a difficult task due to its complexity. For example, there is an infinite number of different ways to arrange words in a sentence. Also, words can have several meanings and contextual information is necessary to correctly interpret sentences. Every Language is unique and ambiguous. Just look at the following newspaper headline ‘The Pope’s baby steps on gays’. This sentence clearly has two very different interpretations, which is a pretty good example of the challenges in NLP.

- Note that a perfect understanding of language by a computer would result in an AI that can process the whole information that is available on the internet, which in turn would probably result in artificial general intelligence.

## Syntactic & Semantic Analysis

- Syntactic Analysis (Syntax) and Semantic Analysis (Semantic) are the two main techniques that lead to the understanding of natural language. Language is a set of valid sentences, but what makes a sentence valid? You can break validity down into two things: **Syntax** and **Semantics**.
- The term Syntax refers to the grammatical structure of the text whereas the term Semantics refers to the meaning that is conveyed by it. However, a sentence that is syntactically correct, does not have to be semantically correct. Just look at the following example. The sentence "cows flow supremely" is grammatically valid (subject—verb—adverb) but does not make any sense.

## Syntactic Analysis

- Syntactic Analysis, also named Syntax Analysis or Parsing is the process of analyzing natural language conforming to the rules of a formal grammar. Grammatical rules are applied to categories and groups of words, not individual words. Syntactic Analysis basically assigns a semantic structure to text.

  For example, a sentence includes a subject and a predicate where the subject is a noun phrase and the predicate is a verb phrase. Look at the following sentence: "The dog (noun-phrase) went away (verb-phrase)". Note that we can combine every noun phrase with a verb phrase. Like I already mentioned, sentences that are formed like that doesn't really have to make sense although they are syntactically correct.

## Semantic Analysis

For us as humans, the way we understand what someone has said is an unconscious process that relies on our intuition and our knowledge about language itself. Therefore, the way we understand language is heavily based on meaning and context. Since computers can not rely on these techniques, they need a different approach. The word "Semantic" is a linguistic term and means something related to meaning or logic.

Therefore, Semantic Analysis is the process of understanding the meaning and interpretation of words, signs, and sentence structure. This enables computers partly to understand natural language the way humans do, involving meaning and context. I say partly because Semantic Analysis is one of the toughest parts of NLP and not fully solved yet. For example, Speech Recognition has become very good and works almost flawlessly but we are still lacking this kind of proficiency in Natural Language Understanding (e.g. Semantic). Your phone basically understands what you have said but often can't do anything with it because it doesn't understand the meaning behind it. Also, note that some of the technologies out there only make you think they understand the meaning of a text. An approach based on keywords or statistics or even pure machine learning may be using a matching or frequency technique for clues as to what a text is "about." These methods are limited because they are not looking at the real underlying meaning.

# 2. PROBLEM STATEMENT

## Sim Eval 2019 Task 6

- One of the widely used natural language processing task in different business problems is “Text Classification”. The goal of text classification is to automatically classify the text documents into one or more defined categories.
  Some examples of text classification are:

  - Understanding audience sentiment from social media,
  - Detection of spam and non-spam emails
  - Auto tagging of customer queries
  - Categorization of news articles into defined topics.

## Sub-tasks

- A -> Offensive Language Identification **Chosen to Implement**
- B -> Automatic categorization of offensive types
- C -> Offense target identification

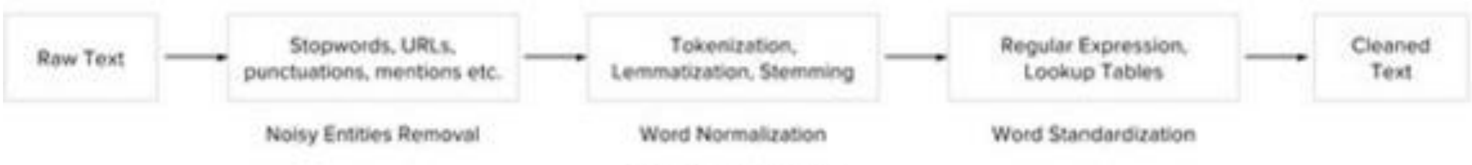
## Motivation

- Offensive language is pervasive in social media. Individuals frequently take advantage of the perceived anonymity of computer-mediated communication, using this to engage in behavior that many of them would not consider in real life. Online communities, social media platforms, and technology companies have been investing heavily in ways to cope with offensive language to prevent abusive behavior in social media.

- One of the most effective strategies for tackling this problem is to use computational methods to identify offense, aggression, and hate speech in user-generated content (e.g. posts, comments, microblogs, etc.).

# 3. TEXT PREPROCESSING

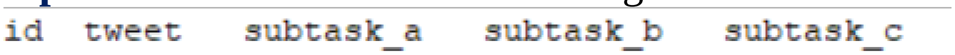
Since, text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing.
The following image shows the architecture of text preprocessing pipeline.


Raw Text
Stopwords, URLs, punctuations, mentions etc.
Tokenization, Lemmatization, Stemming
Regular Expression, Lookup Tables
Cleaned Text
Noisy Entities Removal
Word Normalization
Word Standardization

Text Cleaning Pipeline

## Reading Dataset

- The data is retrieved from social media and distributed in **tab separated format** with the following format:



```
id   tweet   subtask_a   subtask_b   subtask_c
```

- We downloaded the training dataset from the official website of the competition with name **offenseval-training-v1**
- We needed to format our **txt** dataset file, as txt files do not provide any kind of formatting files. There exist two kinds of formats:
  - **tsv file** : **T**ab **S**eparated **V**alues. This kind of format has tap as a delimiter between values.
  - **csv file** : **C**omma **S**eparated **V**alues. This kind of format has the comma as a separator between values.

  We've chosen to format our dataset file with **tsv** format for two main reasons:
    - The downloaded dataset originally has tab separator between tweets and labels of each subtask.
    - The data in each tweet could have commas, so it is not convenient to have a csv file with this dataset.

After successfully reading the Dataset we begin some preprocessing step to clean our dataset data…

## Cleaning

- For each tweet line in the dataset, we manage to remove the words **@user - URL** -if exists- as they are not useful in our classification. To do this we defined a list holding these words (can be expanded to exclude any additional unnecessary words) and iterating through our file using the code below:
- Next, we re-iterate through the fie again to do the following:
  - Removing from each line any symbols or emotions by using a **Regex expression** to keep only alphabetic characters.
  - Transforming all lines into lower case letter to avoid double meaning for same word (i.e. LOVE - love) .

## Noise Removal

- Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.
  For example – language stop-words (commonly used words of a language – is, am, the, of, in, etc.), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.
- A general approach for noise removal is to prepare a dictionary of noisy entities and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.
- Instead of preparing a dictionary of noisy entities, we can use a noise list imported from **nltk module.**
- In order to exclude noisy words from each tweet line, each line must first be **tokenized.**

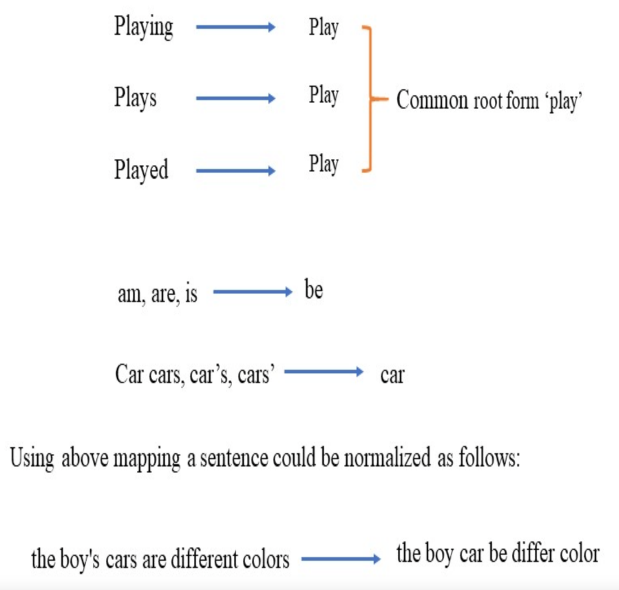
### Tokenization

It’s the operation of splitting up a larger body of text into smaller lines, words or even creating words for a non-English language. So, it is the process of converting a line of text into **tokens** (words or entities present in the text).
The various tokenization functions in-built into the **nltk module** itself and can be used in programs.

## Lexicon Normalization

- Another type of textual noise is about the multiple representations exhibited by single word.
  For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma).
- The most common lexicon normalization practices are :
  - **Stemming**

    Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc.) from a word.
    There are different algorithms that can be used in the stemming process, but the most common in English is Porter stemmer. The rules contained in this algorithm are divided in five different phases numbered from 1 to 5. The purpose of these rules is to reduce the words to the root.


Playing → Play
Plays → Play
Played → Play
Common root form ‘play’
am, are, is → be
Car cars, car’s, cars’ → car
Using above mapping a sentence could be normalized as follows:
the boy's cars are different colors → the boy car be differ color

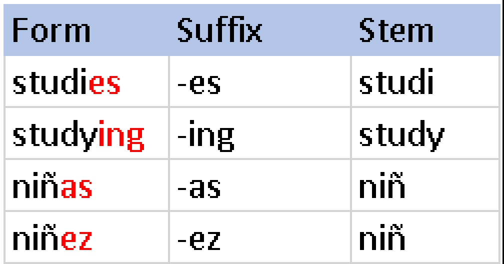
### Lemmatization

Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

The key to this methodology is linguistics. To extract the proper lemma, it is necessary to look at the morphological analysis of each word. This requires having dictionaries for every language to provide that kind of analysis.

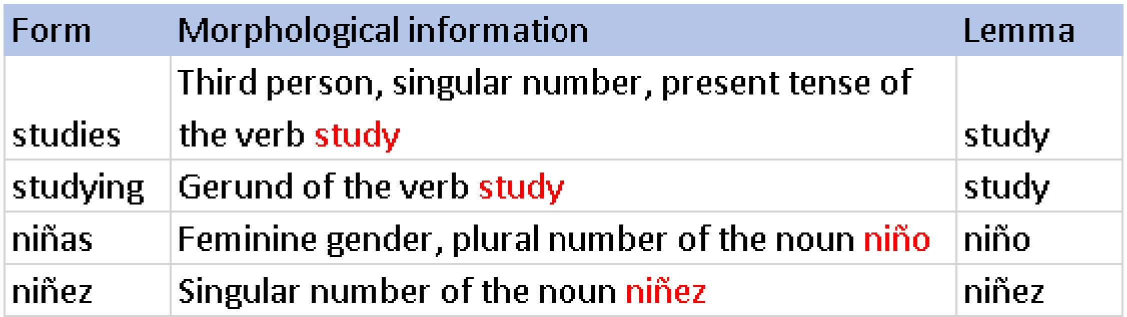
### Main differences between Stemming and Lemmatization

- **Stemming** algorithms work by cutting off the end or the beginning of the word, considering a list of common prefixes and suffixes that can be found in an inflected word. This indiscriminate cutting can be successful in some occasions, but not always, and that is why we affirm that this approach presents some limitations. Below we illustrate the method with examples in both English and Spanish.



| Form | Suffix | Stem |
|---|---|---|
| studies | -es | studi |
| studying | -ing | study |
| niñas | -as | niñ |
| niñez | -ez | niñ |

- **Lemmatization**, on the other hand, takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. Again, you can see how it works with the same example words.



| Form | Morphological information | Lemma |
|---|---|---|
| studies | Third person, singular number, present tense of the verb study | study |
| studying | Gerund of the verb study | study |
| niñas | Feminine gender, plural number of the noun niño | niño |
| niñez | Singular number of the noun niñez | niñez |

- Another important difference to highlight is that a lemma is the base form of all its inflectional forms, whereas a stem isn’t. Therefore, regular dictionaries are lists of lemmas, not stems.
- This has two consequences:
    - First, **the stem can be the same for the inflectional forms of different lemmas**. This translates into noise in our search results. In fact, it is very common to find entire forms as instances of several lemmas.
      For Example, in Telugu, the form for “robe” is identic to the form for “I don’t share”, so their stems are indistinguishable too. But they, of course, belong to different lemmas. The same happens in Gujarati, where the forms and stems for “beat” and “set up” coincide, but we can separate one from another by looking at their lemmas.

    - Also, **the same lemma can correspond to forms with different stems**, and we need to treat them as the same word. For example, in Greek, a typical verb has different stems for perfective forms and for imperfective ones. If we were using stemming algorithms, we won't be able to relate them with the same verb but using lemmatization it is possible to do so. We can clearly observe it in the example below:

So, we can say developing a stemmer is far simpler than building a lemmatizer. In the latter, deep linguistics knowledge is required to create the dictionaries that allow the algorithm to look for the proper form of the word. Once this is done, the noise will be reduced, and the results provided on the information retrieval process will be more accurate. Lemmatizing is more powerful because it gives a meaningful word. That’s why we’ve chosen it as our implementation in noise removal.

# 4. FEATURE ENGINEERING ON TEXT DATA

## Motivation

Feature Engineering is often known as the secret sauce to creating superior and better performing machine learning models. The importance of feature engineering is even more important for unstructured, textual data because we need to convert free flowing text into some numeric representations which can then be understood by machine learning algorithms. Even with the advent of automated feature engineering capabilities, you would still need to understand the core concepts behind different feature engineering strategies before applying them as black box models.

We'll use **Bag of Words Model** to extract features from text.

## Bag of words Model

### Model Definition

- Bag of words model is one of a series of techniques from NLP to extract features from text. The way it does this is by counting the frequency of words in a document.
- A document can be defined as you need, it can be a single sentence or all Wikipedia.

  The output of the bag of words model is a **frequency vector**.
- A bag-of-words involves two things:
  - I. Vocabulary of known words.
  - II. A measure of the presence of known words.
- It is called a ***bag*** of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.
- The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document.

- The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

## Model Building Steps

Here are the steps required to build a Bag of Words Model

**STEP A – Collect Data**

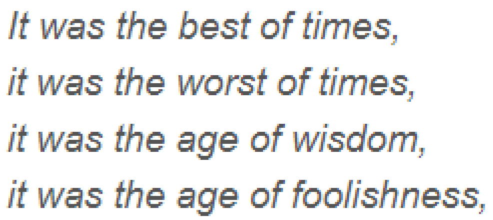
Our data to be used is the Dataset of user tweets after being passed to the Preprocessing Step described above.

**STEP B – Design Vocabulary**

In this step, we make a list of all unique words in our model vocabulary given our collected clean data

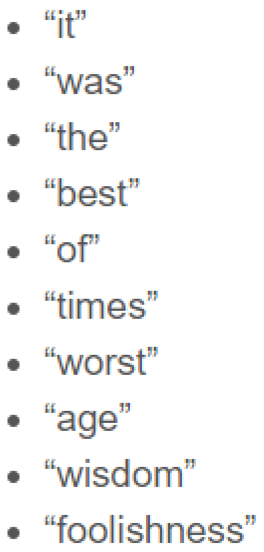
Small Example:

**Collected Data**



*It was the best of times,*
*it was the worst of times,*
*it was the age of wisdom,*
*it was the age of foolishness,*

**Vocabulary Design**



- “it”
- “was”
- “the”
- “best”
- “of”
- “times”
- “worst”
- “age”
- “wisdom”
- “foolishness”

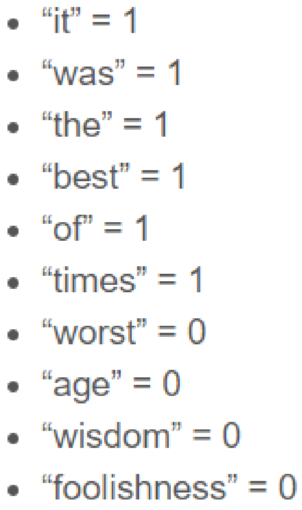
That is a vocabulary of 10 words from a **corpus** containing 24 words.

“**Corpus** is a large collection of texts. It is a body of written or spoken material upon which a linguistic analysis is based.”

## STEP C – Document vectors Creation

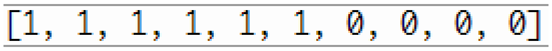
The next step is to score the words in each document.
The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.

Result of our small example:



- “it” = 1
- “was” = 1
- “the” = 1
- “best” = 1
- “of” = 1
- “times” = 1
- “worst” = 0
- “age” = 0
- “wisdom” = 0
- “foolishness” = 0

**Numerical vector Result**



```
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
```

The result of above steps is a **Numerical Vector** which can be utilized as inputs in the various machine learning algorithms

## Managing Vocabulary

- As the vocabulary size increases, so does the vector representation of documents. You can imagine that for a very large corpus, such as thousands of books, that the length of the vector might be thousands or millions of positions. Further, each document may contain very few of the known words in the vocabulary. This results in a vector with lots of zero scores, called a sparse vector or sparse representation.
- Sparse vectors require more memory and computational resources when modeling and the vast number of positions or dimensions can make the modeling process very challenging for traditional algorithms.
- As such, there is pressure to decrease the size of the vocabulary when using a bag-of-words model. This is done by decreasing the number of features used by the model to create the Bag of Words.

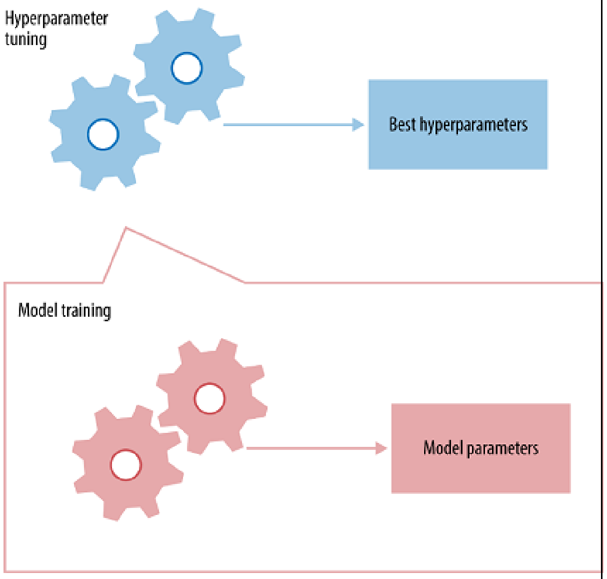
# 5. TASK IMPLEMENTATION

## Hyperparameter Tuning

Gathering more data and feature engineering usually has the greatest payoff in terms of time invested versus improved performance, but when we have exhausted all data sources, it’s time to move on to model hyperparameter tuning…

### Explanation

- 
- Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error.



**Difference between model *parameters* and *hyperparameters***

**model parameters**

are learned during training such as the slope and intercept in a linear regression

**hyperparameters**

must be set by the data scientist before training.

- Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting.

- An overfit model may look impressive on the training set but will be useless in a real application.

  Therefore, the standard procedure for hyperparameter optimization accounts for overfitting through **Cross Validation**.

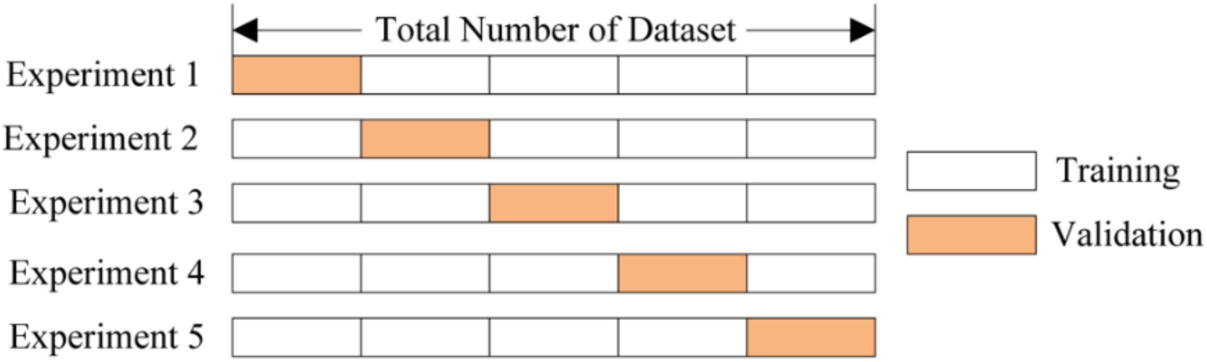
### Cross Validation

- We used the technique of cross validation (CV) using the method, **K-Fold Cross Validation.**

  We split our training set into a training and a testing set (validation set). We don't touch the original test set in the K-Fold CV.

- **K-Fold CV**, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data).

  Here is an illustrative example with k=5



### Using K-Fold Cross Validation for Hyperparameter Tuning

- For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings.

  We then compare all the models, select the best one, train it on the full training set, and then evaluate on the testing set.

- Each time we want to assess a different set of hyperparameters, we must split our training data into K fold and train and evaluate K times.

If we have 10 sets of hyperparameters and are using 5-Fold CV, that represents 50 training loops.

In our project ,we use model tuning with K-Fold CV can be automatically implemented in Scikit-Learn.

### Random Search CV in Scikit-Learn

- Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges for each parameter we want to tune, and randomly sample from the grid, performing K-Fold CV with each combination of values.

To use RandomizedSearchCV, we first need to create a parameter grid to sample from during fitting…

### Random Hyperparameter Grid

- On each iteration, the algorithm will choose a difference combination of the features.

  Altogether, there are (n1 * n2 * n3 * n4 * n5 * n6 ) combinations !!!!! However, the benefit of a random search is that we are not trying every combination but selecting at random to sample a wide range of values.
- The most important arguments in RandomizedSearchCV are **n_iter**, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation.

  More iterations will cover a wider search space and more cv folds reduces the chances of overfitting but raising each will increase the run time. Machine learning is a field of trade-offs, and performance vs time is one of the most fundamental.
- After doing the RandomizedSearchCV ,Random search allowed us to narrow down the range for each hyperparameter. Now that we know where to concentrate our search, we can explicitly specify every combination of settings to try. We do this with GridSearchCV, a method that, instead of sampling randomly from a distribution (RandomizedSearchCV), evaluates all combinations we define.

## Model Saving

- Now we are ready to apply any classifier that will do the 2 following procedures:
    1. Fitting and creating a decision model based on Training model.
    2. Testing the created model from the classifier

These operations usually take large times, so for testing purposes we had saved our model created by each classifier to quickly reuse it many times.

- We saved our trained algorithm so every time we want to use our trained algorithm we don't have to go through and retrain it . So I can save my trained algorithm it's very useful specially if you're doing the same training set every time .
- We are Saving using **pickles**

  After training the classifier once, we can save this classifier

  in file.pickle .

  1- Open it in extension (wb) 'write in byte'.

  2- Pickle.dump : to save the classifier.

  3- Close the file.

  So, after that :

  1- We can open this file in the extenstion(rb) 'read in byte'.

  2- pickle.load: to load the classifier

  3- Close file

  4- We can use this file to predict labels of the test set .

## Classifiers

### Random Forest Classifier

**Adjusted Hyperparameters**

**n_estimators** integer, optional (default=10)

The number of trees in the forest.

**max_features** int, float, string or None, optional (default=”auto”)

The number of features to consider when looking for the best split.

- If int, then consider max_features at each split.
- If float, then max_features is a fraction and int (max_features * n_features) features are considered at each split.
- If “auto”, then max_features=sqrt(n_features).
- If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).
- If “log2”, then max_features=log2(n_features).
- If None, then max_features=n_features.

**max_depth** : integer or None, optional (default=None)

The maximum depth of the tree.

If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

**min_samples_split** : int, float, optional (default=2)

The minimum number of samples required to split an internal node.

**min_samples_leaf** : int, float, optional (default=1)

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.

## Brief Explanation

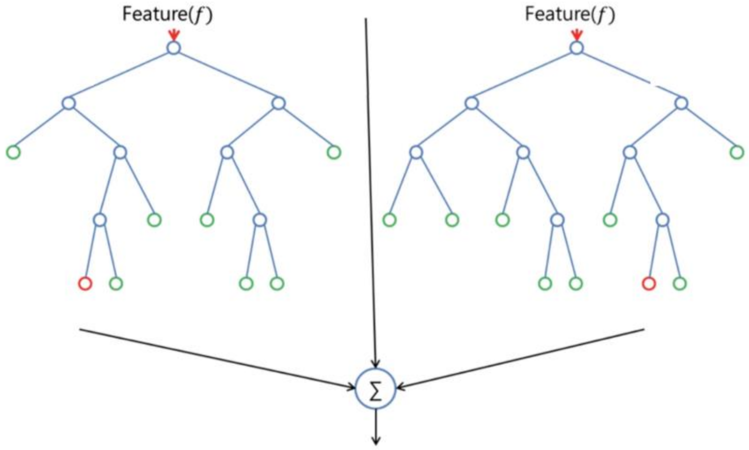
- Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

- Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The „forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

***In simple word***

***Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.***

- One big advantage of random forest is, that it can be used for both classification and regression problems, which form most current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



- Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, you don't have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.
- Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.
- Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

## Naïve Bayes Classifier

### Brief Explanation

- Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output. Naive Bayes classifiers have been successfully applied to many domains, particularly Natural Language Processing(NLP).
- It is particularly suited when the dimensionality of the inputs is high. Because of its simplicity (naïve assumption) it would be very attractive to be used in an NLP classification problem.
- Naïve Bayes classifier do not require any parameter to be tuned. Also, it's the fastest algorithm to classify as all it does applying some calculations on a given model.

## Decision Tree Classifier

### Adjusted Hyperparameters

**criterion** string, optional (default=”gini”)

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

**max_depth** int or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

### Brief Explanation

- Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

- A tree can be *“learned”* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*.

Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy.

The strengths of decision tree methods are:

- Decision trees can generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees can handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

## Logistic Regression Classifier

### Adjusted Hyperparameters

**C** float, default: 1.0

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

### Brief Explanation

- Logistic regression is named for the function used at the core of the method, the logistic function.
  The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.
- The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation.
  Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).
  The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

## K-Nearest Neighbor Classifier

### Adjusted Hyperparameters

**n_neighbors** int, optional (default = 5)

Number of neighbors to use.

### Brief Explanation

- KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

  To evaluate any technique, we generally look at 3 important aspects:

  1. Ease to interpret output
  2. Calculation time
  3. Predictive Power

- We can implement a KNN model by following the below steps:

  A. Load the data

  B. Initialize the value of k

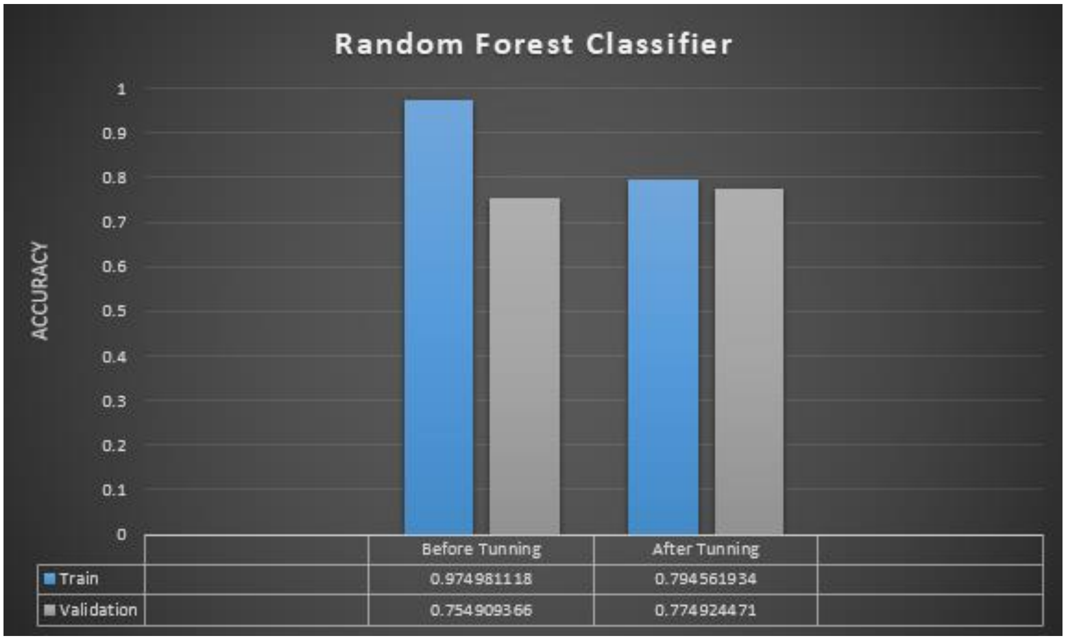
  C. For getting the predicted class, iterate from 1 to total number of training data points

     1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
     2. Sort the calculated distances in ascending order based on distance values
     3. Get top k rows from the sorted array
     4. Get the most frequent class of these rows
     5. Return the predicted class

# 6. RESULTS

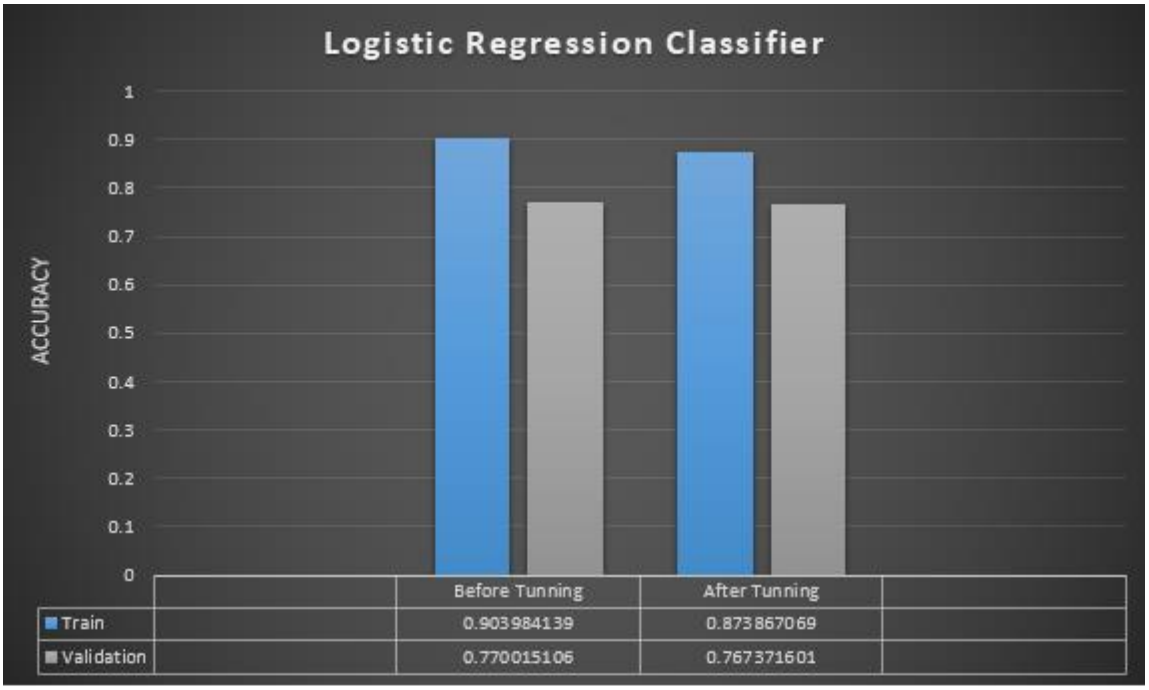
After running classification using all above mentioned classifiers, we got these results…

## Random Forest Classifier

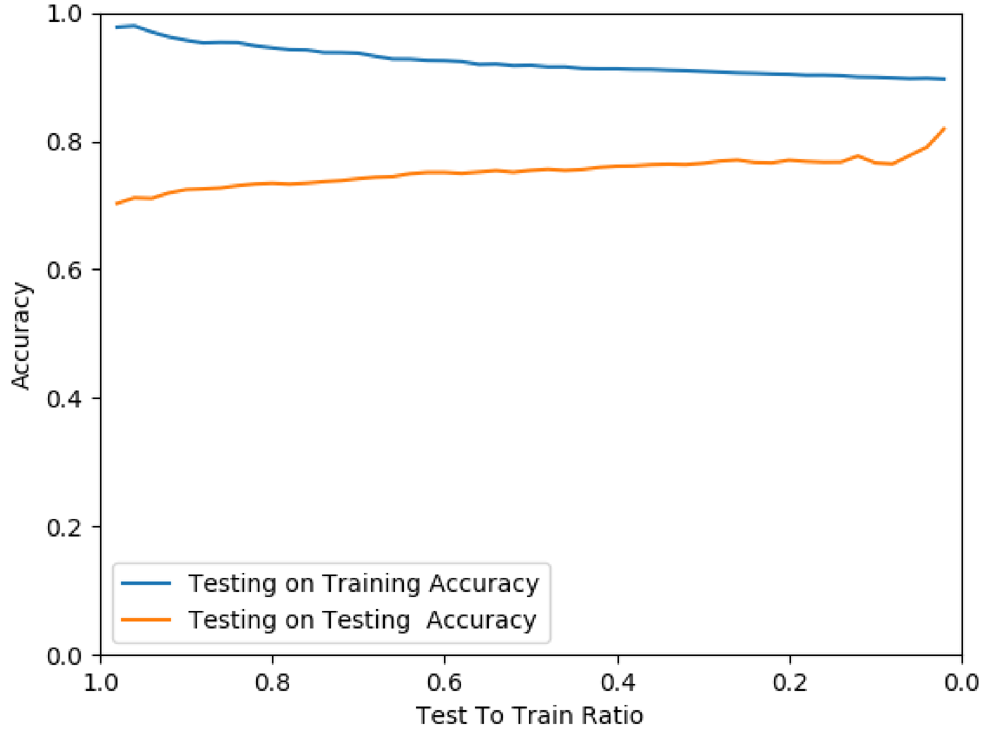


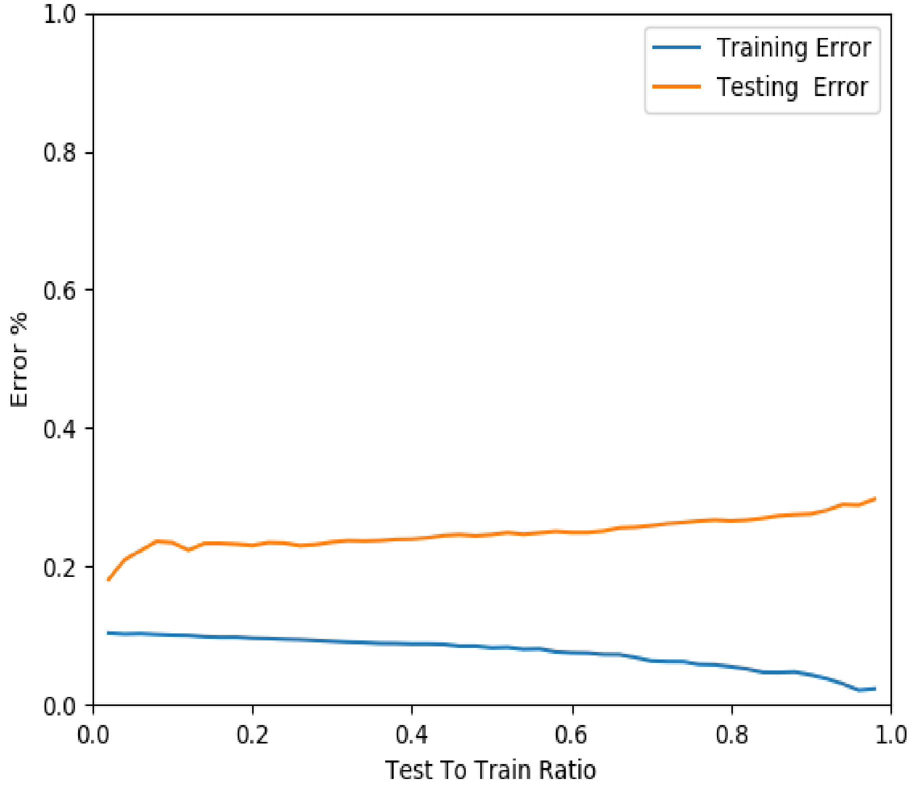
| | Before Tunning | After Tunning |
|---|---|---|
| ■ Train | 0.974981118 | 0.794561934 |
| ■ Validation | 0.754909366 | 0.774924471 |

## Logistic Regression Classifier

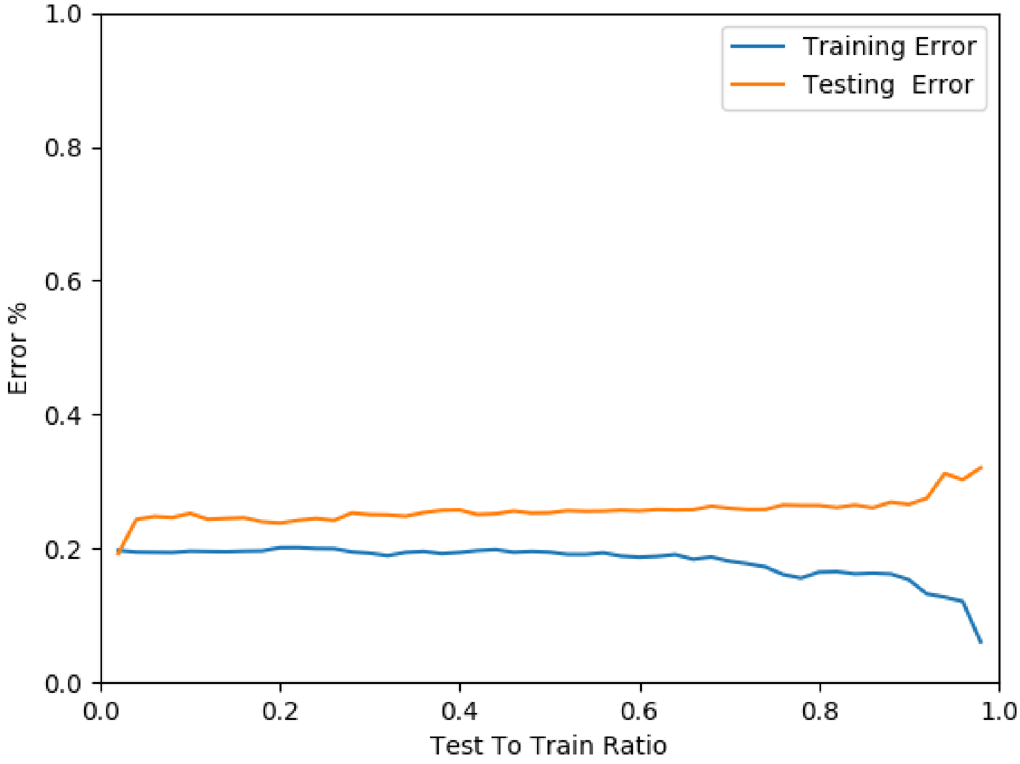


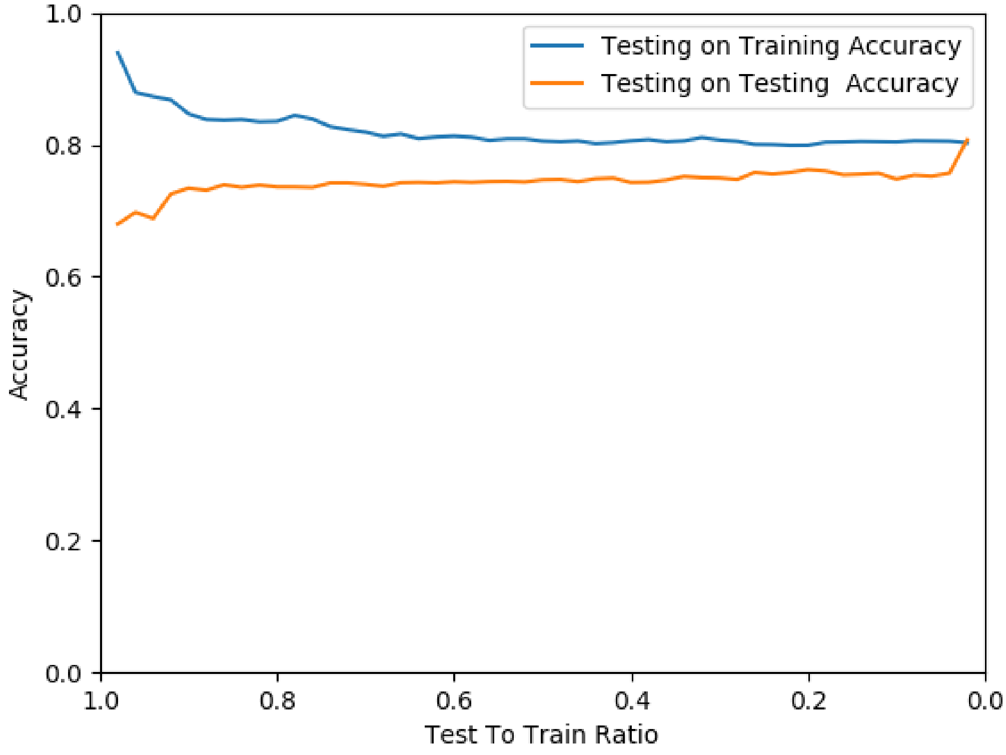
| | Before Tunning | After Tunning |
|---|---|---|
| ■ Train | 0.903984139 | 0.873867069 |
| ■ Validation | 0.770015106 | 0.767371601 |


1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Test To Train Ratio
Testing on Training Accuracy
Testing on Testing  Accuracy

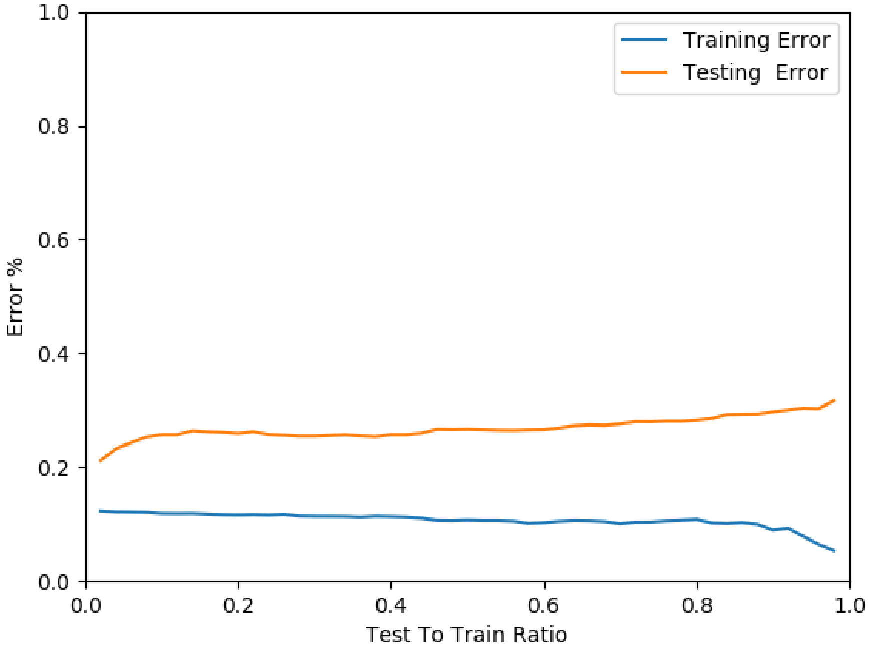

1.0
0.8
0.6
0.4
0.2
0.0
Error %
0.0
0.2
0.4
0.6
0.8
1.0
Test To Train Ratio
Training Error
Testing  Error

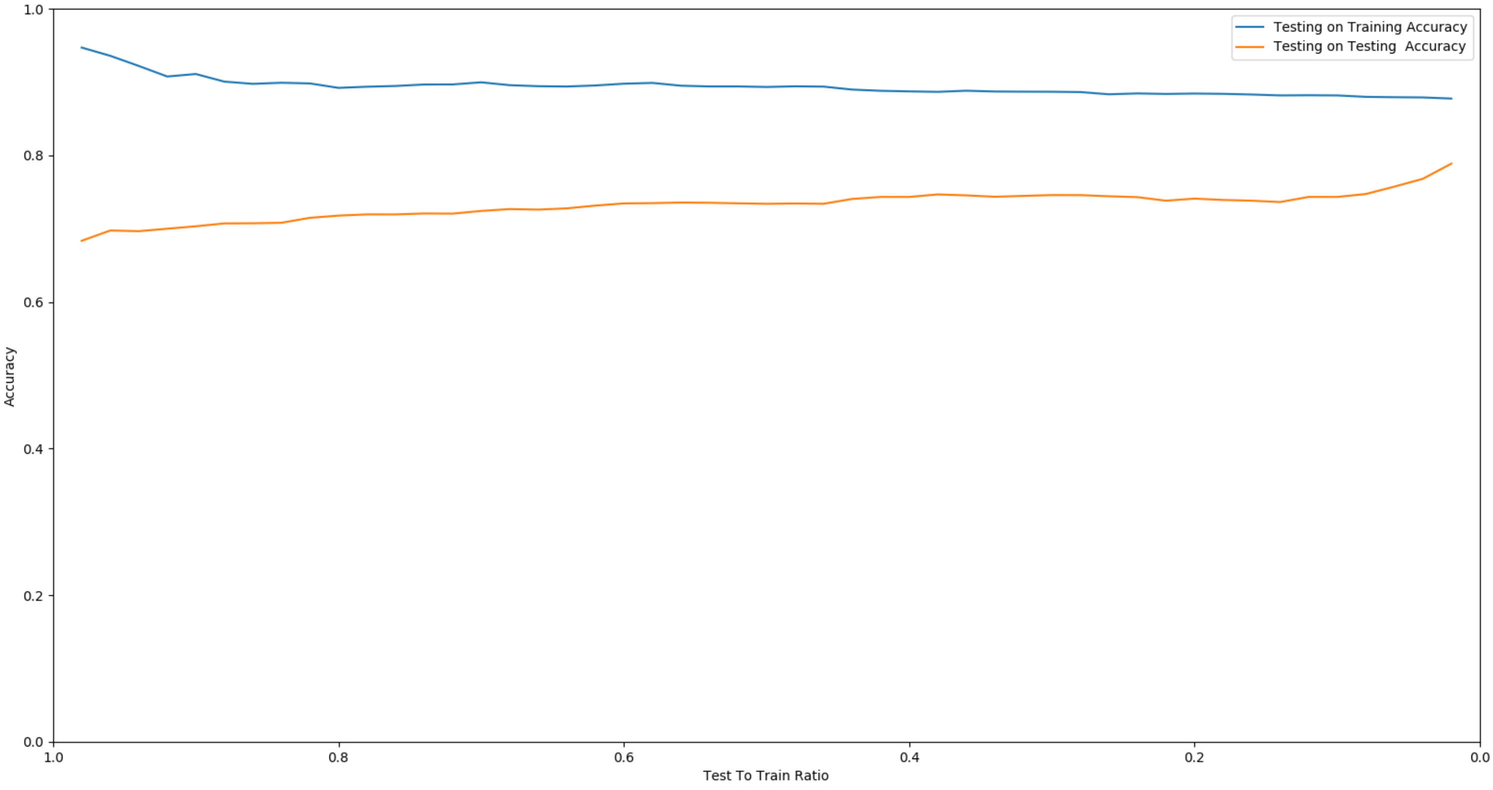
# Decision Tree Classifier





# Naïve Bayes Classifier





# 7.IMPORTANT LIBRARIES USED

### Scikit-learn

Machine learning in Python.

From where we implemented the following functions:

CountVectorizer
train_test_split
confusion_matrix
MultinomialNB
RandomForestClassifier
RandomForestRegressor
DecisionTreeClassifier
LogisticRegression

### Natural Language Toolkit (NLTK)

The complete toolkit for all NLP techniques.

From where we implemented the following functions:

stopwords
WordNetLemmatizer
word_tokenize

### Pandas

Python has long been great for data munging and preparation, but less so for data analysis and modeling. *pandas* help fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

### Pickle

Used in saving each created model from classifiers in order to reuse it so much saving in running time.

# 8. REFERENCES

https://competitions.codalab.org/competitions/20011?fbclid=IwAR1tkbbZntXwceDwq0Cpzv_wuytH8plyXp50U3HGn5Qjf-Kj3f9gdj2XxDk#learn_the_details-overview

https://courses.edx.org/courses/course-v1:ColumbiaX+CSMM.101x+3T2018/course/

https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/?fbclid=IwAR2MsLQ_72ep3cAbqBIYMpWwQP5t8UAr6wN5-MpSM4wFibd2jsSlnwUujTI

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/?fbclid=IwAR3f7C0bDxDXmrNDY326jy09k2J7_bzj9GE7bs6wk-24lg3soscI59QSNBU

https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41

https://en.wikipedia.org/wiki/Feature_extraction

https://www.tutorialspoint.com/python/python_tokenization.htm

http://www.insightsbot.com/blog/R8fu5/bag-of-words-algorithm-in-python-introduction

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74?fbclid=IwAR2hkODabmoAI0opmK9ANWqNn-rponFaqVQJDmkxy1taXpZFNmx_iTrGuEc https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

https://towardsdatascience.com/introduction-to-nlp-5bff2b2a7170

https://machinelearningmastery.com/gentle-introduction-bag-words-model/

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?fbclid=IwAR0DddCw_TBQ5GPpI1zqt-ejZnHPry6d7AcooahRbytGWv0_ecpsytYkR9I

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html?fbclid=IwAR1t8zqlQliuXlYG1BK_Lhd-Xfs-xQnBBItz0Jeq7yfAsqPZaZe4v4JHV7s

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?fbclid=IwAR3VowCsjerp7aGN4pEXGzoyynNWBysl0KlgH6-AvJFWxt80aypqiZFSPDs

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?fbclid=IwAR314vd7MQT3ckVrJZG1DVaySrqVFObscYEIq9T1c8Ut_y2FJm7YNEL_zqg

https://pandas.pydata.org/

**THANK YOU**