CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

PROJECT REPORT

Comparative Analysis of Prediction Accuracies of ML Classifiers in Classification of Text Articles

PREPARED BY

Saikat Chakraborty

Anirban Manna

POST GRADUATE DIPLOMA IN BIG DATA ANALYTICS

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

KOLKATA

TABLE OF CONTENTS

ACKNOWLEDGEMENT

First of all we would like to take the opportunity to express our profound gratitude to Mr. Asok Bandyopadhyay, Associate Director, CDAC Kolkata, for giving us the opportunity to work in this project titled COMPARATIVE ANALYSIS OF PREDICTION ACCURACIES OF ML CLASSIFIERS IN CLASSIFICATION OF TEXT ARTICLES. His valuable and knowledgeable guidance helped us throughout the project.

We are highly indebted to our Project guide, Mr. Debabrata Pal, Principal Technical Officer, CDAC Kolkata, for his valuable suggestions, insight, helpful advices and constant encouragement throughout the project.

We would also like to express our gratitude towards Mr. Bibekananda Kundu, Joint Director, CDAC Kolkata, for his valuable insights and assistance which allowed us to overcome several obstacles throughout the course of the project.

Last but not the least, our parents are the source of constant inspiration for us in all facets of our daily life. So with due regards, we express our gratitude to them.

Saikat Chakraborty

(190870625005)

Anirban Manna

(190870625004)

CERTIFICATE

This is to certify that the project report titled "COMPARATIVE ANALYSIS OF PREDICTION ACCURACIES OF ML CLASSIFIERS IN CLASSIFICATION OF TEXT ARTICLES" which is submitted by Saikat Chakraborty and Anirban Manna in partial fulfillment of the requirement for the award of POST GRADUATE DIPLOMA IN BIG DATA ANALYTICS from the CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING, Kolkata, is a record of the candidates' bona fide work carried out by them under our supervision and guidance. The matter embodied in this project report is original.

PROJECT GUIDE

Mr. Debabrata Pal

Principal Technical Officer

ICT&S

C-DAC, Kolkata

SUPERVISOR

Mr. Asok Bandyopadhyay

Associate Director

ICT&S

C-DAC, Kolkata

ABSTRACT

The need for efficient tools to retrieve, categorize and analyze textual data is ever-present in the digital age. Text classification is a process of automated assignment of natural language texts into predefined categories based on their content. It is required primarily in text retrieval systems for providing response to user queries, and in text comprehension systems for answering simple to complex questions, producing summaries or extracting necessary data. The enormous data volume, inherent complexity of natural languages and high dimensionality of textual data are a few of the challenges faced in this process. By leveraging the prowess of machine learning models, we have investigated a number of methods in our project in order to classify online articles of newspapers and magazines into three broad categories - Sports, Style and Fashion, and Business and Economics. The models we have utilized in this project are - the Naive Bayes classifier, K-Nearest Neighbours classifier, Decision tree classifier, Logistic Regression classifier, Random Forest classifier and Multilayer Perceptron (MLP) classifier. These classifiers have been trained on a dataset that is obtained by utilizing web scraping methods on twenty different websites pertaining to each category. The scraped data has been preprocessed primarily utilizing the Natural Language Toolkit (NLTK) library available in Python. Our preprocessing steps include case conversion, removal of special and non-Unicode characters, contraction expansion, number removal, stopwords removal, lemmatization and stemming. The TF-IDF model has been used for feature extraction. We have attempted to perform a comparative analysis of the prediction accuracies of the mentioned classifiers. The project has been primarily designed using the Python language.

INTRODUCTION

Text Classification (or Text Categorization) is the process of consists of assigning one or more categories to a document, based on its semantic content. Traditionally, text categorization has a history that dates back to the 1960s. Over the last two decades, there has been an enormous increase in text documents available to the general public over the internet. This has intensified the need of people across all domains to procure automated text classification systems that is capable of handling the ever-increasing volume of high dimensional text data.

Some popular applications of Text Classification include Sentiment Analysis, Topic Labelling, Language Detection, Intent Detection, Brand Monitoring, Social Media Monitoring and Customer Services.

In Mathematical terms, Text Classification can be described as follows:

Set of training documents D = $\{d_1,...,d_N\}$ such that each record is labeled with a class value 'c' from C = $\{c_1,...,c_J\}$.

Traditional rule based approach for Text Classification involved categorizing text using handcrafted linguistic rules. These rules would instruct the system to use semantically relevant elements of a text to identify relevant categories based on its content. Each rule consisted of an antecedent or pattern and a predicted category. Rule-based systems were human comprehensible and could be improved over time. But this approach had some disadvantages. Crafting the rules required extensive domain knowledge, were time consuming to build and had problems with maintainability and scalability.

Text Classification with Machine Learning overcomes most of these problems by learning to make classifications based on past observations. By using pre-labeled examples as training data, a machine learning algorithm can learn the different associations between pieces of text and predict the expected category for an input text.

Machine Learning is a field of study under the broader category of Artificial Intelligence. It enables computational systems the capability of learning without being explicitly programmed in a tight rule-based system.

A Machine Learning model is called Supervised when the model is getting trained on a labelled dataset. Labelled dataset is one which have both input and output parameters as features and class label. A supervised learning task where the predicted output belongs to a particular class label, is called a classification task. The class can be binary (denoted with 0 and 1) or multi-class ($c_1, c_2, \ldots, c_n$).

Dataset is set of data values that contain features important to solving the classification problem. Features are distinct characteristics of the data that help us understand the problem. These are fed in to a Machine Learning algorithm to help it learn.

Natural Language Processing (NLP) is a wide area of research where the worlds of artificial intelligence, computer science, and linguistics collide. It is focused on enabling the computers to understand and interpret the human language.

In this project, the NLP functionality of python (using NLTK package) has been extensively exploited. Supervised classifiers have been used on a labelled dataset that has been built from scratch using web-scraping techniques. The articles analyzed may be classified into more than one category (Ranking Classification), but for the purpose of this project, only single categories have been taken under consideration.

Almost all text classification or categorization systems can be broken down into the following five basic phases: Data extraction, Data Preprocessing, Feature Extraction, Classifier Selection, and Model Training.

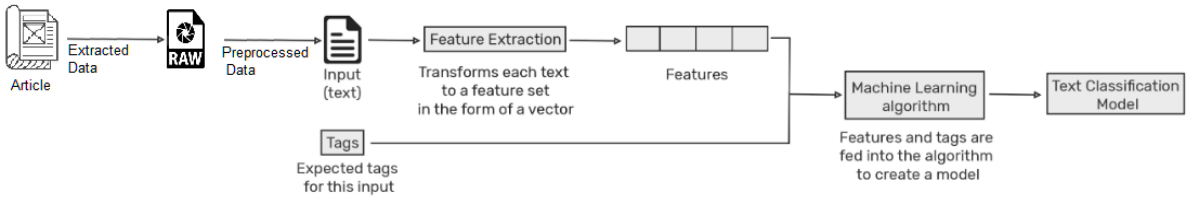


Figure 1 - Text Classification Process

Data extraction involves collecting the data from a single or multiple sources. Preprocessing involves converting the raw, extracted data in a useful and efficient format. Feature extraction is a method by which the preprocessed data is transformed into a numerical representation in the form of a vector, which is readable by Machine Learning models. Classifier selection involves fitting the training data into a suitable Machine Learning Classifier. Based on this, the model will get trained with the training data, after which it can give a prediction on test data and classify it into its suitable category.

In the following sections, the various phases have been explored in details, followed by the analysis of the evaluated models along with the inference and future scope of the project.

RELATED WORKS

Priyanka patil T. and S.I. Nipanikar [1] focused on the progress made so far in the character feature of scene text recognition system and provided an overview of the technological prospective of scene text recognition systems.

Kumuda T. and Basavaraj L [2] proposed a two stage hybrid text extraction approach by combining texture and CC-based information.

Miss. Poonam B. Kadam, Mrs.Latika R. and Desai [3] explored hybrid approaches to detect and recognize texts in CAPTCHA image. Through this approach the strength of CAPTCHA could be analyzed.

Navathe, shamkant B and Elmasri Ramez [4] discussed about the text mining process – a process of extracting interesting and non-trivial information and knowledge from unstructured text - and the techniques used in text mining.

Haralampos Karanikas and Manchester [5] explored Knowledge Discovery in Text and Text Mining Software.

Franca Debole and Fabrizio Sebastiani [6] proposed the idea of Supervised Term Weighting for automating categorization of text.

Takeru Miyato, Andrew M. Dai, Ian Goodfellow [7] explored adversarial training methods for Semi-supervised text classification.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov [8] explored fastText – a simple and efficient baseline for text classification designed by them.

Ghayda Altalib and Hind Salman [9] introduced a weighting method based on statistical estimation of the importance of a word for SMS categorization.

Waleed Zaghloul, Sang M. Lee, Silvana Trimi [10] compared the performance of neural networks (NN) and support vector machines (SVM) as text classifiers.

DATA EXTRACTION

This step deals with gathering the data from various resources that contain textual data.

This data can be *internal data* - generated from apps and tools that are used frequently like CRMs (e.g. Salesforce, Hubspot), chat apps (e.g. Slack, Drift, Intercom), help desk software (e.g. Zendesk, Freshdesk, Front), survey tools (e.g. SurveyMonkey, Typeform, Google Forms), and customer satisfaction tools (e.g. Promoter.io, Retently, Satismeter)

Another option is using *external data* available on the web, either by using web scraping, APIs, or public datasets.

In this project, the data has been gathered by scraping twenty different websites of each of the following category – *Sports*, *Business and Economics* and *Style and Fashion*.

The BeautifulSoup package of Python has been used for this.

```
url_to_parse = Request(url, headers={'User-Agent': 'Mozilla/5.0 (X11; U; Linux i686) Gecko/20071127 Firefox/2.0.0.11'})
html_raw = urlopen(url_to_parse).read()
webpage = html_raw.decode('utf-8')
soup = BeautifulSoup(webpage, features="lxml")
#For removing the scrips and style tags
for script in soup(["script", "style"]):
    script.extract()

# get raw text
text = soup.body.get_text()
```

The text obtained has been cleaned by removing leading and trailing spaces, breaking multi headlines into single lines, removing blank lines. Words that got concatenated together after removal of HTML elements have been split while keeping abbreviations (like BBCI, ICC) intact.

```
# break into lines and remove leading and trailing space on each
lines = (line.strip() for line in text.splitlines())

# break multi-headlines into a single line

chunks = (phrase.strip() for line in lines for phrase in line.split("  "))

# drop blank lines
text = '\n'.join(chunk for chunk in chunks if chunk)

#Fixed the word concatenation issue - Splitting at uppercase while
#keeping abbreviations intact
listSplits = re.findall(r'[A-Z](?:[A-Z]*(?![a-z])|[a-z]*)', text)
listToStr = ' '.join(map(str, listSplits))
```

We store the raw data into a Dataframe and store the same into a pickle file, to be used later for preprocessing.

```
sports_data = build_dataset_sports(seed_url_sports)
style_data = build_dataset_styleandfashion(seed_url_styleandfashion)
business_data = build_dataset_businessandeconomics(seed_url_businessandeconomics)

#Merge the three datasets into a single DataFrame
df_fulldataset_raw = pd.concat([sports_data, style_data, business_data],ignore_index=True)

#Save the DataFrame
df_fulldataset_raw.to_pickle(r'F:\scrapedtext\rawdataset')
```

DATA PREPROCESSING

The extracted data by itself is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. The data may contain repeated stopwords (explained later), slangs, contractions, misspellings and several other forms that can have adverse effects on the learning capability of most Machine Learning algorithms.

Data Preprocessing involves transforming the raw data into an understandable format for Natural Language Processing models. The preprocessed data will be devoid of noise and unnecessary features and will be 'clean'.

The steps taken for preprocessing has no formal rules. Some steps can be appropriate for some datasets while grossly inappropriate for others. The following steps taken are therefore suitable for data in this project. It may not be ideal for other Text Classification processes, although some steps will be similar, as they are required in all known Text Classification models.

Case conversion – This helps with maintaining consistency of expected output especially in case of datasets which are not very large. It brings all words diversified on the basis of capitalizations into the same feature space. It also resolves sparsity issues.

```
#change text to lowercase
if lower_case:
    doc = doc.lower()
```

Removal of special and non-Unicode characters – In this project, only English articles are considered. Therefore any special symbols or words containing non-Unicode characters, which are common mostly in cases of non-English languages, have been ignored for this Project, in order to reduce the data volume.

```
def remove_nonunicode_char(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text
```

Contraction expansion – Contractions in English language are in shortened versions of words and syllables (like *wouldn't* for *would not*). Converting each contraction to its expanded, original form helps with text standardization

```
def expand_contractions(text):

    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)

    # general
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)

    return text
```

Number and punctuation removal – There are usually two ways to handle numerical data in during Text Preprocessing. Either their removal or converting them into their textual representation. For this project, the relevance of numbers on the accuracy of the prediction of the models was not apparent, so they have been removed.

```
def remove_numbers(text):
    text = re.sub(r'\d+', '', text)
    return text
```

Punctuation removal – Punctuations have been removed to avoid having different forms of the same word.

```
def remove_punctuations(text):
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)
    return text
```

Tokenization – Tokenization (Text Segmentation) refers to the process of splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms – called *tokens*. Semantic interpretation of the text can be easily performed by analyzing the words individually in the text.

Stemming and Lemmatization – Word Stems or base forms of word (e.g. Jump for jumping, jumped, etc.) constitute the backbone of its various inflections. They are formed by joining *suffixes* and prefixes to the root or base word. **Stemming** refers to the reverse process of obtaining the base form of the word. Lemmatization is similar to Stemming but here, the base form is the root **word** (lemma) and not the stem. Root word is always a lexicographically correct word (present in the dictionary) but a Stem may not always be so. (e.g. eaten, ate, eating – all map to eat). For our purpose, retrieving the 'stem' and 'lemma' of the word and using it for all its inflections helps standardize words.

```
def lemmatize_text(text):
    word_tokens = tokenizer.tokenize(text)
    text = ' '.join([lemmatizer.lemmatize(word, pos ='v') for word in word_tokens])
    return text

def stem_text(text):
    ps = nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text
```

Stop words removal – Stop words are words which have little or no significance especially when constructing meaningful features from textual data. These are usually words which are most frequent in the corpus. (e.g. a, an, the…). In this project, a Standard English language stopwords list from NLTK has been used.

```
stopword_list = nltk.corpus.stopwords.words('english')
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
```

FEATURE EXTRACTION

The preprocessed text data is saved in a pickle file similar to the raw data, to be used for extracting features from it.

```
#Preprocessing the scraped data

df_raw = pd.read_pickle(r'F:\scrapedtext\rawdataset')
clean_data = preprocess_dataset(df_raw)

df_preprocessed = pd.DataFrame(clean_data, columns=['document'])

df_preprocessed.to_pickle(r'F:\scrapedtext\preprocessed_dataset')
```

This data is still in the form of **text**. Text themselves cannot be used by machine learning models. They expect their input to be numeric.

Feature extraction methods transform input text into numeric feature in a meaningful way. One of the most popular representation of features in textual data is in the form of **Vector Space Model** (VSM) – a mathematical model to represent unstructured text document as numeric vectors, such that each dimension of the vector is a specific feature/attribute.

This project implements the *TF-IDF (Term Frequency – Inverse Document Frequency)* model to extract relevant features from the document.

This approach is a normalized version of the *Bag of Words (BOW)* approach, which represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values.

In *Bag of Words* model, the feature vectors are based on absolute term frequencies. Because of this, the terms which occur frequently across all documents may tend to overshadow other terms in the feature set.

The TF-IDF based feature vectors for each text document overcomes the lacuna of the BOW approach by using scaled and normalized values in its computation. TF-IDF are word frequency scores that highlight words that are more interesting, e.g. frequent in a document but not across documents. In other words, it evaluates how important a word is, in a particular text document. So for each document (d_j) the words are represented as vectors ($w_{1,j}$, $w_{2,j}$,..., $w_{t,j}$) where $1,2,...,t$ represents the number of terms in the document.

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

TF-IDF stands for Term Frequency-Inverse Document Frequency, uses a combination of two metrics in its computation, viz: *term frequency (tf)* and *inverse document frequency (idf).*

This technique was developed for ranking results for queries in search engines and now it is an indispensable model in the world of information retrieval and NLP.

Mathematically, we can define TF-IDF as *tfidf = tf * idf*, which can be expanded further to be represented as follows.

$$tfidf(w,D) = tf(w,D) \times idf(w,D) = tf(w,D) \times log\left(\frac{C}{df(w)}\right)$$

Here, ***tfidf(w, D)*** is the TF-IDF score for word w in document ***D***. The term ***tf(w, D)*** represents the **term frequency** of the word ***w*** in document ***D***, which can be obtained from the Bag of Words model.

The term ***idf(w, D)*** is the inverse document frequency for the term ***w***, which can be computed as the log transform of the total number of documents in the corpus ***C*** divided by ***df(w)***, the document frequency of the word w, which is basically the frequency of documents in the corpus where the word w occurs.

In this project, we use the **TfidfVectorizer** class from **sklearn** package in Python to convert our preprocessed dataset into a TF-IDF feature matrix. We have selected a maximum of 1500 unique features (word) from our corpus containing over 6000 words.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(max_features = 1500, min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(processed)
tv_matrix = tv_matrix.toarray()
vocab = tv.get_feature_names()
X = pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

creat	creativ	credit	cricket	crossword	crude	cs
0	0	0	0	0	0	0
0	0	0	0.01	0	0	0
0.01	0	0	0.14	0	0	0
0	0	0	0.05	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0.1	0.08	0	0
0	0	0	0.05	0.04	0	0
0	0	0	0.01	0	0	0
0	0	0	0	0	0	0
0.04	0.02	0	0	0	0	0.08
0	0	0	0	0	0	0

Figure 2 - TF-IDF Matrix snapshot

PREPARING TRAINING AND TESTING DATASETS

To build and train the Machine Learning models, The Corpus is split into two data sets, *Training* and *Test*. The training data set will be used to *fit* the model and the *predictions* will be performed on the test data set.

In this project two approaches have been explored for this – **Train/Test Split** and **K-fold Cross Validation**.

In the Train/Test Split approach, the dataset is *split* based on a parameter called *test_size*. In this project, *test_size* has been set to 0.2, which means 80% of the corpus will be used as *Training Data* and remaining 20% will be used for Testing the model.

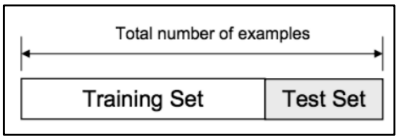


Figure 3 - Train-Test Split

This approach gives rise to a possibility of a high bias if limited data is available for training, because there is a chance of missing information about the data which have not been used for training.

In order to build a less biased model, the K-fold Cross Validation method is used. In this approach, the dataset is split into *k* equal partitions (*folds*), *one fold at a time* is used for *testing* and the *union of other folds* as *training*. The testing accuracies of each iteration of a different fold, is calculated and *averaged*.

By using this approach, all the data is effectively used for training, more metrics about the data is obtained and each model's input stays independent of the other models. Thus, it helps in a more efficient use of the entire training dataset.

In this project, the **KFold** class from **sklearn** package of Python has been used.

```
#K-fold cross validation
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)

for train_index, test_index in kf.split(X,y):
      X_train, X_test = X.iloc[train_index], X.iloc[test_index]
      y_train, y_test = y[train_index], y[test_index]
```

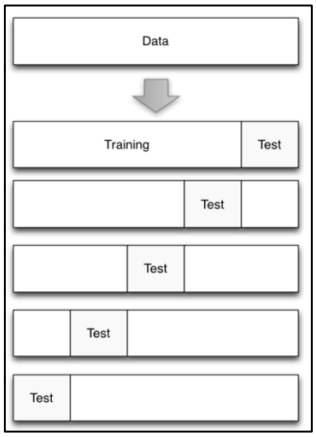


Figure 4 - K-fold Cross Validation (5 folds)

The final evaluation of the models have been done using the *K-fold Cross Validation* approach, with the value of k taken as **10**, as a generalized standard.

PREDICTION USING MACHINE LEARNING CLASSIFIERS

The classifiers used for the Project are:

1. Naïve Bayes Classifier
2. K-Nearest Neighbour (KNN) Classifier
3. Decision Tree Classifier
4. Random Forest Classifier
5. Logistic Regression (Multiclass) Classifier
6. Multilayer Perceptron (MLP) Classifier

The **sklearn** package has been utilized for all the classifiers. The *fit()* function has been used to fit the training data into the classifier and the *predict()* function has been used to perform the classification function on the test data.

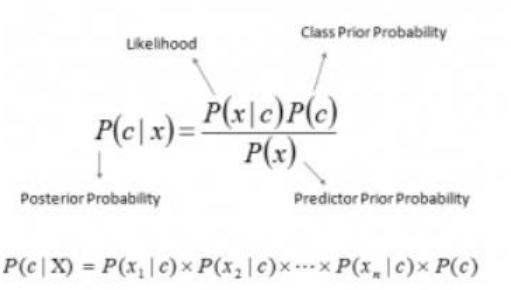
The following sections contain a brief overview of the various Classifiers along with the code used to implement them in this project.

NAÏVE BAYES CLASSIFIER

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors.

In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naïve Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.



***P*(*c*|*x*)** is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).

***P*(*c*)** is the prior probability of *class*.

***P*(*x*|*c*)** is the likelihood which is the probability of *predictor* given *class*.

***P*(*x*)** is the prior probability of *predictor*.

In our project, we use the **GaussianNB** class from **sklearn** package to implement this classifier.

```
#Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

K- NEAREST NEIGBOURS (KNN)

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

KNN is simple to understand, implement, fast and has a fair amount of predictive power.

The basic process is explained in the following steps:

1. *Load the data*
2. *Initialize the value of k*
3. *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.* ***Euclidean distance*** *is the most popular method.*
    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*

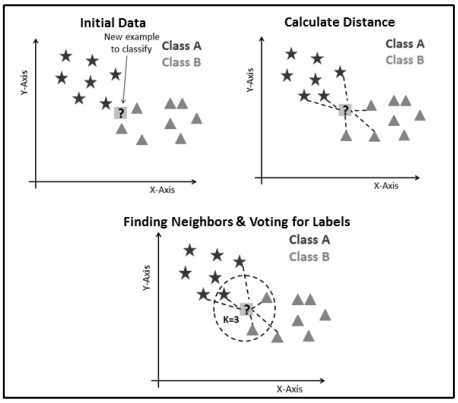


Figure 5 - KNN process

In our project, we use the **KNeighborsClassifier** class from **sklearn** package to implement this classifier.

```
#K-Nearest Neighbour (KNN) Classifier

from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)

neighpred = neigh.predict(X_test)
```

DECISION TREE

Decision Tree is a simple representation for classifying examples. It is a supervised classifier where the data is continuously split according to a certain parameter.

Decision Tree consists of

1. *Nodes*: Test for the value of a certain attribute.
2. *Edges/ Branch*: Correspond to the outcome of a test and connect to the next node or leaf.
3. *Leaf nodes*: Terminal nodes that predict the outcome (represent class labels or class distribution).

Decision Tree is inexpensive to construct, extremely fast at classifying unknown records, easy to interpret for small-sized trees and excludes unimportant features. However, it is also easy to overfit and small changes in the training data can result in large changes to decision logic.

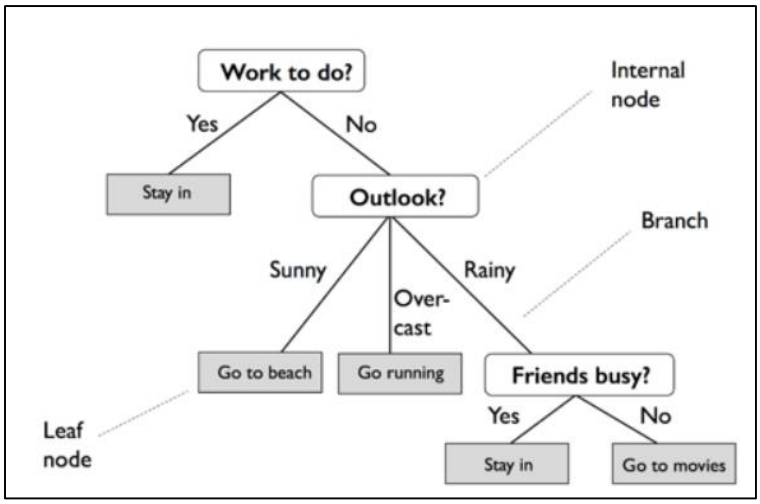


Figure 6 - Decision Tree

In our project, we use the **DecisionTreeClassifier** class from **sklearn** package to implement this classifier.

```
#Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=10)
dt.fit(X_train, y_train)
dtpred = dt.predict(X_test)
```

RANDOM FOREST

Random Forest is an ensemble learning method for classification, regression and other tasks that operates by constructing a *multitude of decision trees* at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random Forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of *majority voting.*

It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

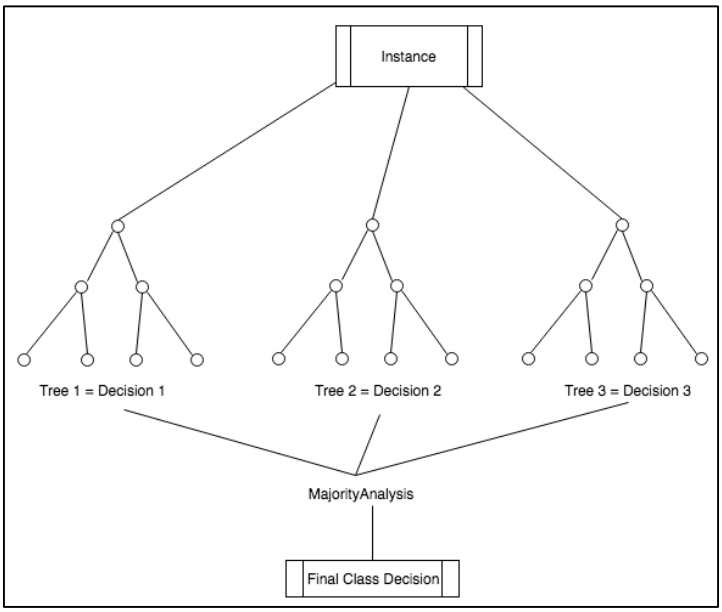


Figure 7 - Random Forest

In our project, we use the **RandomForestClassifier** class from **sklearn** package to implement this classifier.

```
#Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1)
rf.fit(X_train, y_train)
rfpred = rf.predict(X_test)
```

LOGISTIC REGRESSION (MULTICLASS)

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes.

Unlike linear regression which outputs *continuous number values*, logistic regression *transforms* its output using the logistic sigmoid function to return a *probability value* which can then be mapped to two or more *discrete classes*.

Linear Regression Equation:

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

$$p = 1/1 + e^{-y}$$

Apply Sigmoid function on linear regression:

$$p = 1/1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 \ldots\ldots \beta nXn)}$$

In this project, the logistic regression is multiclass. The binary classifier is re-run multiple times, once for each class.

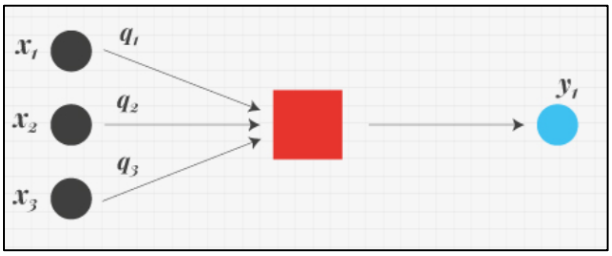


Figure 8 - Logistic Regression for 3 classes

In our project, we use the **LogisticRegression** class from **sklearn** package to implement this classifier.

```
#Logistic Regression multiclass Classifier

from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e5, multi_class='auto', solver='lbfgs')
logreg.fit(X_train, y_train)

logpred = logreg.predict(X_test)
```

MULTILAYER PERCEPTRON (MLP)

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes.

Multilayer perceptron (MLP) is a classifier based on the *feed-forward* artificial neural network. MLP Classifier consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network.

A simple MLP model consists of one *input layer*, one *hidden layer* and one *output layer*.

Nodes in the input layer represent the **input data**. All other nodes maps inputs to the outputs by performing linear combination of the inputs with the node's *weights* ***w*** and *bias* **b** and applying an *activation function*.

Weight increases the *steepness* of activation function. This means weight decide how *fast* the activation function will trigger whereas bias is used to *delay* the triggering of the activation function. Bias is used to adjust the output of a neuron along with the weighted sum of the inputs to the neuron.

output = sum (weights * inputs) + bias

The number of nodes in the output layer corresponds to the number of classes in a multiclass classification problem, like in this project. In this case a *softmax* function may be used to output a probability of the network predicting each of the class values.

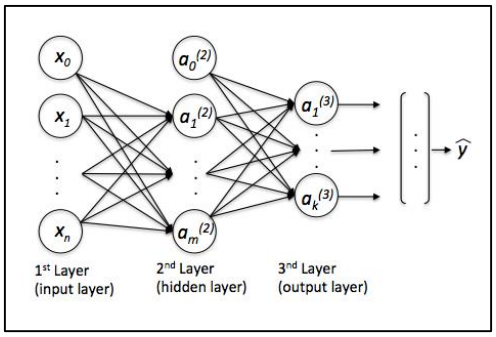


Figure 9 - MLP diagram

In our project, we use the **LogisticRegression** class from **sklearn** package to implement this classifier.

```
#Fitting Multilayer Perceptron Classifier

from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(alpha=1, max_iter=1000)

mlp.fit(X_train, y_train)

mlppred = mlp.predict(X_test)
```

EVALUATION OF MODELS AND ANALYSIS

The classifier evaluations are done based on the following constraints

1. Corpus size of 60 (20 for each Category).
2. Three categories (Sports, Style and Fashion, Business and Economics) labelled 1, 2 and 3 respectively, by hand.
3. Each classifier is run for 100 iterations and the prediction accuracies of each are averaged to obtain an average accuracy score.
4. 1500 unique features are considered from a feature set of over 6000.
5. Only TF-IDF model has been used for feature extraction.

There are various methods to determine a model's effectiveness, but the most used are **precision**, **recall** and **accuracy.**

To determine the above measures, it must first be determined whether the classification of the document was a *True Positive (TP), False Positive (FP), True Negative (TN)* or *False Negative (FN).*

TP	Determined as a document being classified correctly as relating to a category.
FP	Determined as a document that is said to be related to the category incorrectly.
FN	Determined as a document that is not marked as related to a category but should be.
TN	Documents that should not be marked as being in a particular category and are not.

Table 1 – Classification possibilities of a Document

Precision means the percentage of results which are relevant. It measures the *exactness* of a classifier. A *higher* precision means *less* false positives, while a lower precision means more false positives.

Recall refers to the percentage of total relevant results correctly classified by the classifier. Recall measures the *completeness*, or sensitivity, of a classifier. *Higher* recall means *less* false negatives, while lower recall means more false negatives.

Precision and recall can be combined to produce a single metric known as *F-measure*, which is the weighted harmonic mean of precision and recall.

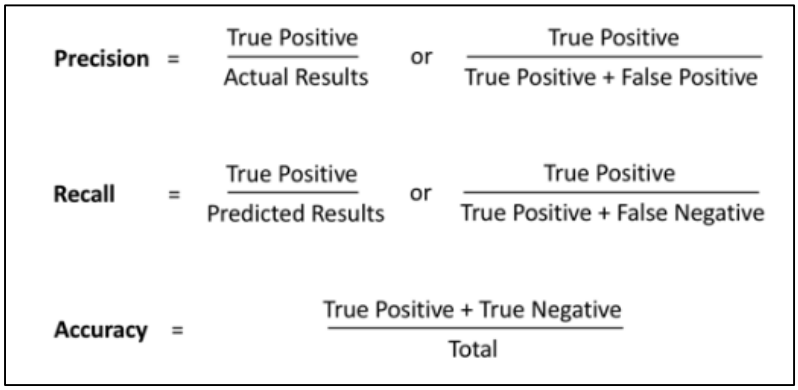
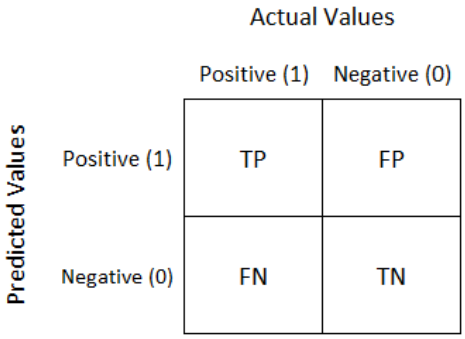


Figure 10 - Precision, Recall and Accuracy

All these measures can be calculated from the ***confusion matrix***.

A confusion matrix is a table that is often used to describe the performance of a classifier, on a set of test data for which the true values are known.

A confusion matrix is arranged in the following order:



In our project, we have generated the confusion matrix by importing **confusion_matrix()** function from **sklearn.metrics.**

```
from sklearn.metrics import confusion_matrix
log_cm = confusion_matrix(y_test, logpred)
print("Confusion Matrix:\n",log_cm)
```

Result:

```
Confusion Matrix:
 [[3 1 0]
 [0 0 0]
 [0 0 2]]
```

For a three-class classification problem, the confusion matrix is structured like this.

Confusion Matrix		Predicted: Sports	Predicted: Style and Fashion	Predicted: Business and Economics
Actual	**Sports**	3 ***(TP)***	1	0
	Style and Fashion	0	0 ***(TP)***	0
	Business and Economics	0	0	2 ***(TP)***

Table 2 – Confusion Matrix for a 3 class classification

Instead of manually calculating, we can get the scores of precision, recall, accuracy and even F1-score from **classification_report()** function under **sklearn.metrics**.

```
from sklearn.metrics import classification_report
print(classification_report(y_test,logpred))
```

Result:

```
              precision    recall  f1-score   support

           1       1.00      0.75      0.86         4
           2       0.00      0.00      0.00         0
           3       1.00      1.00      1.00         2

    accuracy                           0.83         6
   macro avg       0.67      0.58      0.62         6
weighted avg       1.00      0.83      0.90         6
```

Precision and Recall (Macro) Analysis of Classifiers (Averaged over 100 iterations)

	Naïve Bayes	KNN	Decision Tree	Random Forest	Logistic Regression	MLP
Precision (Macro)	0.89	0.735	0.646	0.54	0.915	0.845
Recall (Macro)	0.83	0.78	0.662	0.536	0.94	0.883

Table 3 – Macro Precision and Macro Recall of Classifiers

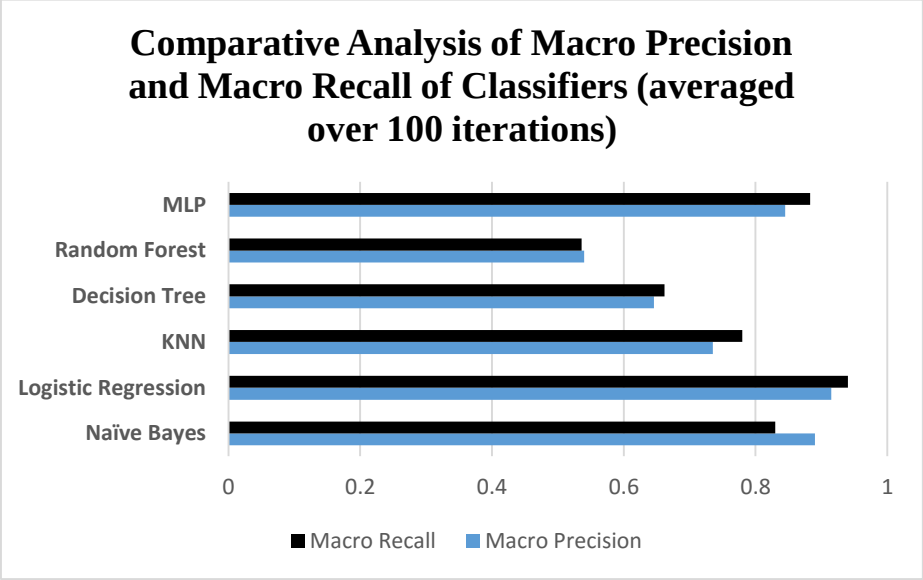


Figure 11 - Comparative Analysis of Precision and Recall of Classifiers

Accuracy Analysis of Classifiers (Averaged over 100 iterations)

Naïve Bayes	KNN	Decision Tree	Random Forest	Logistic Regression	MLP
83.34 %	78.28 %	74.67 %	58.22 %	89.92 %	88.91 %

Table 4 – Average Accuracies of Classifiers

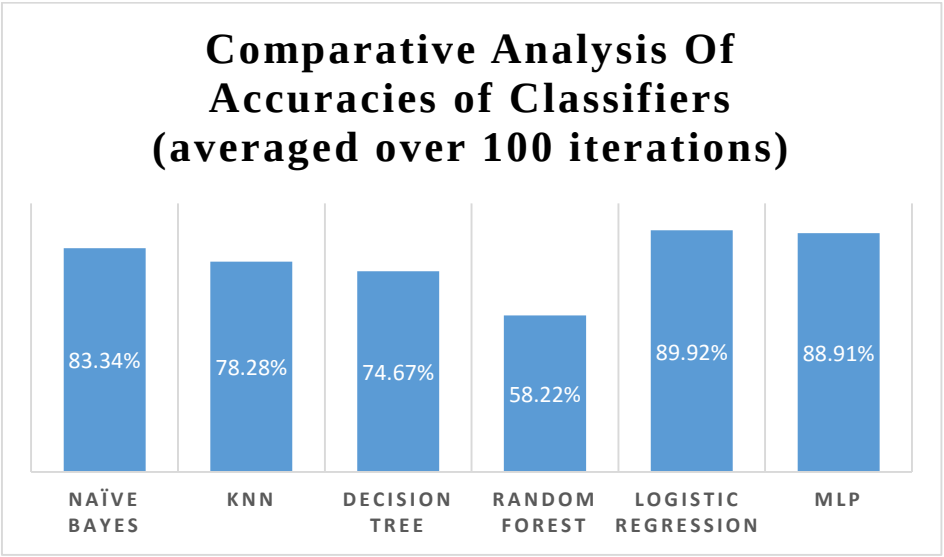


Figure 12 - Comparative Analysis of Classifier Accuracies

CONCLUSION AND FUTURE SCOPE

Conclusion

In this project, a fast and efficient method of text classification by exploiting the rich libraries of the Python language has been presented.

By comparing the prediction accuracies achieved by the six classifiers explored, it is evident from the analysis that *Logistic Regression (Multiclass) classifier* performs the *best* in terms of accuracy (~90%) followed closely by *Multilayer Perceptron (MLP) Classifier (~89%). Random Forest* performs the *poorest* in terms of accuracy (~58%).

Naïve Bayes, one of the simplest in terms of implementation and mechanism, gives a pretty decent accuracy of over 83%. K-Nearest Neighbour (KNN) and Decision Tree Classifiers gives an average accuracy.

In terms of Macro Precision and Macro Recall (Macro is considered because even though it's a multi-class classification scenario and there might be class imbalance, it is assumed for this project that the class labels are independent), *Logistic Regression (Multiclass) classifier* is the clear winner (both values over 0.9) followed by *Multilayer Perceptron (MLP).* Both these classifiers generated the lowest number of False Positives (FP) and False Negatives (FN). *Random Forest* has the lowest metric for both amongst all the classifiers.

Some important issues have not yet been tackled in this project. They include

a) Testing the classifiers with a larger corpus size to see if the measures for model evaluations improve.
b) Exploring the effect of proper dimensionality reduction on the datasets to accommodate for testing of articles with lesser number of features than the training data.
c) Exploring other models of feature extraction and measure their performance against the TF-IDF model (Document Similarity, Topic Models, Word2Vec model etc.)

Future Scope

Future research on this topic may include extending the application to handle both online and offline text articles (in the form of PDF, Text or Word files).

The three categories can be extended to several more, to provide a finer distinction between articles.

In case of a document being assigned to multiple categories, a Ranking Classification approach can be explored.

This application can be extended into a Recommendation Engine which will simultaneously classify articles and also provide recommendation for similar type of articles to the user.

BIBLIOGRAPHY

1. Mitchell. *Web Scraping with Python*. O′Reilly Media; 1st edition (2015)
2. Bengfort, Bilbro, Ojeda. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning*. O′Reilly Media; 1st edition (2018)
3. Bird, Klein, Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media; 1st edition (2009)
4. Ingersoll, Morton, Farris. *Taming Text: How to Find, Organize, and Manipulate It.* Manning Publications; 1st edition (2013)
5. Miller. *Marketing Data Science - Modeling Techniques in Predictive Analytics with R and Python.* Pearson Education; 1st edition (2018)
6. https://towardsdatascience.com/
7. https://www.kdnuggets.com/
8. https://www.techopedia.com/
9. https://www.analyticsvidhya.com/
10. https://stackabuse.com/
11. https://medium.com/@bedigunjit/
12. https://machinelearningmastery.com/
13. https://www.coursera.org/
14. https://jakevdp.github.io/

REFERENCES

[1] Priyanka patil T. and S.I. Nipanikar. 2016. Survey on scene text detection and text recognition. *(IJARCCE) International Journal of Advanced Research in Computer and Communication Engineering*. ISSN (e):2278-1021, Vol.5, Issue 3.

[2] Kumuda T. and Basavaraj L. 2016. Hybrid approach to extract text in natural scene images. *International Journal of Computer Applications (IJOCA).*ISSN 0975-8887 vol.142, No.10, PP.18-22.

[3] Miss. Poonam B. Kadam, Mrs.Latika R. and Desai. 2014. A hybrid approach to detect and recognize texts in images. *(IOSRJEN), IOSR Journal of Engineering*. ISSN 2250-3021 Volume 04, Number 7 (July 2014), pp. 19-20.

[4] Navathe, shamkant B and Elmasri Ramez. 2000. Data mining and text mining in fundamental database system. *Pearson education pvt.inc*, Singapore, 841-872.

[5] Haralampos Karanikas and Manchester. 2005. Knowledge discovery in text and text mining software. *Center for Research in Information Management*.

[6] Franca Debole and Fabrizio Sebastiani. 2003. Supervised term weighting for automated text categorization. *ACM symposium on Applied computing - SAC '03, 2003*

[7] Takeru Miyato, Andrew M. Dai and Ian Goodfellow. 2016. Adversarial Training Methods for Semi-Supervised Text Classification. *International Conference on Learning Representations (ICLR) 2017*

[8] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. *15th Conference of the European Chapter of the Association for Computational Linguistics*: Volume 2, Short Papers

[9] Ghayda Altalib and Hind Salman. 2013. A Study on Analysis of SMS Classification Using TF-IDF Weighting. *International Journal of Computer Networks and Communications Security.*

[10] Waleed Zaghloul, Sang M. Lee, Silvana Trimi. 2009. Text classification: neural networks vs support vector machines. *Industrial Management & Data Systems.* ISSN: 0263-5577