To Predict whether a tweet "Acknowledges" or "Denies" Climate Change [Tweet Stance Detection] using Data Analytics and Machine Learning

A Project Report

submitted in partial fulfillment of the

requirements for the award of the degree of

Bachelor of Technology (B.Tech.)

Submitted by

KUSHAGRA SETH

[60096403115]

IT(I-9)

Under the supervision of

Dr. Amita Goel

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

SEC-22, ROHINI, DELHI –110086, INDIA

DEC, 2018

ACKNOWLDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to **Dr. Amita Goel** (Professor, IT) MAIT, Delhi for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing this project.

Secondly, I will like to thank my parents as well as my family members whose blessings and support always helped me face the challenges ahead.

I am highly indebted to **HOD**(IT) and other members of IT Department for their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my colleague **Sahil Singh Patwal** in developing the project and people who have willingly helped me out with their abilities.

KUSHAGRA SETH

(60096403115)

CANDIDATE’S DECLARATION

I hereby declare that the work presented in this project report titled, **“To Predict whether a tweet “Acknowledges” or “Denies” Climate Change [Tweet Stance Detection] using Data Analytics and Machine Learning”** is submitted by me in the partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology (B.Tech.), s**ubmitted in the Department of **Information Technology,** Maharaja Agrasen Institute of Technology is an authentic record of my project work carried out under the guidance of **Dr. Amita Goel**.

Date: 12/11/2018 **KUSHAGRA SETH**
Place: Delhi 60096403115

SUPERVISOR’S CERTIFICATE

It is to certify that the Project entitled **“To Predict whether a tweet “Acknowledges” or “Denies” Climate Change [Tweet Stance Detection] using Data Analytics and Machine Learning”** which is being submitted by **Mr. KUSHAGRA SETH** to the Maharaja Agrasen Institute of Technology(MAIT), Rohini in the fulfillment of the requirement for the award of the degree of **Bachelor of Technology (B.Tech.)**, is a record of bonafide project work carried out by him under my guidance and supervision. The matter presented in this project report has not been submitted either in part or full to any University or Institute for award of any degree.

Signature
(Dr. M.L. Sharma, HOD, IT)

Signature
(Dr. Amita Goel, Professor, IT)

ABSTRACT

We can often detect from a person's utterances whether he/she is in **favor** of or **against** a **given target entity** (a product, topic, another person, etc.). Here, we present a dataset of tweets where the tweeter is in favor of or against pre-chosen targets of interest—their stance. The targets of interest may or may not be referred to in the tweets, and they may or may not be the target of opinion in the tweets. The data pertains to tweets about climate change and global warming commonly known and debated in the United States.
Apart from stance, the tweets are also annotated for whether the target of interest is the target of opinion in the tweet. The annotations were performed by crowdsourcing. Several techniques were employed to encourage high-quality annotations (for example, providing clear and simple instructions) and to identify and discard poor annotations (for example, using a small set of check questions annotated by the authors). This Stance Dataset, which was subsequently also annotated for sentiment, can be used to better understand the relationship between stance, sentiment, entity relationships, and textual inference.

This is a project with the objective to develop a robust-classification model which can detect the stance of a tweeter and accordingly classify him into its correct class, i.e., either favor or against the given target entity. We will select the classifier which will produce the highest f1_score and highest macro average.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Stance detection is the task of automatically determining from text whether the author of the text is in **favor** of or **against** a proposition or target. The target may be a person, an organization, a government policy, a movement, a product, etc.

For example, one can infer from Barack Obama's speeches that he is in favor of stricter gun laws in the US. Similarly, people often express stance towards various target entities through posts on online forums, blogs, Twitter, YouTube, Instagram, etc. Automatically detecting stance has widespread applications in information retrieval, text summarization, and textual entailment. Over the last decade, there has been active research in modeling stance. The task we explore is detecting stance from tweets, and it is formulated as follows: given a tweet and a target entity (person, organization, movement, policy, etc.), can automatic natural language systems determine whether the tweeter is in favor of the given target or against the given target?

For example, consider the target–tweet pair:

Target: legalization of abortion

Tweet: A foetus has rights too! Make your voice heard. #StopAbortions

Humans can deduce from the tweet that the tweeter is likely against the target (the tweeter's stance is against the legalization of abortion). Our goal is to create labeled training and test data that can be used in the developments of automatic systems for detecting stance.

To successfully detect stance, automatic systems often have to rely on world knowledge that may not be explicitly stated in the focus text. For example, systems benefit from knowing that if one is actively supporting foetus rights, then he or she is likely against the right to abortion. This world knowledge may be acquired from large text corpora. Thus, for each target, we also acquire a corpus of unlabeled tweets that include hashtags related to the target. We will refer to this set of tweets as the domain corpus for the target. Automatic systems can gather information from the domain corpus to help with the detection of stance—for example, by identifying how entities are related. Stance detection is related to sentiment analysis, but the two have significant differences. In sentiment analysis, systems determine whether a piece of text is positive, negative. However, in stance detection, systems are to determine favorability towards a given target of interest—and the target may not be explicitly mentioned in the text.

For example, consider the target–text pair below:

Target: Donald Trump

Tweet: Jeb Bush is the only sane candidate in this republican lineup. #PresedentialElections2k17

The target of opinion in the tweet is Jeb Bush, but the given stance target is Donald Trump. The tweet expresses positive opinion towards Jeb Bush, from which we can infer that the tweeter is likely to be unfavorable towards Donald Trump. Note that it is possible that one can be in favor of Jeb Bush and yet also be in favor of Donald Trump.

However, the goal in stance detection, is to determine which is more probable: that the author is in favor of, against, or neutral towards the target. In this case, most annotators will agree that the tweeter is likely against Donald Trump. To aid further analysis, the tweets in the Stance Dataset are also annotated for whether target of interest is the target of opinion in the tweet.

CHAPTER 2

Creating the Dataset for Stance in Tweets

In order to create a suitable dataset of tweet–target pairs annotated for stance, we first identified useful properties for such a dataset, then selected tweet–target pairs in a manner that is consistent with those properties, and finally annotated the tweet–target pairs.

Properties of a Good Stance-Labeled Dataset: We wanted to create a dataset of stance-labeled tweet–target pairs that had the following properties:

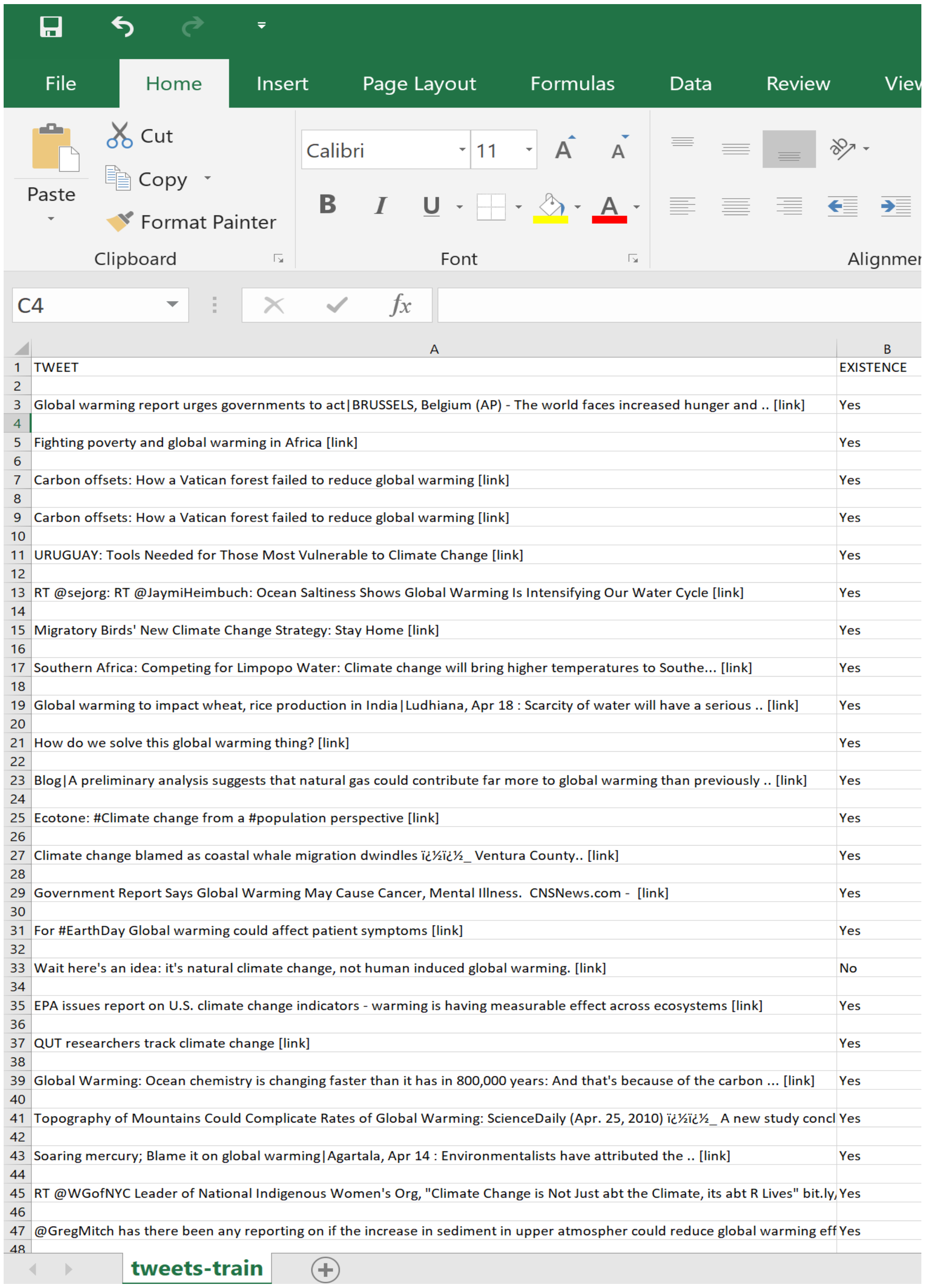
(1.) The tweet and target are commonly understood by a wide number of people in the United States. This is important because the tweet-target pairs will later be annotated by English speakers.

(2.) There must be a significant amount of data for each of the two classes: favor or against. Often, the proportion of tweets in favor of a target may not be similarly numerous as those against it.
However, we did not want scenarios where there are no tweets in favor of a target or no tweets against it. Also, the total number of tweet–target pairs from which the stance cannot be inferred ('neither' instances) can be very large. However, creating a dataset where 99% of the tweets are from this category makes the dataset less interesting and less useful. So, we down-sample the number of 'neither' instances.

(3.) Apart from tweets that explicitly mention the target, the dataset should include a significant number of tweets that express opinion towards the target without referring to it by name. We wanted to include the relatively harder cases for stance detection where the target is referred to in indirect ways such as through pronouns, epithets, honorifics, and relationships.

(4.) Apart from tweets that express opinion towards the target, the dataset should include a significant number of tweets in which the target of opinion is different from the given stance target. As mentioned earlier with the Donald Trump example, sometimes stance towards a target can be inferred even if that target is not the target of opinion in the text. Including such instances makes the task more challenging. Downstream applications often require stance towards particular pre-chosen targets, and having data where the target of opinion is different from the target of stance helps test how well stance detection systems can cope with such instances. These properties influenced various choices in how our dataset was created.

DATASET: https://data.world/xprizeai-env/sentiment-of-climate-change



	A	B
1	TWEET	EXISTENCE
2		
3	Global warming report urges governments to act|BRUSSELS, Belgium (AP) - The world faces increased hunger and .. [link]	Yes
4		
5	Fighting poverty and global warming in Africa [link]	Yes
6		
7	Carbon offsets: How a Vatican forest failed to reduce global warming [link]	Yes
8		
9	Carbon offsets: How a Vatican forest failed to reduce global warming [link]	Yes
10		
11	URUGUAY: Tools Needed for Those Most Vulnerable to Climate Change [link]	Yes
12		
13	RT @sejorg: RT @JaymiHeimbuch: Ocean Saltiness Shows Global Warming Is Intensifying Our Water Cycle [link]	Yes
14		
15	Migratory Birds' New Climate Change Strategy: Stay Home [link]	Yes
16		
17	Southern Africa: Competing for Limpopo Water: Climate change will bring higher temperatures to Southe... [link]	Yes
18		
19	Global warming to impact wheat, rice production in India|Ludhiana, Apr 18 : Scarcity of water will have a serious .. [link]	Yes
20		
21	How do we solve this global warming thing? [link]	Yes
22		
23	Blog|A preliminary analysis suggests that natural gas could contribute far more to global warming than previously .. [link]	Yes
24		
25	Ecotone: #Climate change from a #population perspective [link]	Yes
26		
27	Climate change blamed as coastal whale migration dwindles ï¿½ï¿½_ Ventura County.. [link]	Yes
28		
29	Government Report Says Global Warming May Cause Cancer, Mental Illness. CNSNews.com - [link]	Yes
30		
31	For #EarthDay Global warming could affect patient symptoms [link]	Yes
32		
33	Wait here's an idea: it's natural climate change, not human induced global warming. [link]	No
34		
35	EPA issues report on U.S. climate change indicators - warming is having measurable effect across ecosystems [link]	Yes
36		
37	QUT researchers track climate change [link]	Yes
38		
39	Global Warming: Ocean chemistry is changing faster than it has in 800,000 years: And that's because of the carbon ... [link]	Yes
40		
41	Topography of Mountains Could Complicate Rates of Global Warming: ScienceDaily (Apr. 25, 2010) ï¿½ï¿½_ A new study concl	Yes
42		
43	Soaring mercury; Blame it on global warming|Agartala, Apr 14 : Environmentalists have attributed the .. [link]	Yes
44		
45	RT @WGofNYC Leader of National Indigenous Women's Org, "Climate Change is Not Just abt the Climate, its abt R Lives" bit.ly,	Yes
46		
47	@GregMitch has there been any reporting on if the increase in sediment in upper atmospher could reduce global warming eff	Yes
48		

CHAPTER 3

Natural Language Processing [NLP]

NLP refers to the computer understanding and manipulation of human language, i.e., parsing the human language data and extracting meaningful information out of it. Large amounts of data is generated every hour, every day of every year which is used for various types of analysis like Market Segmentation, Clustering, Emotion Prediction, Sentiment Analysis, Stance Detection etc.

Sources of Data: Twitter, Facebook, YouTube, Reddit, Online Transactions, Online User History etc.

The real reason why NLP is hard: The process of reading and understanding language is far more complex than it seems at first glance. There are many things that go in to *truly* understanding what a piece of text means in the real-world.

For example, what do you think the following piece of text means?

"Steph Curry was on fire last nice. He totally destroyed the other team"

To a human it's probably quite obvious what this sentence means. We know Steph Curry is a basketball player; or even if you don't, we know that he plays on some kind of team, probably a sports team. When we see "on fire" and "destroyed" we know that it means Steph Curry played really well last night and beat the other team.

Computers tend to take things a bit too literally. Viewing things literally like a computer, we would see “Steph Curry” and based on the capitalization assume it’s a person, place, or otherwise important thing which is great! But then we see that Steph Curry “was on fire” …. A computer might tell you that someone literally lit Steph Curry on fire yesterday! After that, the computer might say that Mr. Curry has physically destroyed the other team, i.e., they no longer exist according to this computer.

Important Python Libraries for NLP: sklearn (Scikit-learn), nltk (Natural Language Toolkit) and imblearn (Toolbox for Imbalanced Datasets)

CHAPTER 4

Pre-Processing the Dataset and performing Tweet Normalization

- **Stopword Removal**

  Text may contain many stop words like ‘the’, ‘is’, ‘are’, ‘that’ etc. which are common across all the documents in the corpus and carry no meaning attached to them. So, these words are filtered as we do not want these words taking up memory space and take up processing time. As such there is no universal list of stop words but “nltk” module in python has a list of stop words in 16 different languages which can be removed from our dataset.

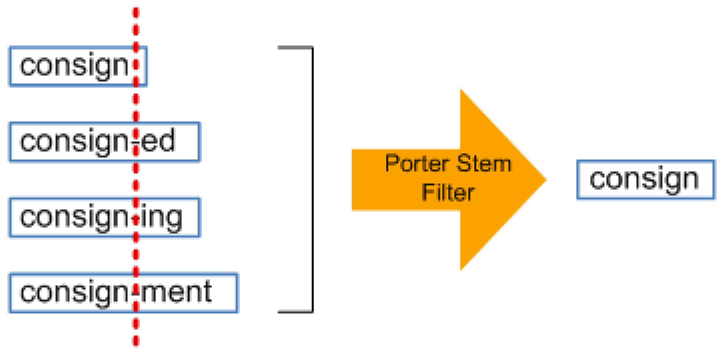
  Code Snippet:

```
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
sentences = nltk.sent_tokenize(paragraph)
paragraph = “ ………………… “
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    newWords = [word for word in words if word not in stopwords.words("english")]
    sentences[i] = " ".join(newWords)
```

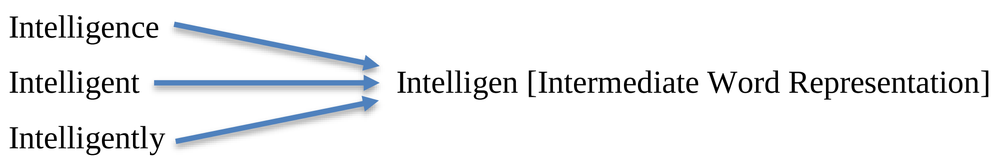
Text with Stopwords	Text without Stopwords
• Can listening be exhausting? • I like reading, so I read,	• listening exhausting? • like reading read.

- **Lemmatization and Stemming**

Stemming: It is the process of reducing inflected or derived word to a common base form (root form).



[Intermediate Word Representation]



Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Common Stemmers used: Lovins Stemmer, Porter Stemmer, Paice Stemmer

Problem with Stemming: The produced word representation/root form may or may not have any meaning. E.g.: intelligen, fina etc.

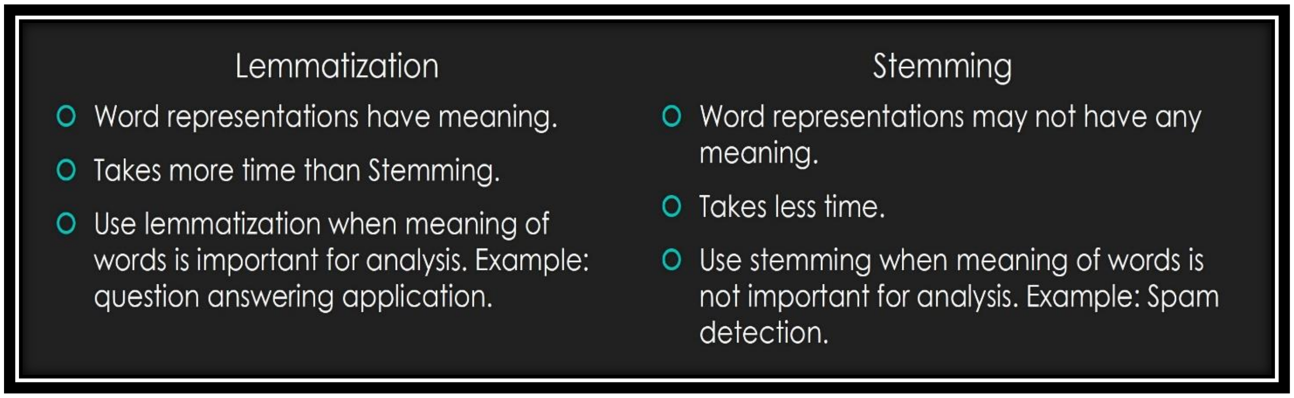
Solution to Stemming:

Lemmatization: It is same as Stemming but the word representation/root form has a meaning.

It usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the ***lemma***. If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.

$$\begin{tikzcd}
\text{Intelligence} \arrow[dr] & \\
\text{Intelligent} \arrow[r] & \text{Intelligent [Intermediate Word Representation]} \\
\text{Intelligently} \arrow[ur] &
\end{tikzcd}$$

<u>Difference between Lemmatization and Stemming:</u>



Lemmatization	Stemming
Word representations have meaning.	Word representations may not have any meaning.
Takes more time than Stemming.	Takes less time.
Use lemmatization when meaning of words is important for analysis. Example: question answering application.	Use stemming when meaning of words is not important for analysis. Example: Spam detection.

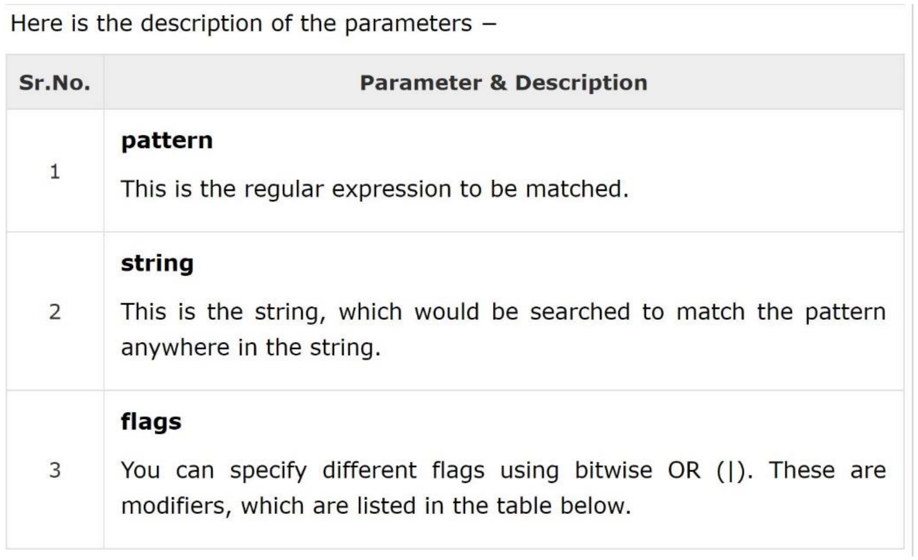
Example: "Better" has "good" as its lemma, so its missed by stemming but matched by lemmatization as it requires dictionary look-up.
"Walking" has "walk" as its lemma, so its matched by both stemming and lemmatization

- ## Pre-processing using Regular Expression (Reg-Ex):

Reg-Ex is sequence of characters that refines a specific search pattern using which you can substitute patterns inside text with least amount of code.
In simple words, Reg-Ex is used for detecting word patterns and substituting patterns inside text.
The module **re** provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

Commonly used Reg-Ex functions of re module:

- **The *match* Function**: re.match(pattern, string, flags=0)
  This function attempts to match RE *pattern* to *string* with optional *flags*.
- **The search Function:** re.search(pattern, string, flags=0)
  This function searches for first occurrence of RE *pattern* within *string* with optional *flags*.
- **Search and Replace:** re.sub(pattern, repl, string, max=0)
  This method replaces all occurrences of the RE *pattern* in *string* with *repl*, substituting all occurrences unless *max* provided. This method returns modified string.


Here is the description of the parameters –

Sr.No.	Parameter & Description
1	**pattern** This is the regular expression to be matched.
2	**string** This is the string, which would be searched to match the pattern anywhere in the string.
3	**flags** You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

Code Snippet:

```
sentence = "1996 was the year when i was born"
re.match(r"[a-zA-Z]+", sentence) # no o/p
re.search(r"[a-zA-Z]+", sentence) # was
#Starts with
if re.search(r"^1996", sentence):
        print("Match")
else:
        print("No Match")
```

Use of Reg-Ex to pre-process the dataset:

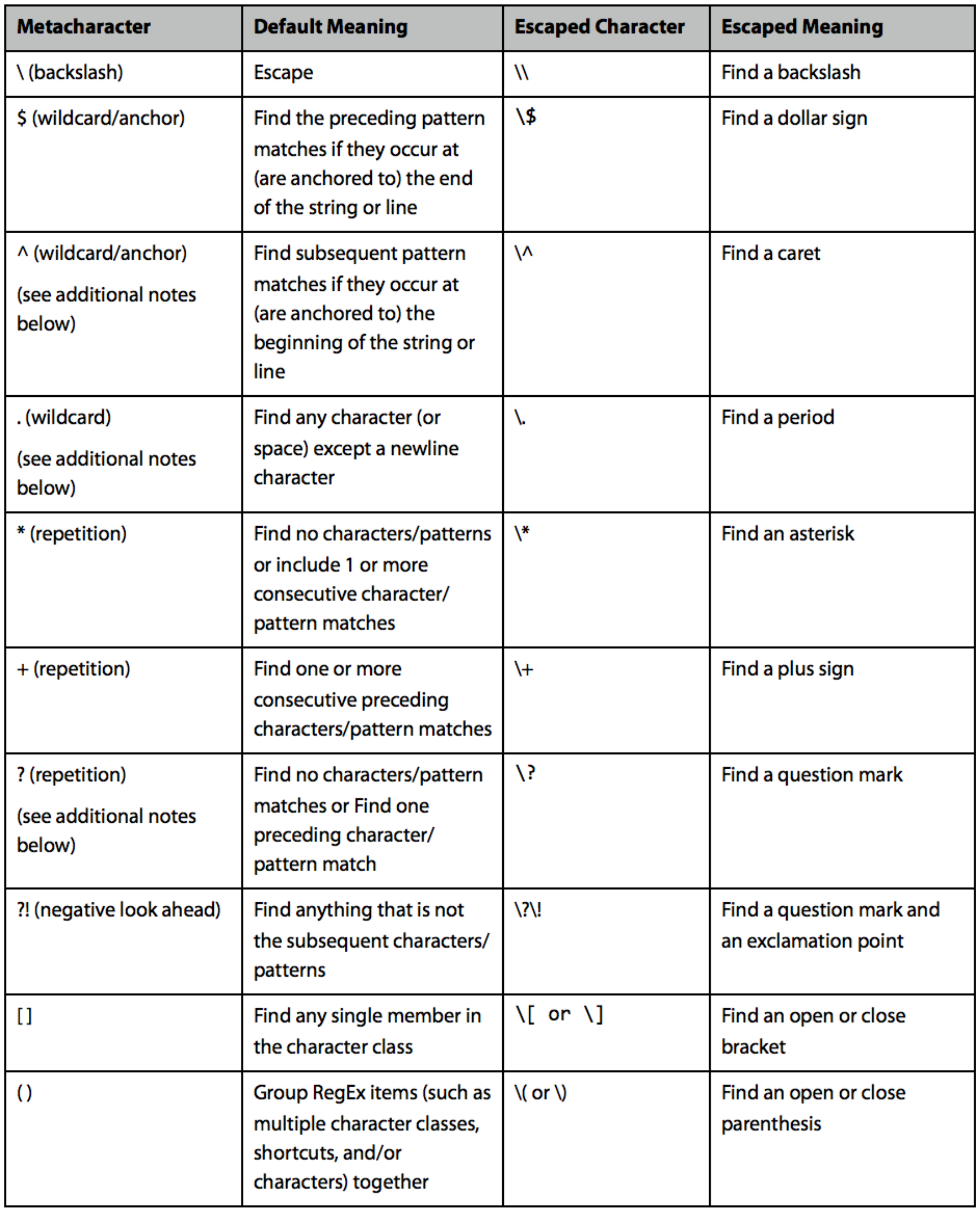
-Removing non-word characters

-Removing single words from the corpus

-Substituting multi-spaces by a single space in the corpus

-Tweet Normalization

Reg-Ex basic Metacharacters:



Metacharacter	Default Meaning	Escaped Character	Escaped Meaning
\ (backslash)	Escape	\\	Find a backslash
$ (wildcard/anchor)	Find the preceding pattern matches if they occur at (are anchored to) the end of the string or line	\$	Find a dollar sign
^ (wildcard/anchor) (see additional notes below)	Find subsequent pattern matches if they occur at (are anchored to) the beginning of the string or line	\^	Find a caret
. (wildcard) (see additional notes below)	Find any character (or space) except a newline character	\.	Find a period
* (repetition)	Find no characters/patterns or include 1 or more consecutive character/ pattern matches	*	Find an asterisk
+ (repetition)	Find one or more consecutive preceding characters/pattern matches	\+	Find a plus sign
? (repetition) (see additional notes below)	Find no characters/pattern matches or Find one preceding character/ pattern match	\?	Find a question mark
?! (negative look ahead)	Find anything that is not the subsequent characters/ patterns	\?\!	Find a question mark and an exclamation point
[ ]	Find any single member in the character class	\[or \]	Find an open or close bracket
( )	Group RegEx items (such as multiple character classes, shortcuts, and/or characters) together	\(or \)	Find an open or close parenthesis

CHAPTER 5

Creating the Feature Matrix using Feature Extraction and Textual Data Representation

- ### Bag-of-Words Model

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.

Problem with modeling text: It it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs.
Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.
In language processing, the vectors x are derived from textual data, in order to reflect various linguistic properties of the text.
This is called **feature extraction** or **feature encoding**.
A popular and simple method of feature extraction with text data is called the bag-of-words model of text.

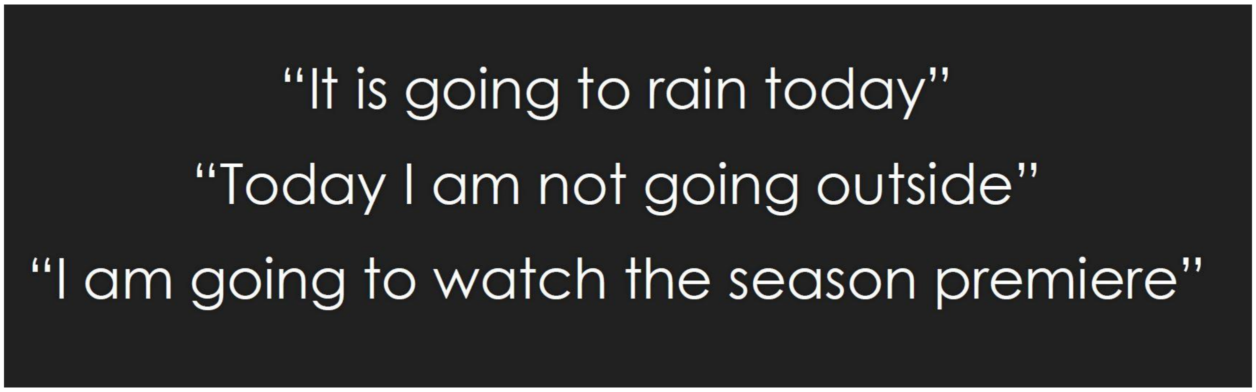
A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.
The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.
A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A vocabulary of known words.
2. 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.

A very common feature extraction procedures for sentences and documents is the bag-of-words approach (BOW). In this approach, we look at the histogram of the words within the text, i.e. considering each word count as a feature.

Step 1: Collect Data

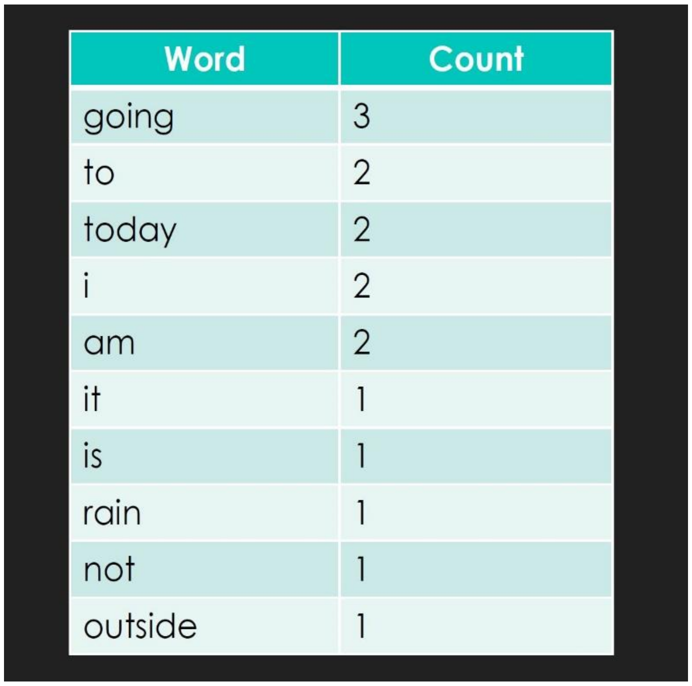


"It is going to rain today"

"Today I am not going outside"

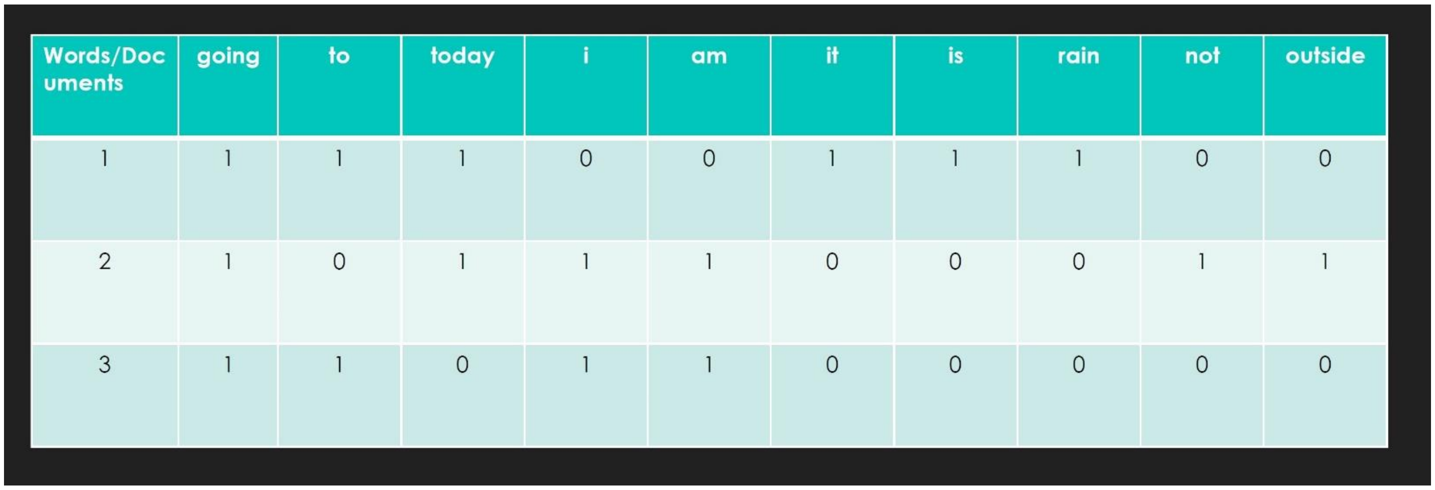
"I am going to watch the season premiere"

Step 2: Design the Vocabulary



Word	Count
going	3
to	2
today	2
i	2
am	2
it	1
is	1
rain	1
not	1
outside	1

Step 3: Create Document Vectors



Words/Documents	going	to	today	i	am	it	is	rain	not	outside
1	1	1	1	0	0	1	1	1	0	0
2	1	0	1	1	1	0	0	0	1	1
3	1	1	0	1	1	0	0	0	0	0

Limitations of Bag-of-Words

The bag-of-words model is very simple to understand and implement and offers a lot of flexibility for customization on your specific text data.

It has been used with great success on prediction problems like language modeling and documentation classification.

Nevertheless, it suffers from some shortcomings, such as:

- **Vocabulary**: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
- **Sparsity**: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
- **Meaning**: Discarding word order ignores the context, and in turn meaning of words in the document (semantics). Context and meaning can offer a lot to the model, that if modeled could tell the difference between the same words differently arranged (“this is interesting” vs “is this interesting”), synonyms (“old bike” vs “used bike”), and much more.

- ***TF-IDF* Model**

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

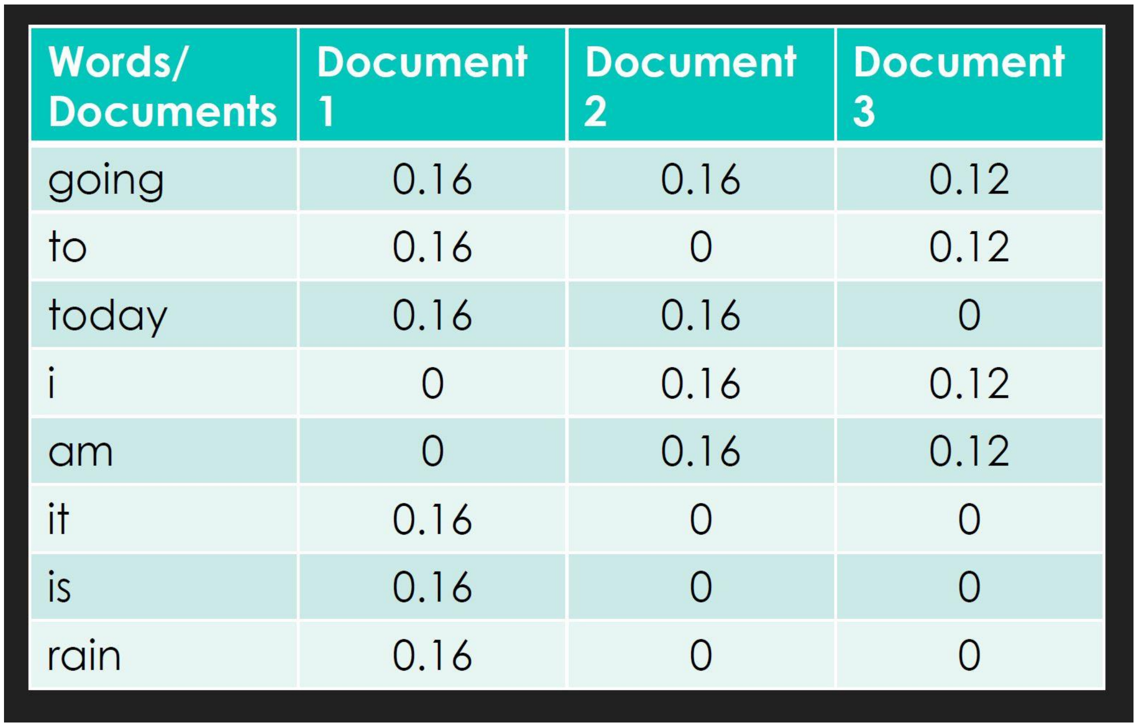
- **Term Frequency**: is a scoring of the frequency of the word in the current document.
- **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting.

The scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

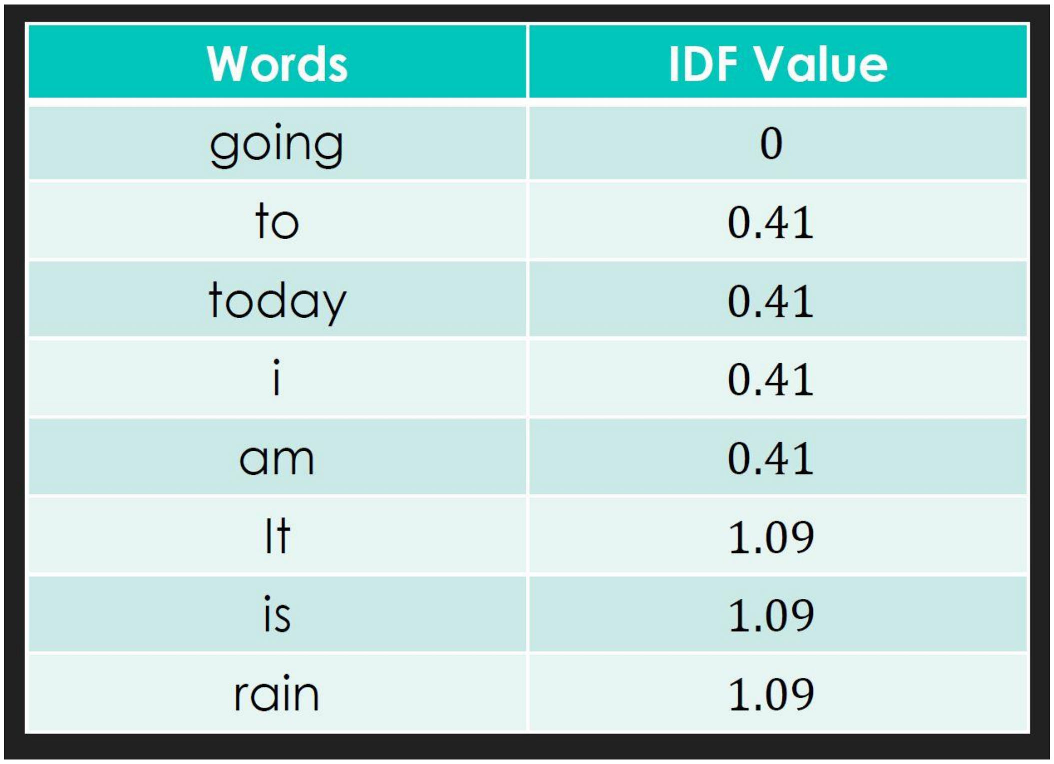
Thus, the idf of a rare term is high, whereas the idf of a frequent term is likely to be low.

Step 1: TF =
$$\frac{(Number\ of\ occurrences\ of\ a\ word\ in\ a\ document)}{(Number\ of\ words\ in\ that\ document)}$$



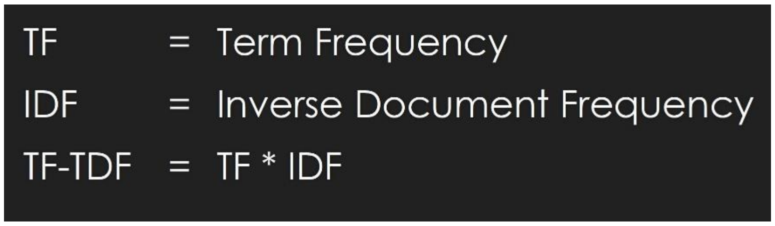
Words/ Documents	Document 1	Document 2	Document 3
going	0.16	0.16	0.12
to	0.16	0	0.12
today	0.16	0.16	0
i	0	0.16	0.12
am	0	0.16	0.12
it	0.16	0	0
is	0.16	0	0
rain	0.16	0	0

Step 2: IDF = $$log(\frac{(Number\ of\ documents)}{(Number\ of\ documents\ containing\ word)})$$

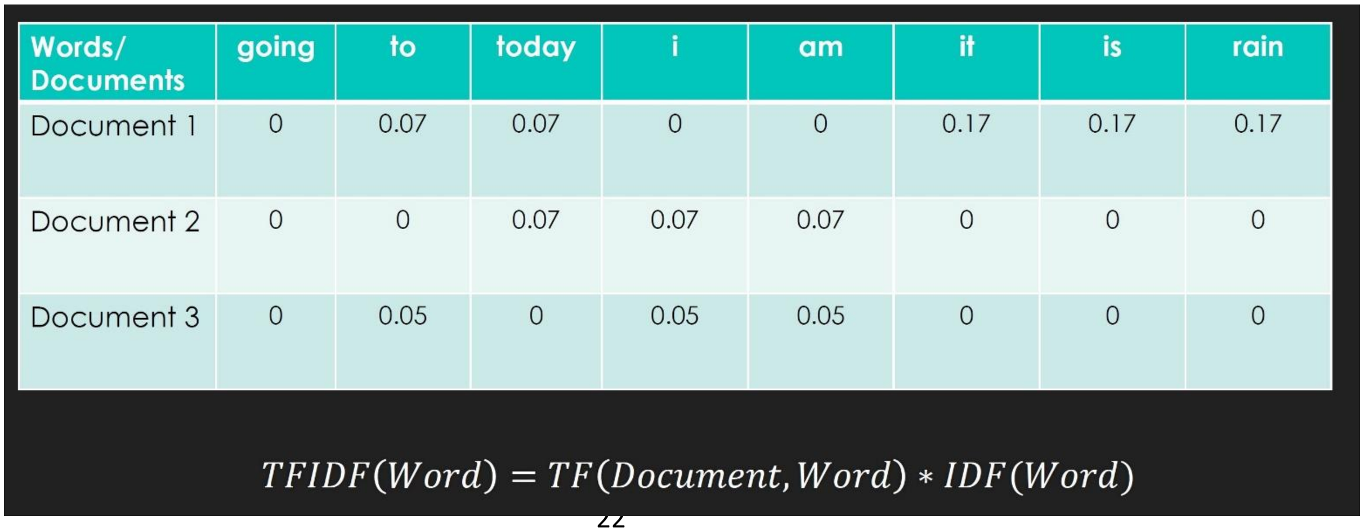


Words	IDF Value
going	0
to	0.41
today	0.41
i	0.41
am	0.41
It	1.09
is	1.09
rain	1.09

Step 3: TF-IDF



TF	=	Term Frequency
IDF	=	Inverse Document Frequency
TF-TDF	=	TF * IDF



Words/ Documents	going	to	today	i	am	it	is	rain
Document 1	0	0.07	0.07	0	0	0.17	0.17	0.17
Document 2	0	0	0.07	0.07	0.07	0	0	0
Document 3	0	0.05	0	0.05	0.05	0	0	0

$$TFIDF(Word) = TF(Document, Word) * IDF(Word)$$

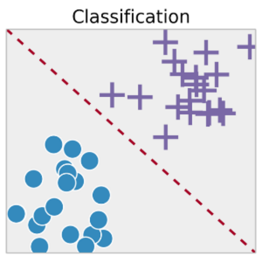
CHAPTER 6

Applying Classifiers to the Feature Matrix

Classification Models in Machine Learning are used to predict a category/class of an input data point.

It maps the i/p to a discrete class label, i.e., it predicts the class/category to which an i/p value will belong. For e.g.: An e-mail can be classified into 2 classes: “spam” or “not spam”.

In other words, it analyzes a dataset having one or more independent variables(x) to predict the class of the i/p, i.e., a binary outcome (0 or1).

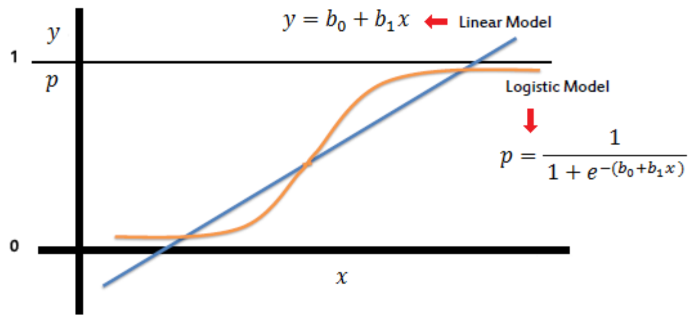


There is a wide variety of classification applications from medicine to marketing.

Different types of classifiers used on the feature matrix of the dataset:

- ***Logistic Regression***

  It predicts the probability of an outcome that can only have two values (i.e. a dichotomy). The prediction is based on the use of one or several predictors (numerical and categorical). A linear regression is not appropriate for predicting the value of a binary variable for two reasons:
  A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)
  Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.
  On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1.
  Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the “odds” of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.



In the logistic regression the constant (b0) moves the curve left and right and the slope (b1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

$$\frac{p}{1-p} = \exp\left(b_0 + b_1 x\right)$$

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b1) is the amount the logit (log-odds) changes with a one unit change in x.

$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

- ***Support Vector Machine (SVM)***

Maximal-Margin Classifier

The Maximal-Margin Classifier is a hypothetical classifier that best explains how SVM works in practice.

The numeric input variables (x) in your data (the columns) form an n-dimensional space. For example, if you had two input variables, this would form a two-dimensional space.

A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions you can visualize this as a line and let’s assume that all of our input points can be completely separated by this line. For example:

B0 + (B1 * X1) + (B2 * X2) = 0

Where the coefficients (B1 and B2) that determine the slope of the line and the intercept (B0) are found by the learning algorithm, and X1 and X2 are the two input variables.

You can make classifications using this line. By plugging in input values into the line equation, you can calculate whether a new point is above or below the line.

Above the line, the equation returns a value greater than 0 and the point belongs to the first class (class 0).
Below the line, the equation returns a value less than 0 and the point belongs to the second class (class 1).
A value close to the line returns a value close to zero and the point may be difficult to classify.
If the magnitude of the value is large, the model may have more confidence in the prediction.
The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that as the largest margin. This is called the Maximal-Margin hyperplane.

The margin is calculated as the perpendicular distance from the line to only the closest points. Only these points are relevant in defining the line and in the construction of the classifier. These points are called the support vectors. They support or define the hyperplane.

The hyperplane is learned from training data using an optimization procedure that maximizes the margin.

Soft Margin Classifier

In practice, real data is messy and cannot be separated perfectly with a hyperplane.

The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the soft margin classifier. This change allows some points in the training data to violate the separating line.

An additional set of coefficients are introduced that give the margin wiggle room in each dimension. These coefficients are sometimes called slack variables. This increases the complexity of the model as there are more parameters for the model to fit to the data to provide this complexity.

A tuning parameter is introduced called simply C that defines the magnitude of the wiggle allowed across all dimensions. The C parameters defines the amount of violation of the margin allowed. A C=0 is no violation and we are back to the inflexible Maximal-Margin Classifier described above. The larger the value of C the more violations of the hyperplane are permitted.

During the learning of the hyperplane from data, all training instances that lie within the distance of the margin will affect the placement of the hyperplane and are referred to as support vectors. And as C affects the number of instances that are allowed to fall within the margin, C influences the number of support vectors used by the model.

The smaller the value of C, the more sensitive the algorithm is to the training data (higher variance and lower bias).
The larger the value of C, the less sensitive the algorithm is to the training data (lower variance and higher bias).

Support Vector Machines (Kernels)

The SVM algorithm is implemented in practice using a kernel.

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra, which is out of the scope of this introduction to SVM.

A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. The inner product between two vectors is the sum of the multiplication of each pair of input values.

For example, the inner product of the vectors [2, 3] and [5, 6] is 2*5 + 3*6 or 28.

The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B0 + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

Linear Kernel SVM

The dot-product is called the kernel and can be re-written as:

K(x, xi) = sum(x * xi)

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

Other kernels can be used that transform the input space into higher dimensions such as a Polynomial Kernel and a Radial Kernel. This is called the Kernel Trick.

It is desirable to use more complex kernels as it allows lines to separate the classes that are curved or even more complex. This in turn can lead to more accurate classifiers.

Polynomial Kernel SVM
Instead of the dot-product, we can use a polynomial kernel, for example:

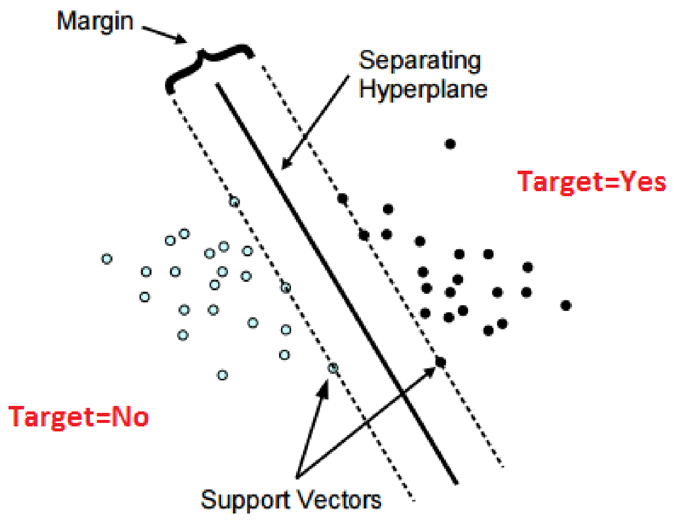
K(x,xi) = 1 + sum(x * xi)^d

Where the degree of the polynomial must be specified by hand to the learning algorithm. When d=1 this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space.

Radial Kernel SVM
Finally, we can also have a more complex radial kernel. For example:

K(x,xi) = exp(-gamma * sum((x – xi^2))

Where gamma is a parameter that must be specified to the learning algorithm. A good default value for gamma is 0.1, where gamma is often 0 < gamma < 1. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.



Support Vector Machine

- ***Decision Tree Classification***

Decision Trees

Classification and Regression Trees or CART for short is a term introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.

Classically, this algorithm is referred to as “decision trees”, but on some platforms like R they are referred to by the more modern term CART.

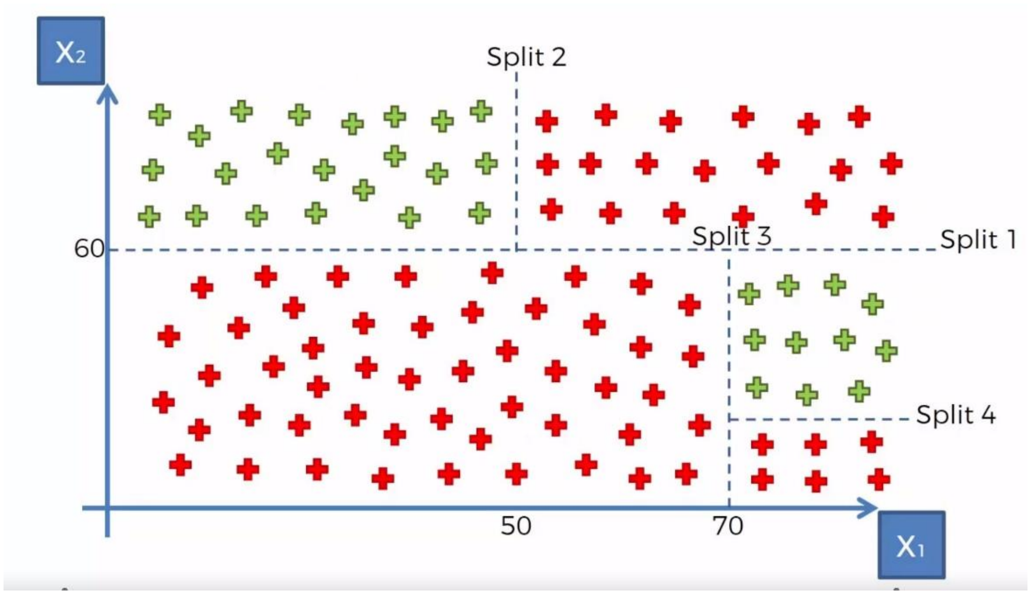
The CART algorithm provides a foundation for important algorithms like bagged decision trees, random forest and boosted decision trees.

CART Model Representation

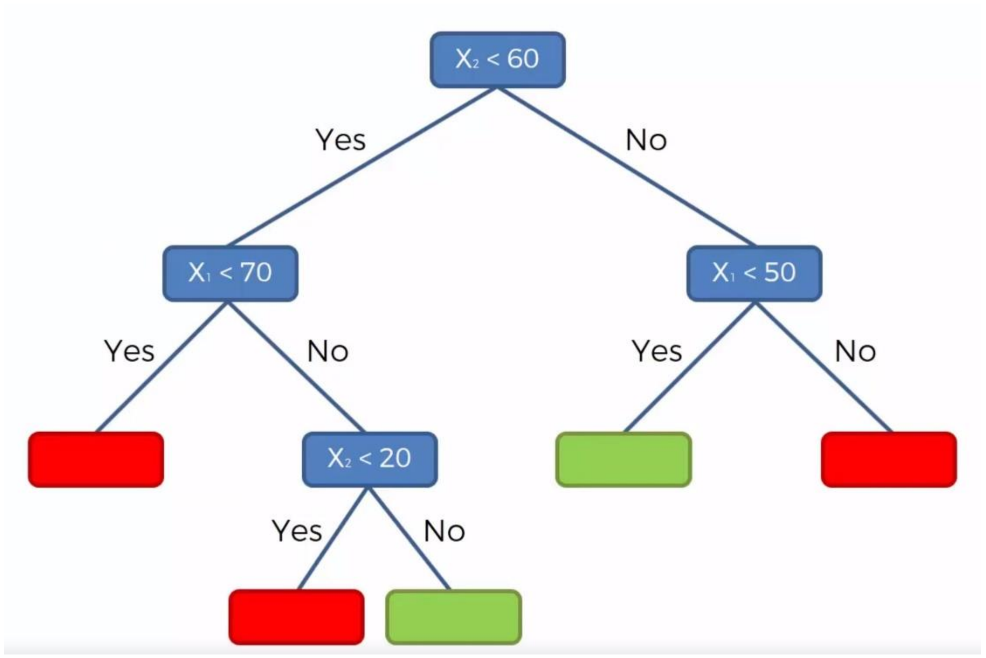
The representation for the CART model is a binary tree.

This is your binary tree from algorithms and data structures, nothing too fancy. Each root node represents a single input variable (x) and a split point on that variable (assuming the variable is numeric).

The leaf nodes of the tree contain an output variable (y) which is used to make a prediction.



Decision Tree:

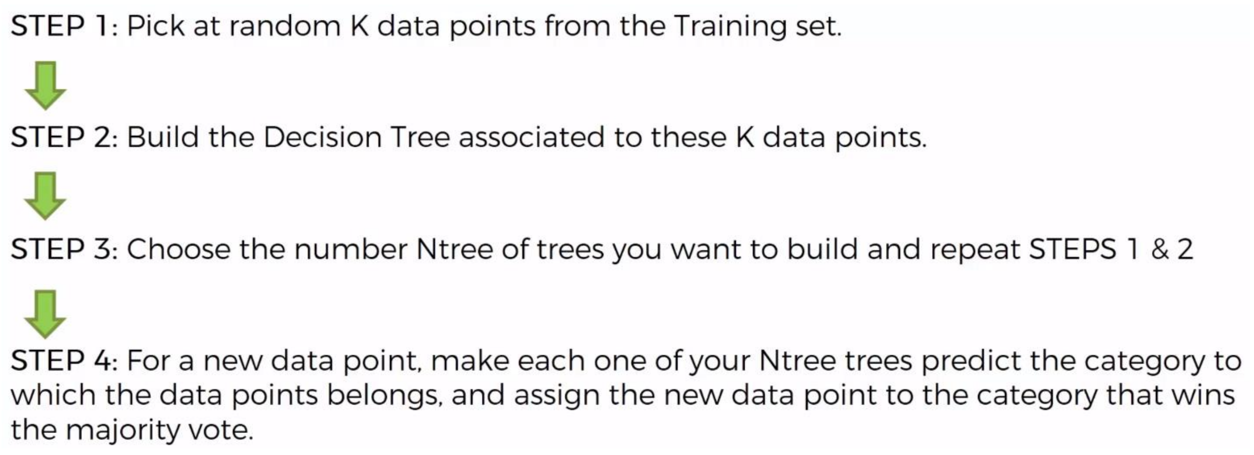


- ***Random Forest Classification***

- Decision trees involve the greedy selection of the best split point from the dataset at each step.
- Random Forest is an ensemble of Decision Trees.
- This algorithm makes decision trees susceptible to high variance if they are not pruned. This high variance can be harnessed and reduced by creating multiple trees with different samples of the training dataset (different views of the problem) and combining their predictions. This approach is called bootstrap aggregation or bagging for short.
- A limitation of bagging is that the same greedy algorithm is used to create each tree, meaning that it is likely that the same or very similar split points will be chosen in each tree making the different trees very similar (trees will be correlated). This, in turn, makes their predictions similar, mitigating the variance originally sought.
- We can force the decision trees to be different by limiting the features (rows) that the greedy algorithm can evaluate at each split point when creating the tree. This is called the Random Forest algorithm.
- Like bagging, multiple samples of the training dataset are taken and a different tree trained on each. The difference is that at each point a split is made in the data and added to the tree, only a fixed subset of attributes can be considered.

- For classification problems, the type of problems we will look at in this tutorial, the number of attributes to be considered for the split is limited to the square root of the number of input features.
- The result of this one small change are trees that are more different from each other (uncorrelated) resulting predictions that are more diverse and a combined prediction that often has better performance that single tree or bagging alone.

STEPS:

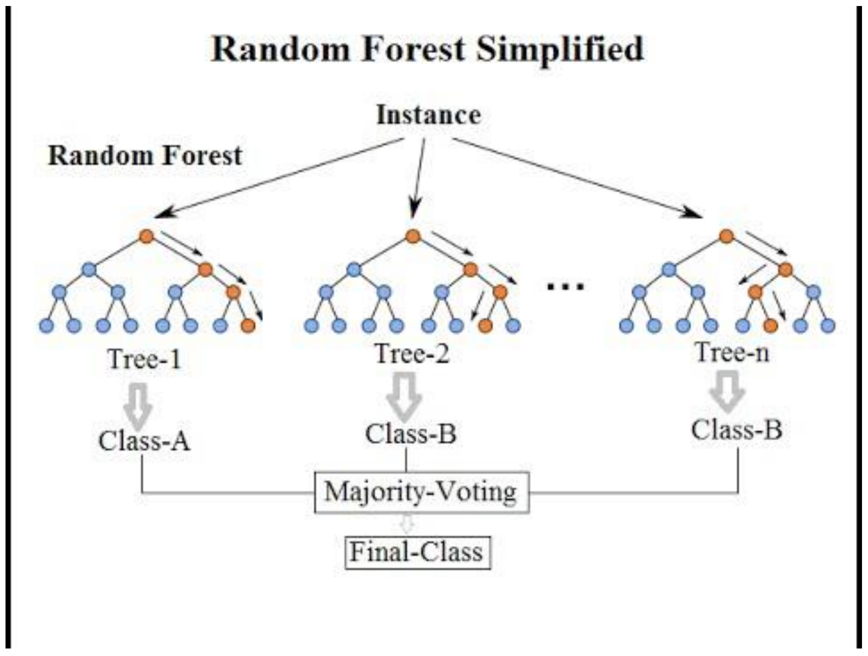


STEP 1: Pick at random K data points from the Training set.

STEP 2: Build the Decision Tree associated to these K data points.

STEP 3: Choose the number Ntree of trees you want to build and repeat STEPS 1 & 2

STEP 4: For a new data point, make each one of your Ntree trees predict the category to which the data points belongs, and assign the new data point to the category that wins the majority vote.



CHAPTER 7

Boosting Classification Algorithms

• *Ensemble Voting Classifier*

The three most popular methods for combining the predictions from different models are:

- **Bagging:** Building multiple models (typically of the same type) from different subsamples of the training dataset.
- **Boosting:** Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.
- **Voting:** Building multiple models (typically of differing types) and simple statistics (like calculating the mean) are used to combine predictions.

This post will not explain each of these methods.

It assumes you are generally familiar with machine learning algorithms and ensemble methods and that you are looking for information on how to create ensembles in Python.

Voting Ensemble

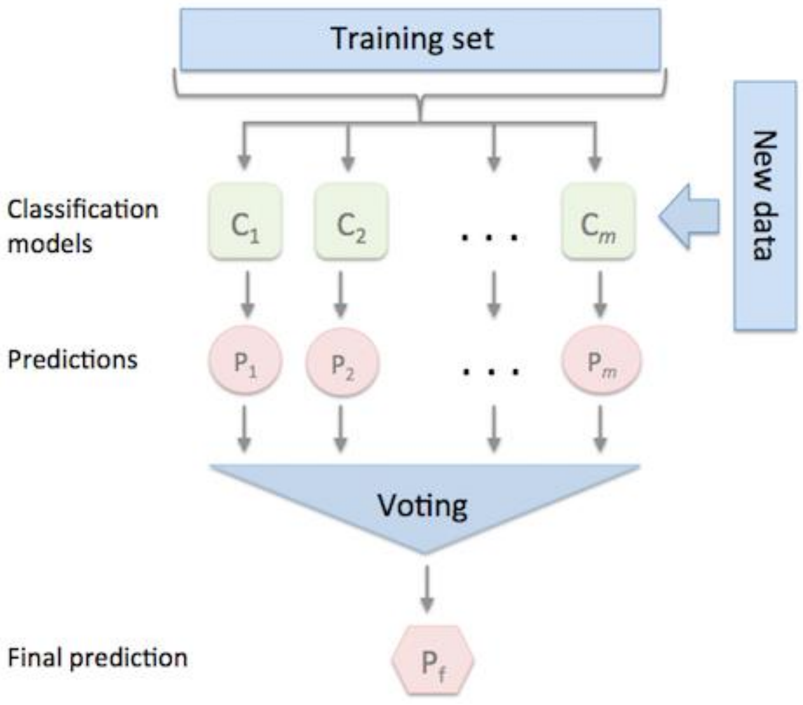
Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data.

The predictions of the sub-models can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from submodels, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn.

The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification problem.

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-
diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```



CHAPTER 8

Code Implementation

- ***Without Normalization***

```
# Logistic Regression Classifier [Un-Normalized]

# Importing reqd. Libraries
import numpy as np
import pandas as pd
import re
import pickle
import nltk
from nltk.corpus import stopwords
from pandas import read_csv
nltk.download("stopwords")

def write_lines(outfile,lists):
    f = open(outfile, "a", encoding='utf-8')
    for lines in lists:
        lines=str(lines)+"\n"
        f.writelines(str(lines))



    f.close()

outfile = "results.txt"
lists = []



# Importing and Cleaning the dataset
dataset = read_csv("tweets-train.csv")
X = dataset.iloc[:, [0]].values
X = np.ndarray.tolist(X)
y = dataset.iloc[:, [1]].values
y = np.ndarray.tolist(y)
y_corpus =[]
```

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(y)
y = le.transform(y)
sum = np.sum(y)

# Pickling the Dataset [Persisting the dataset]
with open("X.pickle", "wb") as f:
    pickle.dump(X, f)
with open("y.pickle", "wb") as f:
    pickle.dump(y, f)

# Pre-processing the dataset, Normalizing the Tweets and Creating the corpus
corpus = []
for i in range(len(X)):
    origTweet = str(X[i])
    corpus.append(origTweet)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
text_Train, text_Test, sd_Train, sd_Test = train_test_split(corpus, y, test_size=0.25,
random_state=0)

# Creating TF-IDF Model using TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# tVectorizer = TfidfVectorizer(max_features=1000, min_df=3, max_df=0.6,
stop_words=stopwords.words("english"))
# tVectorizer = TfidfVectorizer(max_features=1000, min_df=3, max_df=0.6)
tVectorizer = TfidfVectorizer(ngram_range = (1, 3))
# tVectorizer = CountVectorizer(max_features=1000, min_df=3, max_df=0.6,
stop_words=stopwords.words("english"))
# tVectorizer = CountVectorizer(max_features=1000, min_df=3, max_df=0.6)
# tVectorizer = CountVectorizer()
text_Train = tVectorizer.fit_transform(text_Train)

# Random Over-sampling of the majority class
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
text_Train, sd_Train = ros.fit_sample(text_Train, sd_Train)
```

```
text_Test = tVectorizer.transform(text_Test)

# Fitting classifier to the Training set &  Predicting the Test set results
from sklearn.linear_model import LogisticRegression

# Logistic Regression
clf1 = LogisticRegression(random_state=0)
clf1.fit(text_Train, sd_Train)
sd_Pred1 = clf1.predict(text_Test)


# Visualizing the RESULTS

# Making the Confusion Matrix(Classification Evaluation Metric)
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(sd_Test, sd_Pred1)


# Generating the Classification Report(Classification Evaluation Metric)
from sklearn.metrics import classification_report
report_lr = classification_report(sd_Test, sd_Pred1)

# Area Under ROC Curve
import matplotlib.pyplot as plt
from sklearn import metrics
y_pred_proba = clf1.predict_proba(text_Test)[::,1]
fpr, tpr, _ = metrics.roc_curve(sd_Test,  y_pred_proba)
auc = metrics.roc_auc_score(sd_Test, y_pred_proba)
plt.plot(fpr,tpr,label="lr, auc="+str(auc))
plt.legend(loc=4)
plt.show()

str1="Accuracy for lr with Un-Normalized"
lists.append(str1)
lists.append(report_lr)
str2 = "Confusion Matrix for lr"
lists.append(str2)
lists.append(cm_lr)
```

```
write_lines(outfile, lists)
```

- ***Classifiers implemented on Over-Balanced Dataset***

```
# Classifiers [Over-Balanced]

# Importing reqd. Libraries
import numpy as np
import pandas as pd
import re
import pickle
import nltk
from nltk.corpus import stopwords
from pandas import read_csv
nltk.download("stopwords")

def write_lines(outfile,lists):
    f = open(outfile, "a", encoding='utf-8')
    for lines in lists:
        lines=str(lines)+"\n"
        f.writelines(str(lines))


    f.close()

outfile = "results.txt"
lists = []


# Importing and Cleaning the dataset
dataset = read_csv("tweets-train.csv")
X = dataset.iloc[:, [0]].values
X = np.ndarray.tolist(X)
y = dataset.iloc[:, [1]].values
y = np.ndarray.tolist(y)
y_corpus =[]

# Label Encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
le.fit(y)
y = le.transform(y)
sum = np.sum(y)

# Pickling the Dataset [Persisting the dataset]
with open("X.pickle", "wb") as f:
    pickle.dump(X, f)
with open("y.pickle", "wb") as f:
    pickle.dump(y, f)

# Pre-processing the dataset, Normalizing the Tweets and Creating the corpus
corpus = []
for i in range(len(X)):
    origTweet = str(X[i])
    tweet = re.sub(r"\W", " ", str(X[i])) # Removing non-word characters
    tweet = tweet.lower() # Converting into lower case
    tweet = re.sub(r"\s+[a-z]\s+", " ", tweet) # Removing single characters in the corpus
    tweet = re.sub(r"that's","that is",tweet)
    tweet = re.sub(r"there's","there is",tweet)
    tweet = re.sub(r"what's","what is",tweet)
    tweet = re.sub(r"where's","where is",tweet)
    tweet = re.sub(r"it's","it is",tweet)
    tweet = re.sub(r"who's","who is",tweet)
    tweet = re.sub(r"i'm","i am",tweet)
    tweet = re.sub(r"she's","she is",tweet)
    tweet = re.sub(r"he's","he is",tweet)
    tweet = re.sub(r"they're","they are",tweet)
    tweet = re.sub(r"who're","who are",tweet)
    tweet = re.sub(r"ain't","am not",tweet)
    tweet = re.sub(r"wouldn't","would not",tweet)
    tweet = re.sub(r"shouldn't","should not",tweet)
    tweet = re.sub(r"can't","can not",tweet)
    tweet = re.sub(r"couldn't","could not",tweet)
    tweet = re.sub(r"won't","will not",tweet)
    tweet = re.sub(r"\W"," ",tweet)
    tweet = re.sub(r"\d"," ",tweet)
    tweet = re.sub(r"\s+[a-z]\s+"," ",tweet)
    tweet = re.sub(r"\s+[a-z]$"," ",tweet)
    tweet = re.sub(r"^[a-z]\s+"," ",tweet)
    tweet = re.sub(r"\s+", " ", tweet) # Removing multi-spaces by a single space
    # corpus.append(tweet)
    corpus.append(origTweet)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
text_Train, text_Test, sd_Train, sd_Test = train_test_split(corpus, y, test_size=0.25,
random_state=0)

# Creating TF-IDF Model using TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# tVectorizer = TfidfVectorizer(max_features=1000, min_df=3, max_df=0.6,
stop_words=stopwords.words("english"))
# tVectorizer = TfidfVectorizer(max_features=1000, min_df=3, max_df=0.6)
tVectorizer = TfidfVectorizer(ngram_range = (1, 3))
# tVectorizer = CountVectorizer(max_features=1000, min_df=3, max_df=0.6,
stop_words=stopwords.words("english"))
# tVectorizer = CountVectorizer(max_features=1000, min_df=3, max_df=0.6)
# tVectorizer = CountVectorizer()
text_Train = tVectorizer.fit_transform(text_Train)

# Random Over-sampling of the majority class
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
text_Train, sd_Train = ros.fit_sample(text_Train, sd_Train)

text_Test = tVectorizer.transform(text_Test)

# Fitting classifier to the Training set &  Predicting the Test set results
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

# Logistic Regression
clf1 = LogisticRegression(random_state=0)
clf1.fit(text_Train, sd_Train)
sd_Pred1 = clf1.predict(text_Test)

clf2 = SVC(kernel = "linear", random_state = 0)
clf2.fit(text_Train, sd_Train)
```

```
sd_Pred2 = clf2.predict(text_Test)

# Decison Tree Classifier
clf3 = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
clf3.fit(text_Train, sd_Train)
sd_Pred3 = clf3.predict(text_Test)

# Random Forest Classifier
clf4 = RandomForestClassifier(n_estimators=300, criterion = "entropy", random_state=0)
clf4.fit(text_Train, sd_Train)
sd_Pred4 = clf4.predict(text_Test)

# Voting Classifier
eclf = VotingClassifier(estimators=[('lr', clf1), ('svm', clf2), ('dtc', clf3), ('rf', clf4)],
voting='hard')
# eclf = VotingClassifier(estimators=[('lr', clf1), ('svm', clf2), ('rf', clf4)], voting='hard')
eclf.fit(text_Train, sd_Train)
sd_Pred = eclf.predict(text_Test)

# Visualizing the RESULTS

# Making the Confusion Matrix(Classification Evaluation Metric)
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(sd_Test, sd_Pred1)
cm_svm = confusion_matrix(sd_Test, sd_Pred2)
cm_dtc = confusion_matrix(sd_Test, sd_Pred3)
cm_rf = confusion_matrix(sd_Test, sd_Pred4)
cm_eclf = confusion_matrix(sd_Test, sd_Pred)

# Generating the Classification Report(Classification Evaluation Metric)
from sklearn.metrics import classification_report
report_lr = classification_report(sd_Test, sd_Pred1)
report_svm = classification_report(sd_Test, sd_Pred2)
report_dtc = classification_report(sd_Test, sd_Pred3)
report_rf = classification_report(sd_Test, sd_Pred4)
report_eclf = classification_report(sd_Test, sd_Pred)

# F_Score plot
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
```

```
f1_lr = f1_score(sd_Test, sd_Pred1, average='macro')
f1_svm = f1_score(sd_Test, sd_Pred2, average='macro')
f1_dtc = f1_score(sd_Test, sd_Pred3, average='macro')
f1_rf = f1_score(sd_Test, sd_Pred4, average='macro')
f1_eclf = f1_score(sd_Test, sd_Pred, average='macro')

y_plot = np.array([ f1_dtc, f1_rf, f1_eclf, f1_lr, f1_svm])
x_plot = np.array([20, 40, 60, 80, 100])
plt.scatter(x_plot, y_plot)
plt.scatter(x_plot, y_plot, color = "blue")
plt.plot(x_plot, y_plot, color = "fuchsia")
plt.xticks([20, 40, 60, 80, 100], ["dtc", "rf", "eclf", "lr", "svm"])
plt.xlabel("Classifiers")
plt.ylabel("F1_Score")
plt.show()

str1="Accuracy for lr with tfidf n-gram in 1,3"
lists.append(str1)
lists.append(report_lr)
str2 = "Confusion Matrix for lr"
lists.append(str2)
lists.append(cm_lr)

str1="Accuracy for svm with tfidf n-gram in 1,3"
lists.append(str1)
lists.append(report_svm)
str2 = "Confusion Matrix for svm"
lists.append(str2)
lists.append(cm_svm)

str1="Accuracy for dtc with tfidf n-gram in 1,3"
lists.append(str1)
lists.append(report_dtc)
str2 = "Confusion Matrix for dtf"
lists.append(str2)
lists.append(cm_dtc)

str1="Accuracy for rf with tfidf n-gram in 1,3"
lists.append(str1)
lists.append(report_rf)
```

```
str2 = "Confusion Matrix for rf"
lists.append(str2)
lists.append(cm_rf)

str1="Accuracy for eclf with tfidf n-gram in 1,3"
lists.append(str1)
lists.append(report_eclf)
str2 = "Confusion Matrix for rf"
lists.append(str2)
lists.append(cm_eclf)

# Pickling or saving our Classifier
with open("classifier.pickle", "wb") as f:
    pickle.dump(eclf, f)

# Pickling or saving our TfidfVectorizer
with open("tfidf_vectorizer.pickle", "wb") as f:
    pickle.dump(tVectorizer, f)

write_lines(outfile, lists)
```

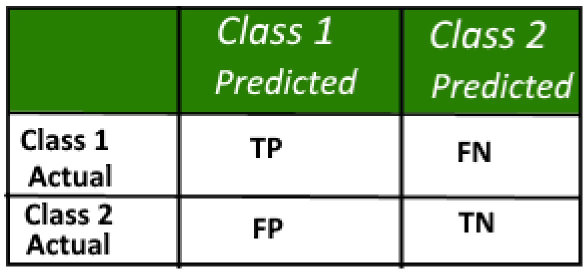
CHAPTER 9

Classification Performance Evaluation Metrics and Plots

[Results and Conclusion]

- ## Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem.
The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.
The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.



	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

• Positive (P) : Observation is positive (for example: is an apple).
• Negative (N) : Observation is not positive (for example: is not an apple).
• True Positive (TP) : Observation is positive, and is predicted to be positive.
• False Negative (FN) : Observation is positive, but is predicted negative.
• True Negative (TN) : Observation is negative, and is predicted to be negative.
• False Positive (FP) : Observation is negative, but is predicted positive.

Confusion Matrix for Decision Tree Classifier:

	Class 1 Predicted	**Class 2 Predicted**
Class 1 Actual	106	126
Class 2 Actual	91	659

Confusion Matrix for Logistic Regression Classifier:

	Class 1 Predicted	**Class 2 Predicted**
Class 1 Actual	173	59
Class 2 Actual	92	658

Confusion Matrix for Random Forest Classifier:

	Class 1 Predicted	**Class 2 Predicted**
Class 1 Actual	120	112
Class 2 Actual	53	697

Confusion Matrix for SVM Classifier:

	Class 1 Predicted	**Class 2 Predicted**
Class 1 Actual	156	76
Class 2 Actual	66	684

Confusion Matrix for Voting Classifier:

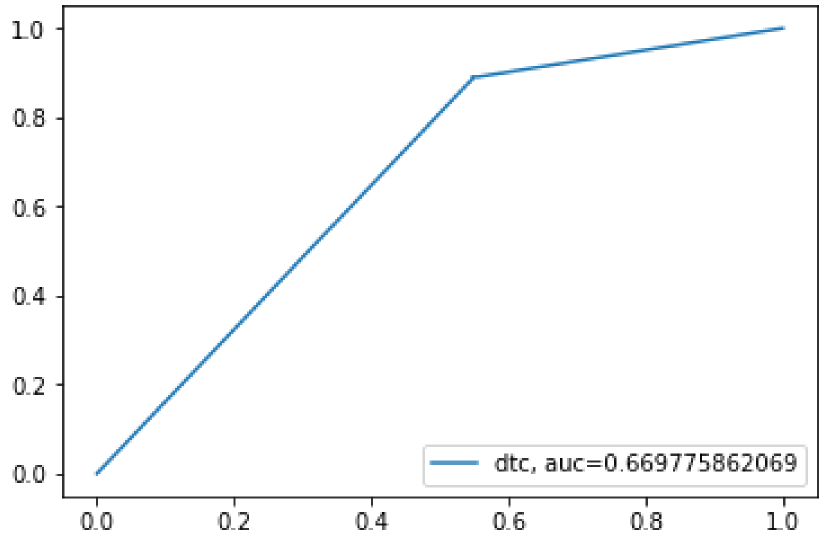
	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	166	66
Class 2 Actual	83	667

- ## Area under ROC Curve (AUC)

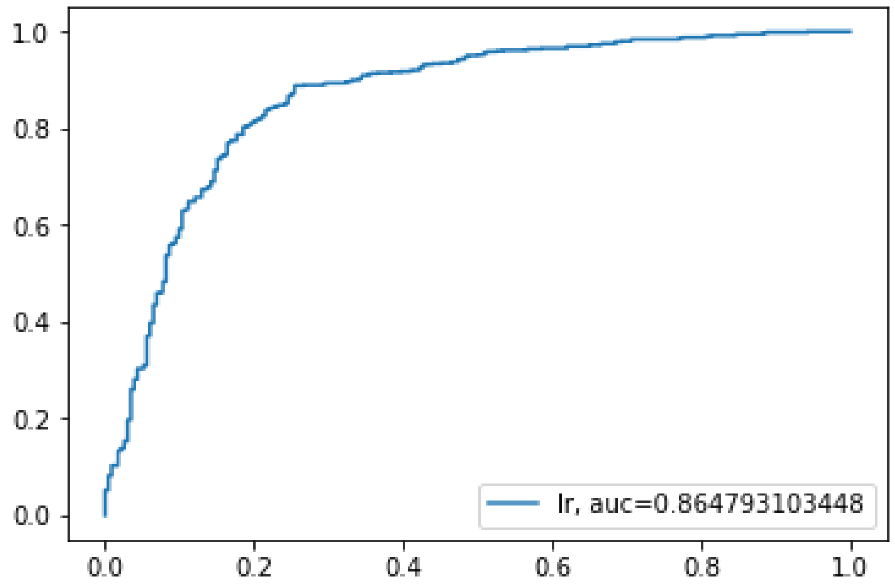
Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems. The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model that is as good as random. ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

->**Sensitivity** is the true positive rate also called the recall. It is the number of instances from the positive class that actually predicted correctly.
->**Specificity** is also called the true negative rate. Is the number of instances from the negative (second) class that were actually predicted correctly.

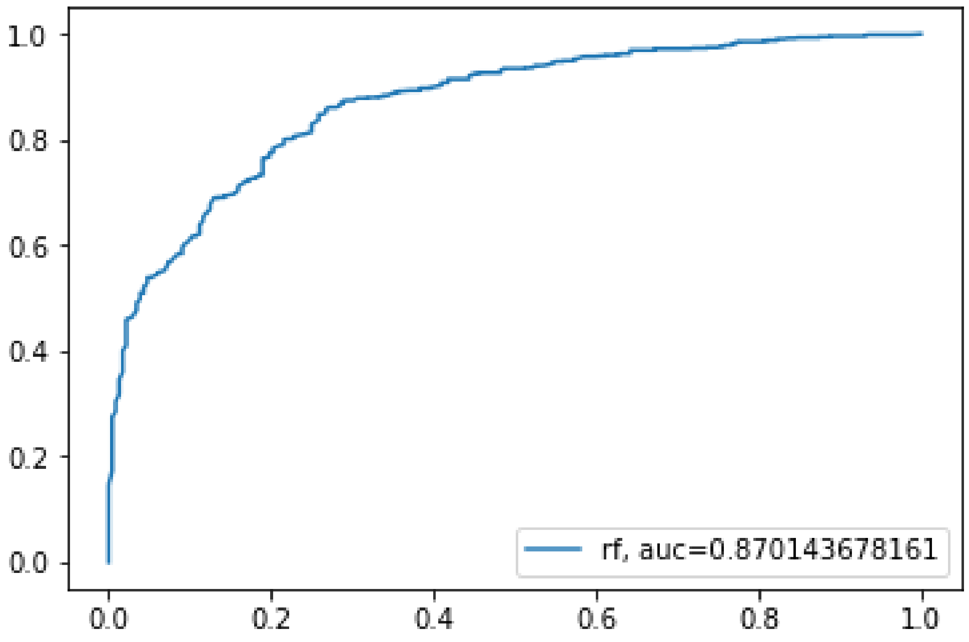
- **Decision Tree Classifier**



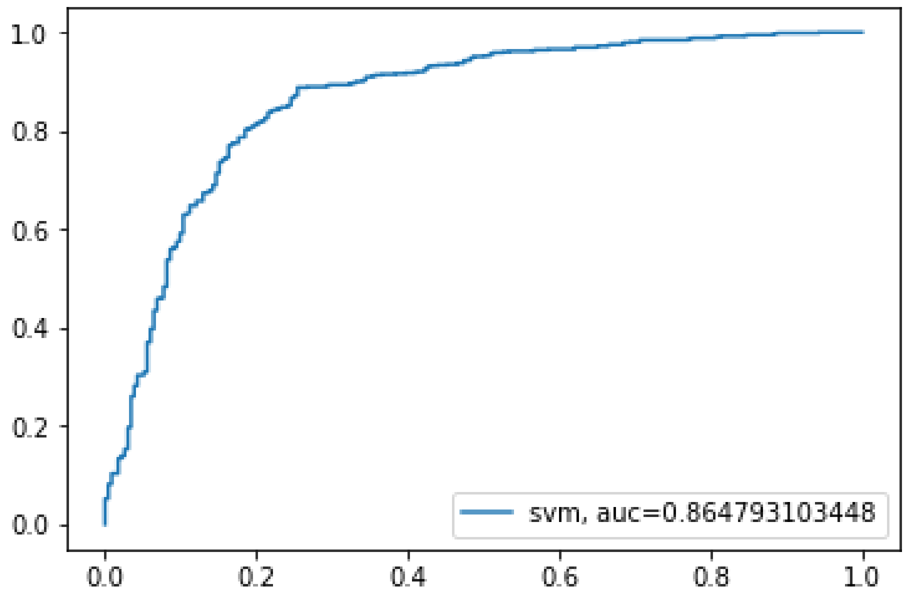
- **Logistic Regression Classifier**



- **Random Forest Classifier**



- **SVM Classifier**



- ## Classification Report

Classification Rate/Accuracy:

Classification Rate or Accuracy is given by the relation:

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall:

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN).

Recall is given by the relation:

$$\textit{Recall} = \frac{TP}{TP + FN}$$

Precision:

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP).
Precision is given by the relation:

$$\textit{Precision} = \frac{TP}{TP + FP}$$

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

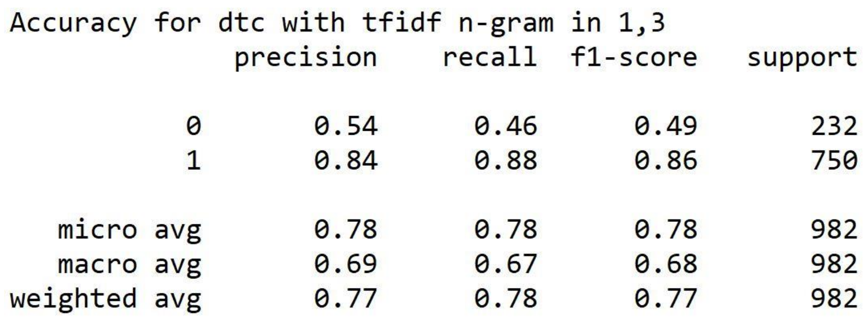
F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.
The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$\textit{F - measure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

RESULTS:

- **Classification Report for Decision Tree Classifier:**

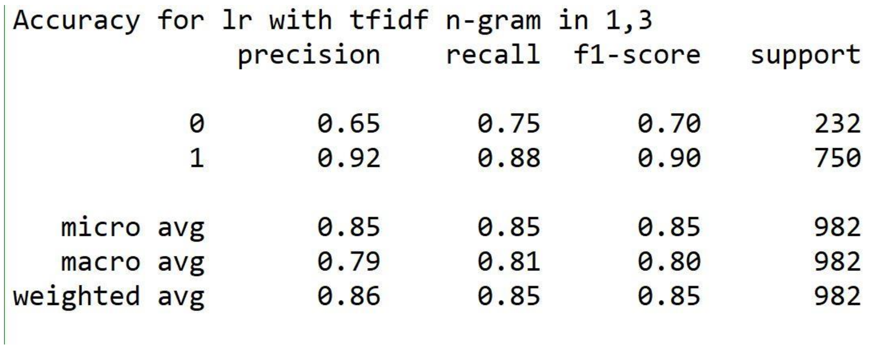


```
Accuracy for dtc with tfidf n-gram in 1,3
              precision    recall  f1-score   support

           0       0.54      0.46      0.49       232
           1       0.84      0.88      0.86       750

   micro avg       0.78      0.78      0.78       982
   macro avg       0.69      0.67      0.68       982
weighted avg       0.77      0.78      0.77       982
```

- **Classification Report for Logistic Regression Classifier:**

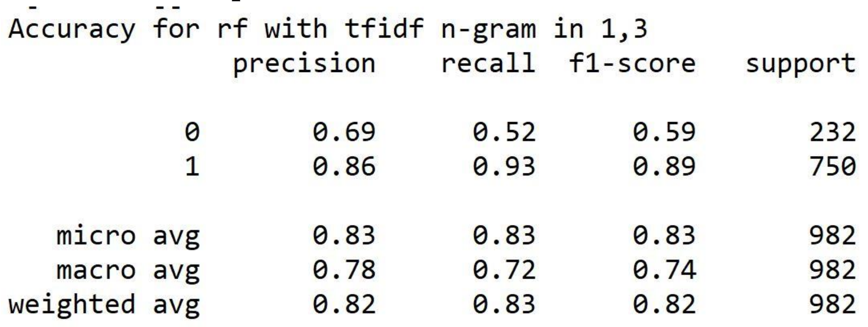


```
Accuracy for lr with tfidf n-gram in 1,3
              precision    recall  f1-score   support

           0       0.65      0.75      0.70       232
           1       0.92      0.88      0.90       750

   micro avg       0.85      0.85      0.85       982
   macro avg       0.79      0.81      0.80       982
weighted avg       0.86      0.85      0.85       982
```

- **Classification Report for Random Forest Classifier:**

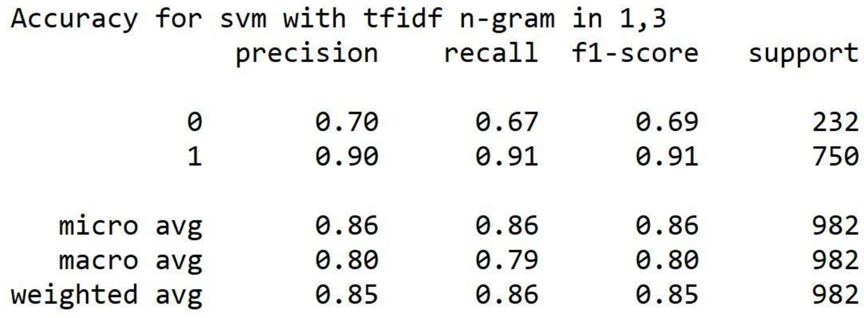


```
Accuracy for rf with tfidf n-gram in 1,3
              precision    recall  f1-score   support

           0       0.69      0.52      0.59       232
           1       0.86      0.93      0.89       750

   micro avg       0.83      0.83      0.83       982
   macro avg       0.78      0.72      0.74       982
weighted avg       0.82      0.83      0.82       982
```

- **Classification Report for SVM Classifier:**



```
Accuracy for svm with tfidf n-gram in 1,3
              precision    recall  f1-score   support

           0       0.70      0.67      0.69       232
           1       0.90      0.91      0.91       750

   micro avg       0.86      0.86      0.86       982
   macro avg       0.80      0.79      0.80       982
weighted avg       0.85      0.86      0.85       982
```

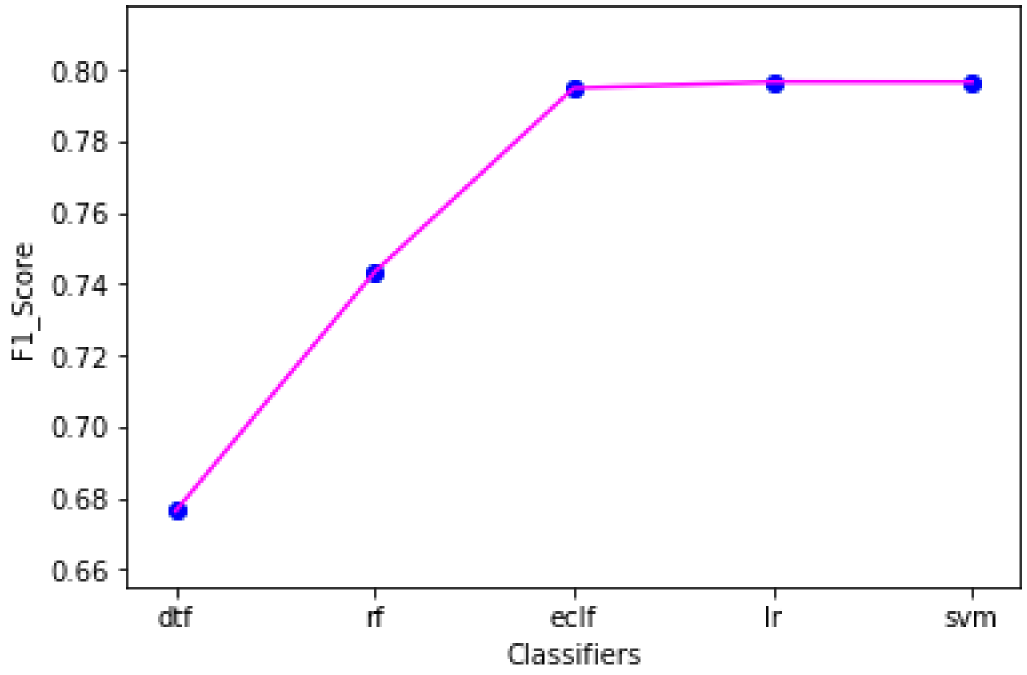
❑ **Classification Report for Ensemble Voting Classifier:**

```
Accuracy for eclf with tfidf n-gram in 1,3
              precision    recall  f1-score   support

           0       0.67      0.72      0.69       232
           1       0.91      0.89      0.90       750

   micro avg       0.85      0.85      0.85       982
   macro avg       0.79      0.80      0.79       982
weighted avg       0.85      0.85      0.85       982
```

- **F1_Score Plot**



SOFTWARE AND HARWARE REQUIREMENTS

1.1 SOFTWARE REQUIREMENTS

- Python
- Spyder (Python IDE)
- Nltk library
- Scikit-learn library
- Numpy library
- Pandas library
- Matplotlib library
- Imb-learn library

1.2 HARDWARE REQUIREMENTS

- Operating system: Windows 8/10, Linux, MAC OS
- Display: 15.6”
- RAM: 4GB
- Processor: Intel CORE i5

REFERENCES

1. **Author**: Jason Brownlee

    - **Title of Work:** Machine Learning Mastery using Python

2. **Authors**: Steven Bird, Ewan Klein and Edward Loper

    - **Title of Work:** Natural Language Processing with Python

3. **Dataset:** https://data.world/xprizeai-env/sentiment-of-climate-change

4. **SemEval-2016 Task 6:** Detecting Stance in Tweets Saif M. Mohammad National Research Council Canada, Svetlana Kiritchenko National Research Council Canada, Parinaz Sobhani University of Ottawa, Xiaodan Zhu National Research Council Canada, Colin Cherry National Research Council Canada

5. **Connecting Targets to Tweets:** Semantic Attention-based Model for Target-specific Stance Detection Yiwei Zhou and Alexandra I. Cristea [Department of Computer Science, University of Warwick, Coventry, UK] , Lei Shi [University of Liverpool, Liverpool, UK]

6. **Stance Detection with Bidirectional Conditional Encoding:** Isabelle Augenstein and Tim Rocktaschel [Department of Computer Science, University College London], Andreas Vlachos and Kalina Bontcheva [Department of Computer Science, University of Sheffield]

7. **Recognizing Stances in Ideological On-Line Debates:** Swapna Somasundaran [Dept. of Computer Science, University of Pittsburgh] and Janyce Wiebe [Dept. of Computer Science and The Intelligent Systems Program, University of Pittsburgh]

8. **Tracking Climate Change Opinions from Twitter Data:** Xiaoran An [Dept. of Mech. & Indus. Eng. Northeastern University Boston], Auroop R. Ganguly [Dept. of Civil & Env. Eng. Northeastern University Boston], Yi Fang [Dept. of Comp. Eng., Santa Clara University], Steven B. Scyphers [Marine Science Center Northeastern University Nahant], Ann M. Hunter [Behavioral Neuroscience Northeastern University Boston] and Jennifer G. Dy [Dept. of Elec. & Comp. Eng. Northeastern University Boston]