Detection of Abusive Language on Social Media - Twitter

Course: Natural Language Processing
Instructor: Professor Lu Xiao
Team Members:
Aishwarya Reddy
Naga Nikhita Annadatha
Rucha Guntoorkar
Srinivas Pachika Reddy

Index

1. Introduction
2. Dataset
3. Data-Preprocessing
4. Feature Engineering
5. Methodology
6. Result
7. Conclusion
8. Future Scope
9. References

1. Introduction:

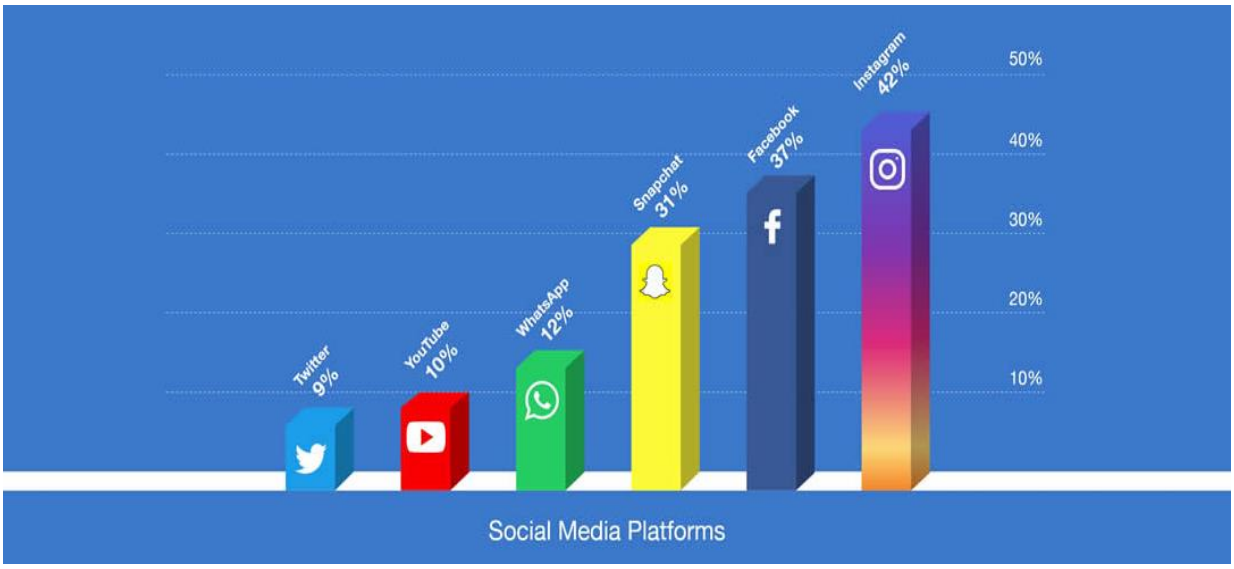
The Internet has become an essential part of life. In recent years, social media has become the platform to share ideas and content, thoughts, expressing views. People from all age groups are the users of social media. It is making a great impact on life. The content shared or posted on social media can be clean or profane. It has become necessary to detect such content and remove from social media since it is affecting people's mindset and thinking. If we consider the example of cyberbullying, it can destroy one's confidence. People have even committed suicide because of that. Hateful content on social media may promote crimes as well. So somewhere protecting social media from profane or offensive content has obtained prime importance.

These days there are many people who have a social media profile online and are willing to put all their personal info online making it visible to anyone. There are many ways to interact such as video calls, texting, sharing opinions that have become so common making communication and interaction even easier. Statistics say that most percentages of participants who use social media are either teenagers or kids. Almost 70% of teenagers or kids aged between 8 to 15 have their social media profiles. Apart from the meaningful and useful conversations, social media also has a widespread of abusive interactions. It is said that almost 40 in 100 young people face hate speech or racist comments. There are many victims of cyberbullying and almost 1 in 4 kids had to face cyberbullying.

Therefore it is necessary for these kinds of interaction to be controlled. This has to be done by the social media platform in order to eliminate this online hate speech. This controlling has to be done in a way that it automatically understands the language and detects the abusive words or

hate speech in the comments or posts. Once hate speech is detected the user has to be reported. But this process is not as easy as it sounds. It is
difficult to analyze the comment understand it and say if it's safe or not. Doing this will include Natural Language Processing and Machine Learning. This process takes in the comment as input and produces a labeled corpus saying if it's abusive or not abusive.

From the image above it can be seen that the tweets that were posted by the users have hate speech in it. This kind of tweet has to be detected and the users must be reported by the platform.



The above image shows the statistics of abusive comments on social media. Instagram has more abusive comments as compared to other social media with 42% and Twitter has the least abusive comments with 9%.

2. Dataset

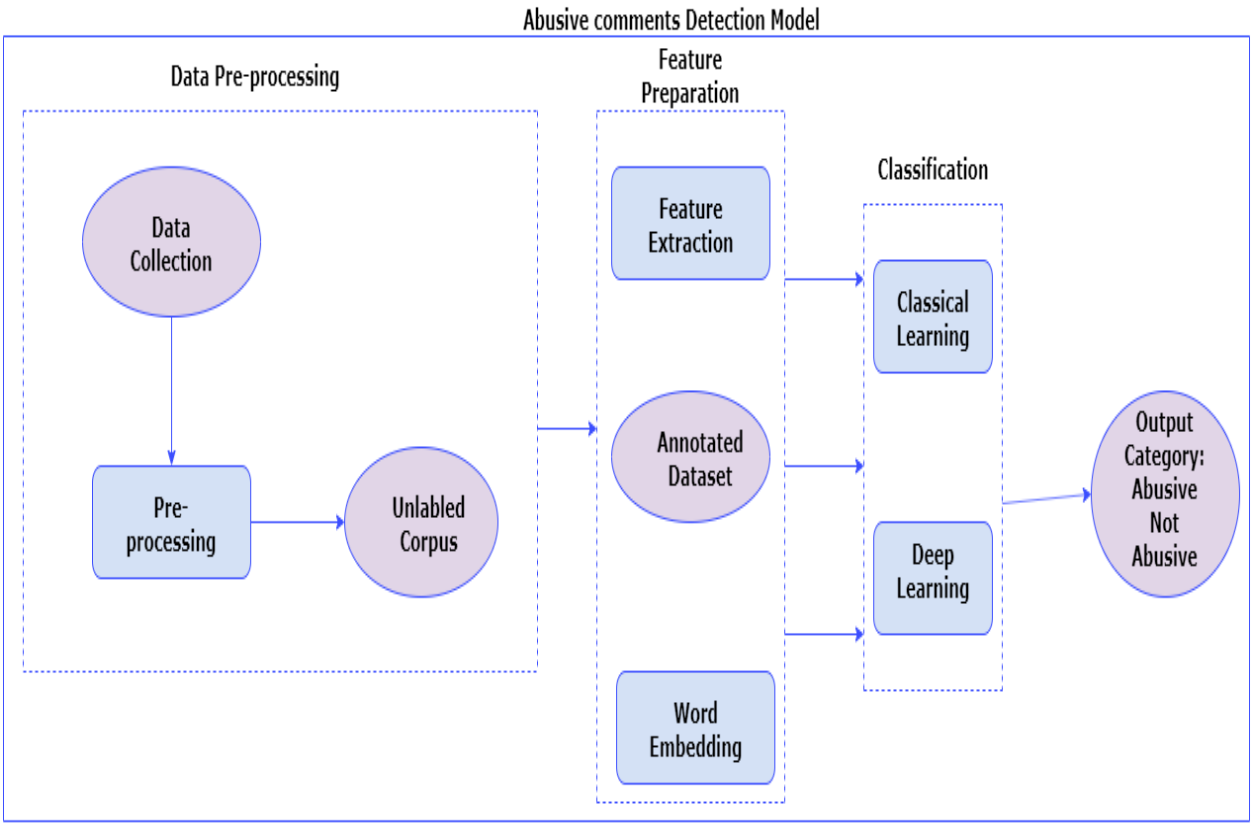
For this project, we extracted tweets from the tweeter. We extracted tweets using tweeter API and tweepy library. The dataset consists of 13241 rows. The dataset is stored in .tsv format. The tweets are extracted for a particular duration and by giving some keywords or hashtags. Since we wanted tweets with more abusive words or language we chose the keywords accordingly. It has fields like id, tweet, and label. Id is the id of the user who tweeted the corresponding tweet, tweets are the twitter comment by the user. The tweets consist of words, symbols, emoticons, etc. Since we are only interested in the tweets, we did not extract the other information provided by the API. We also manually added the column called label and it is manually annotated. We checked each tweet to find if it is offensive or not and gave value either “OFF” if it is offensive otherwise “NOT”. For example “@USER Someone should've Taken this piece of shit to a volcano. ␣” is the offensive tweet so it is labeled as “OFF” whereas, “It’s not my fault you support gun control” is labeled as “NOT” because it is not an offensive comment or tweet. This is important to train the data model and get a more accurate result.

Following is the screenshot of the dataset with fields id, tweet, and label with some sample data.

```
id	tweet	subtask_a	subtask_b	subtask_c
86426	@USER She should ask a few native Americans what their take on this is.	OFF	UNT	NULL
90194	@USER @USER Go home you’re drunk!!! @USER #MAGA #Trump2020 👊🇺🇸👊 URL	OFF	TIN	IND
16820	Amazon is investigating Chinese employees who are selling internal data to third-party sellers looking for an edge in the competitive marketplace. URL #Amazon #MAGA #KAG #CHINA #TCO
62688	"@USER Someone should'veTaken"" this piece of shit to a volcano. 😂"""	OFF	UNT	NULL
43605	@USER @USER Obama wanted liberals &amp; illegals to move into red states	NOT	NULL	NULL
97670	@USER Liberals are all Kookoo !!!	OFF	TIN	OTH
77444	@USER @USER Oh noes! Tough shit.	OFF	UNT	NULL
52415	@USER was literally just talking about this lol all mass shootings like that have been set ups. it’s propaganda used to divide us on major issues like gun control and terrorism
45157	@USER Buy more icecream!!!	NOT	NULL	NULL
13384	@USER Canada doesn’t need another CUCK! We already have enough #LooneyLeft #Liberals f**king up our great country! #Qproofs #TrudeauMustGo	OFF	TIN	IND
82776	@USER @USER @USER It’s not my fault you support gun control	NOT	NULL	NULL
42992	@USER What’s the difference between #Kavanaugh and @USER   One of these men admitted to groping a 15 year old girl years ago.  The other is going to be #confirmed to the SCJ   #Dems
28414	@USER you are a lying corrupt traitor!!! Nobody wants to hear anymore of your lies!!! #DeepStateCorruption URL	OFF	TIN	IND
54920	@USER @USER @USER It should scare every American!  She is playing Hockey with a warped puck!	NOT	NULL	NULL
56392	@USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @USER @
86735	@USER you are also the king of taste	NOT	NULL	NULL
95686	#MAGA @USER  🎶 Sing like no one is listening  ♥ Love like you've never been hurt  ✔ Vote GOP when no one is watching  🙈  And don't listen to Liberals' dirt URL	NOT	NULL
71446	"5/5: @USER The time is right for this House to respond to the concerns of all Canadians. Four out of five Canadians support stronger gun control and with good reason.” #guncontrol
23958	@USER Besides Jax’s mom and maybe Ope he is hands down my favorite he’s like the only good person on the show 😂NOT	NULL	NULL
```

High-level Overview:

- The first step here is to preprocess and clean the tweets by removing the URL's, hashtags and other emoticons using regular expressions
- Then we converted the tweets into lower case and performed tokenization and then did stemming and lemmatization on those tokens.
- We then extracted feature vectors using the TF-IDF method and then used them for training our machine learning models.
- We performed classification using 5 different algorithms like SVM, Multinomial Naive Bayes, Logistic Regression, Decision Tree and Random Forest.
- As a final step, we compared the performance of these models and choose the best out of them to classify the tweets as offensive and non-offensive.



3. Data-Preprocessing:

In the Data Processing part of our project, we first cleaned the data. This is ensured by keeping only the comments and the labels. In this process, the URL is removed. Also, the @ sign in the comments has also been removed. Later Tokenization is performed. First, the sentences are lower-cased to avoid duplicates and then the tokenization is carried out. The words which are most commonly used in the language have also been removed, i.e the removal of the stop words is carried out.

1. **Removal of the Uniform Resource Locator(URL):**
   Although the URL information is valuable we cannot access that in your project. It is always a good idea to remove the URL. We are able to detect the URL with the help of regular expressions.

2. **Removal of comments with @ sign at the beginning:**
   Here our aim is to focus on the original comment. So, it is needed to remove any comments with the @ at the beginning of any word. All the comments which contain @ sign have been successfully removed.

3. **Tokenization:**
   - There will always be a difficulty in determining whether we need to consider a group of words to be only word or as many words. For instance, let us take a word white-and-black. Here it can be considered as three words: white, and white. Or it can be considered as single multiple words. There is no correct answer to the question of which of the above two is correct. The approach for the word tokenization in NLP uses a different theory.
   - The process which breaks down the string into words like units is known as tokenization. The tokenization the English language is a pretty straight forward process. It basically splits the whitespace-delimited data by matching any kinds of special cases like the slang, abbreviations, contractions, and emoticons. The prefix is not matched, the process of matching the special cases are performed once again. The suffix is removed if still there is no match. Before the tokenization is performed all the data is converted into the lower case. This is to avoid classifying the same token in a different token.

4.Morphology and Lemmatization :

- Morphology is defined as the study of affixes or words as meaningful building blocks. The smallest unit of meaning in a given word is termed as a morpheme. Let us take an example

word dog. The word dogs contain two morphine here, one is the dog and the other is -s. The morphine dog is domesticated and is free morphine as it can be used as a stand-alone word. When we consider the -s, it is known as bound morphine. It is called bound morphine because it has to be attached to some other free morphine to make it meaningful. The morphine -s is called as the inflectional affix. They are called so because they add up grammatical information to the word that is already existing. Now let us talk about another affix such as -able, which is known as derivational bound affixes because these create an entirely new word with a different meaning.

- A lexeme is defined as the base form which is used to link the word sense to the complete set of the possible forms. Lemmatization is one of the most common steps performed in natural language data. What exactly is lemmatization? Lemmatization is the process of converting tokens into their base dictionary form. By doing this, we are basically reducing dimensionality which will help us to improve our application. But there is a catch here. By doing lemmatization the ambiguities are also introduced into the scenario. The reason for these ambiguities is the removal of inflectional morphine. In our project, the lemmatizer basically takes in a list that contains all the tokens, along with their parts of speech. The rules are applied which are dependent on endings of each token. And then we get respective lemma by converting the tokens. From first entry to last entry in the table, the lemmatizer applies the rules following in-order. The tokens that undergo a considerable form change, for example, was to be, rules are basically followed. Let us consider an example word making, which is basically tagged as a verb. All verb rules which are in table are seen by lemmatizer. The word that we considered that is making does not have ending matching with either s, es, ies, or ed. So, all these rules are not considered and are skipped. It goes to the rules to find the ending with ing and when it matches, it then stripes it and replaces it with an e. The word that is formed is made which also could be a verb.Hence, the lemmatizer will add the word make as a potential lemma and then it continues. If for example let us consider another word playing, which is also tagged as a verb. Firstly the ing is removed but the rule will fail because playe is not a valid word. What lemmatizer does in this situation is that it would move to the last rule and not replace ing with e, but rather just strip it off.

5. **Removal of the stop words:**

The words that are most commonly used are known as stop words. In English, some examples of stop words are “a”, “the”, “an”, “in”, “as” etc. These words take up a lot of space and that is the reason why they have to be removed. In computing, it is always essential to filter the stop words out before we actually perform the processing of the

natural language data. By removing the stop words, we are basically improving efficiency. The removal of the words is basically done to avoid our database to consume any unnecessary space and also save valuable processing time. Apart from the most commonly used words, the list can be extended by adding words that you don't want to be present and want them to be removed.

```
import re
import nltk
nltk.download('punkt', 'stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.lancaster import LancasterStemmer
lancaster_stemmer = LancasterStemmer()
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
```

```
[nltk_data] Downloading package punkt to stopwords...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

```
def take_data_to_shower(tweet):
    noises = ['URL', '@USER', '\'ve', 'n\'t', '\'s', '\'m']

    for noise in noises:
        tweet = tweet.replace(noise, '')

    return re.sub(r'[^a-zA-Z]', ' ', tweet)


def tokenize(tweet):
    lower_tweet = tweet.lower()
    return word_tokenize(lower_tweet)


def remove_stop_words(tokens):
    clean_tokens = []
    stopWords = set(stopwords.words('english'))
    for token in tokens:
        if token not in stopWords:
            if token.replace(' ', '') != '':
                if len(token) > 1:
                    clean_tokens.append(token)
    return clean_tokens


def stem_and_lem(tokens):
    clean_tokens = []
    for token in tokens:
        token = wordnet_lemmatizer.lemmatize(token)
        token = lancaster_stemmer.stem(token)
        if len(token) > 1:
            clean_tokens.append(token)
    return clean_tokens
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\srini\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\srini\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\srini\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!

True
```

```
tqdm.pandas(desc="Cleaning Data Phase I...")
clean_tweets['tweet'] = tweets['tweet'].progress_apply(take_data_to_shower)

tqdm.pandas(desc="Tokenizing Data...")
clean_tweets['tokens'] = clean_tweets['tweet'].progress_apply(tokenize)

tqdm.pandas(desc="Cleaning Data Phase II...")
clean_tweets['tokens'] = clean_tweets['tokens'].progress_apply(remove_stop_words)

tqdm.pandas(desc="Stemming And Lemmatizing")
clean_tweets['tokens'] = clean_tweets['tokens'].progress_apply(stem_and_lem)

text_vector = clean_tweets['tokens'].tolist()
```

```
Cleaning Data Phase I...: 100%|██████████| 13240/13240 [00:00<00:00, 93863.07it/s]
Tokenizing Data...: 100%|██████████| 13240/13240 [00:01<00:00, 7967.89it/s]
Cleaning Data Phase II...: 100%|██████████| 13240/13240 [00:03<00:00, 4393.81it/s]
```

Feature Engineering

Comments scraped from social media websites lack structure. Those comments may contain various spelling mistakes along with Unicode symbols, emoticons, hyperlinks, website-specific tokens and markers, non-standard language, and a nearly limitless variability in the lexicon. This deficiency of structure makes the task of abusive language detection really challenging. We have used some NLP techniques as described below which will help us in extracting features from text and allow us to pass them as features for training machine learning models.

Feature Vectors

In General, while performing text mining tasks the data we deal with is represented in the form of document term matrix. Each row or instance of this matrix represents a single text instance which is also referred to as document and each column or attribute represents a token that was encountered at least once in our corpus. Entries of this matrix are most of the time boolean representing whether a specific term occurs in the document or not, normalized scored which are done by a method called TF-IDF. This matrix is very high dimension and sparse. We have used this method of TF-IDF for extracting features from our data. The calculation of this Tf-idf score is done as shown below.

$$tf(t, d) = f(t, d)$$

$$idf(t, D) = \log N\ nt$$

$$tf * idf(t, d, D) = tf(t, d) \times idf(t, D)$$

f(t, d) is the number of times a term t occurs in a specific document d. Here D is total set of documents, N is the total number of documents in D, and nt is the number of documents in which term t appears. If a term t occurs frequently in document d, but not a common word that occurs across all documents then that is probably an important word and hence should be given importance and this is what exactly TF-IDF calculation takes into account.

```
from sklearn.feature_extraction.text import TfidfVectorizer

def tfid(text_vector):
    vectorizer = TfidfVectorizer()
    untokenized_data =[' '.join(tweet) for tweet in tqdm(text_vector, "Vectorizing...")]
    vectorizer = vectorizer.fit(untokenized_data)
    vectors = vectorizer.transform(untokenized_data).toarray()
    return vectors

def get_vectors(vectors, labels, keyword):
    if len(vectors) != len(labels):
        print("Unmatching sizes!")
        return
    result = list()
    for vector, label in zip(vectors, labels):
        if label == keyword:
            result.append(vector)
    return result
```

```
vectors_a = tfid(text_vector) # Numerical Vectors A
labels_a = subtask_a_labels['subtask_a'].values.tolist() # Subtask A Labels
```

```
Vectorizing...: 100%|████████| 13240/13240 [00:00<00:00, 1016596.22it/s]
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

def classify(vectors, labels, type="DT"):
    # Random Splitting With Ratio 3 : 1
    train_vectors, test_vectors, train_labels, test_labels = train_test_split(vectors, labels, test_size=0.25)

    # Initialize Model
    classifier = None
    if(type=="MNB"):
        classifier = MultinomialNB(alpha=0.7)
        classifier.fit(train_vectors, train_labels)
    elif(type=="KNN"):
        classifier = KNeighborsClassifier(n_jobs=4)
        params = {'n_neighbors': [3,5,7,9], 'weights':['uniform', 'distance']}
        classifier = GridSearchCV(classifier, params, cv=3, n_jobs=4)
        classifier.fit(train_vectors, train_labels)
        classifier = classifier.best_estimator_
    elif(type=="SVM"):
        classifier = SVC()
        classifier = GridSearchCV(classifier, {'C':[0.001, 0.01, 0.1, 1, 10]}, cv=3, n_jobs=4)
        classifier.fit(train_vectors, train_labels)
        classifier = classifier.best_estimator_
    elif(type=="DT"):
        classifier = DecisionTreeClassifier(max_depth=800, min_samples_split=5)
        params = {'criterion':['gini','entropy']}
        classifier = GridSearchCV(classifier, params, cv=3, n_jobs=4)
        classifier.fit(train_vectors, train_labels)
        classifier = classifier.best_estimator_
    elif(type=="RF"):
        classifier = RandomForestClassifier(max_depth=800, min_samples_split=5)
        params = {'n_estimators': [n for n in range(50,200,50)], 'criterion':['gini','entropy'], }
        classifier = GridSearchCV(classifier, params, cv=3, n_jobs=4)
        classifier.fit(train_vectors, train_labels)
        classifier = classifier.best_estimator_
    elif(type=="LR"):
        classifier = LogisticRegression(multi_class='auto', solver='newton-cg',)
        classifier = GridSearchCV(classifier, {"C":np.logspace(-3,3,7), "penalty":["l2"]}, cv=3, n_jobs=4)
        classifier.fit(train_vectors, train_labels)
        classifier = classifier.best_estimator_
    else:
        print("Wrong Classifier Type!")
        return

    accuracy = accuracy_score(train_labels, classifier.predict(train_vectors))
    print("Training Accuracy:", accuracy)
    test_predictions = classifier.predict(test_vectors)
    accuracy = accuracy_score(test_labels, test_predictions)
    print("Test Accuracy:", accuracy)
    print("Confusion Matrix:", )
    print(confusion_matrix(test_labels, test_predictions))
```

Classifiers:

In this project, we have used several classifiers and computed the result for each. The following screenshot shows the libraries used in the implementation to compute the result.

Model Fitting

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

Naive Bayes Classifier

It is a probabilistic approach in which classification is done using Bayes Theorem with a strong assumption that every feature is not dependent on another feature. Here probability of each category is calculated using Bayes theorem and outputs the category with the highest value. It is one of the widely used classifiers in NLP. It is simplistic in its approach. It does not require any parameter to be tuned. It is faster as compared to other classifiers. Naive Bayes performs well with a high dimension of input data.

$$P(C_k|X) = \frac{P(C_k) \cdot P(X|C_k)}{P(X)}$$

Bayes's theorem is based on conditional probability.
The above expression P(Ck) represents the prior probability of the outcome. It is based on the old data or record or from the training data.
P(X) is the probability of the predictor variable. From the old data, it tells what is the probability of the combination of predicting variables.
P(X/Ck) is the conditional probability. It is measured for each class.
P(Ck/X) is the posterior probability. From all observed info, a priori info is updated and we can measure this probability that the corresponding class has.

The following screenshot shows the accuracy of training and testing after using the Naive Bayes classifier.

Multinomial Naive Baiyes

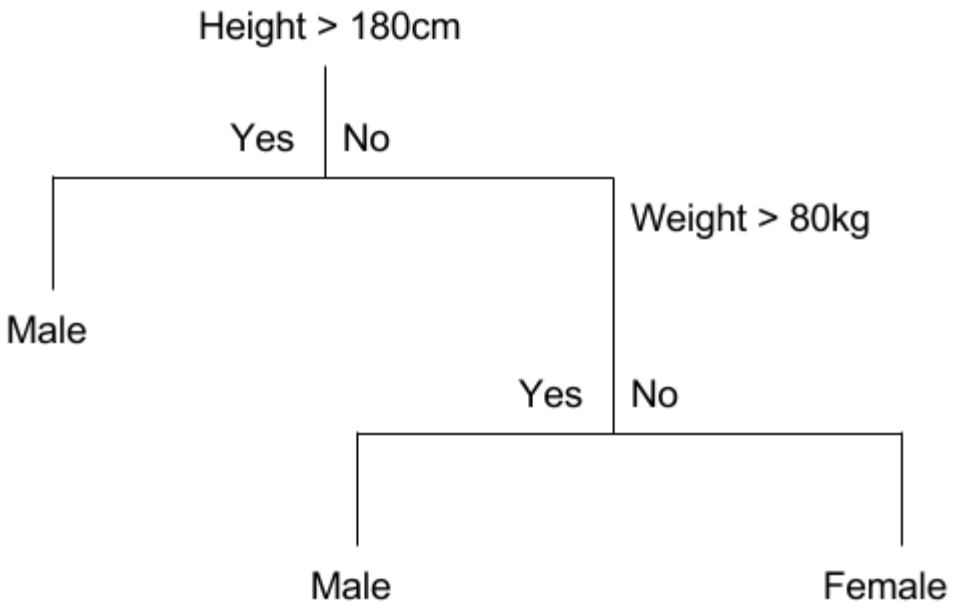
```
print("\nBuilding Model Subtask A...")
classify(vectors_a[1000:2000], labels_a[1000:2000], "MNB")
```

```
Building Model Subtask A...
Training Accuracy: 0.8106666666666666
Test Accuracy: 0.716
Confusion Matrix:
[[172   1]
 [ 70   7]]
```

Decision Tree:

A decision tree is basically a supervised machine learning in which the data is split continuously in accordance with certain parameters. The decision tree basically consists of three parts namely the nodes, edges and the leaf nodes.
Nodes: They test for the value of a certainly given attribute.
Edges: these are also called as Branches. They connect to the next node and correspond to the output of the test.
Leaf Nodes: They are the nodes present at the terminating end, which are used to predict the output.
Mainly there are two types of Decision Trees, they are Classification tree and Regression Tree.
Classification Tree:

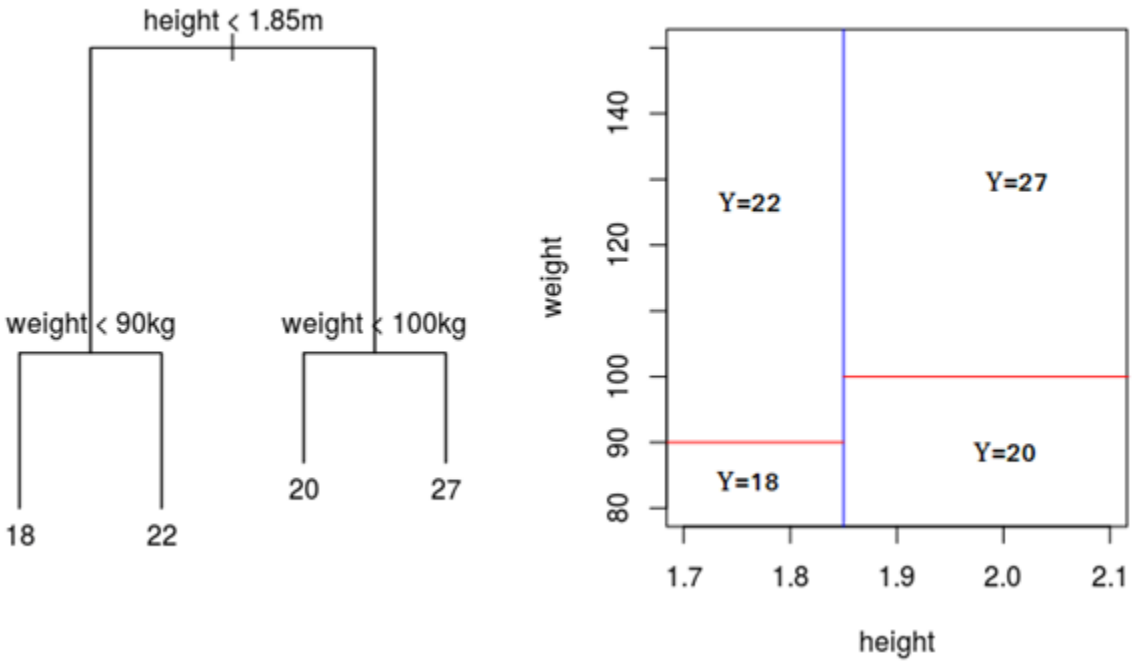
- It is basically a Yes/No type tree.
- These trees are often built following the process named binary recursive partitioning. It is basically an iterative process in which the data is split into partitions and then they are further split into further branches.



Example of the classification tree.

Regression Tree:

- It is basically a Continuous data type.
- In these kinds of trees, the target variables are taken as a continuous value.



Example of Regression tree.

Creation of a Decision Tree:

The decision trees are built by following the approach known as divide and conquer. It is follows recursive partitioning. Divide and conquer splits the data into smaller subsets and further split the subsets into smaller subsets. The following are the steps for the divide and conquer algorithm.

- First, select the test for the root node. Then make a branch of each possible outcome of the created test.
- Then split them into further subsets.
- Repeat the recursion for each branch by using the instances that reach the branch.
- Finally, stop the recursion for the branch if all of the instances have the same class.

Decision Tree Classifier:

- By using this algorithm, we first start from the root of the tree and then further split the data that arises to the largest Information Gain(IG)
- We then carry on the splitting process in a repeated fashion at each child node until we find a pure leaf node.
- Practically we put a limit on the depth of the tree to avoid the problem of overfitting. We actually compromise on purity. That is there might be some impurities in the final leaves.

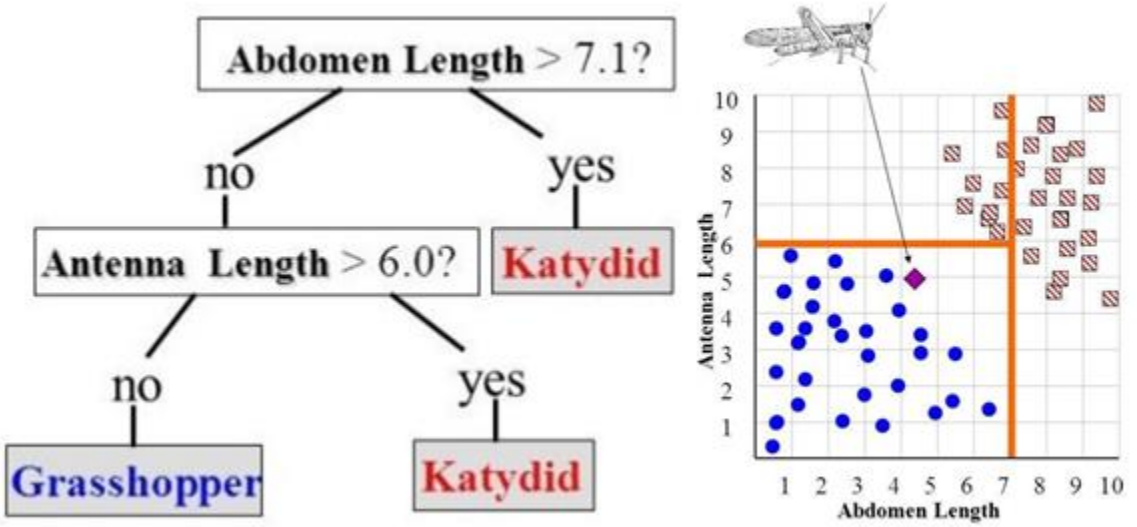


Fig. Classifying whether an insect is a Grasshopper or a Katydid based on Antenna Length and Abdomen Length.

Advantages:

- It is less expensive to construct.
- It is fast at carrying out the classification of unknown records.
- For a small-sized tree, it is extremely easy to interrupt.
- It basically excludes any features which are unimportant.

Disadvantages:

- It is very easy to overfit.
- Minute changes in the training data can cause very large changes in the decision login.
- Large Trees can be hard to interpret and the decision that they made may sometimes seem counter-intuitive.

Application in Real Life:

- Physics
- Astronomy
- System Control
- Financial Analysis.

- Biomedical Engineering.
- Medicines.

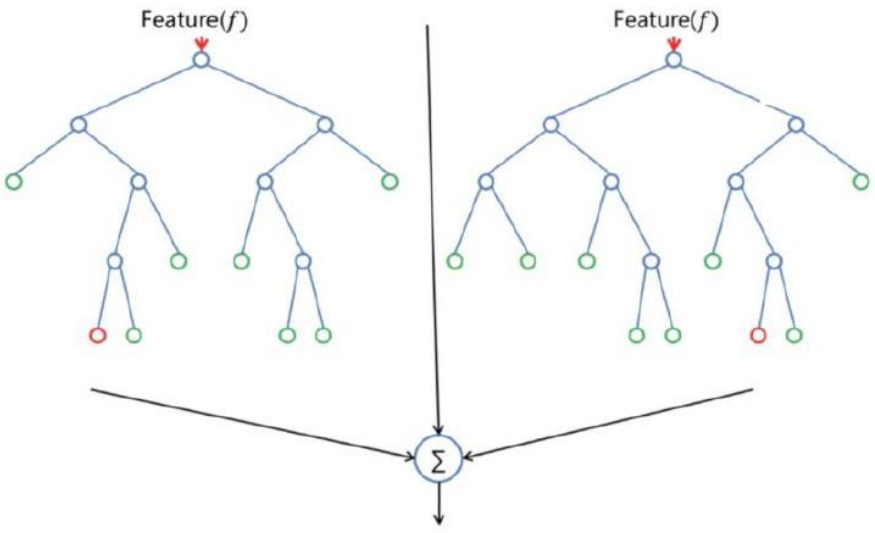
```
# Decision Tree
```

```
print("\nBuilding Model Subtask A...")
classify(vectors_a[1000:2000], labels_a[1000:2000], "DT")
```

```
Building Model Subtask A...
Training Accuracy: 0.992
Test Accuracy: 0.688
Confusion Matrix:
[[137  30]
 [ 48  35]]
```

Random Forest

It is another flexible and easy to use classifier in NLP. It can be used even without hyper-tunning of the parameter. It is a supervised learning classifier. It forms forests which are random in nature. The forests are nothing but decision trees that are created using the bagging method. In nutshell, this algorithm generates many trees and after combining them we get a more accurate result. Typically the decision tree looks like one given below



With a random forest, we can also deal with regression work. When trees are expanded, more randomness is added. So here instead of looking at the most appropriate feature, it takes the best feature from the random set of features which results in diverse results in a better model.

That is why we consider a random subset of the feature while segregating the data into training and testing. For getting more accurate results or prediction we should have features that could impart prediction also the decision trees should be uncorrelated.
Even with the advantages of random forest, it is kind of slow because of the time-consuming process. The model is also difficult to understand as compared to the decision tree to follow the path and make a decision.

Following screenshot shows the accuracy result for training and testing for Random Forest Classifier

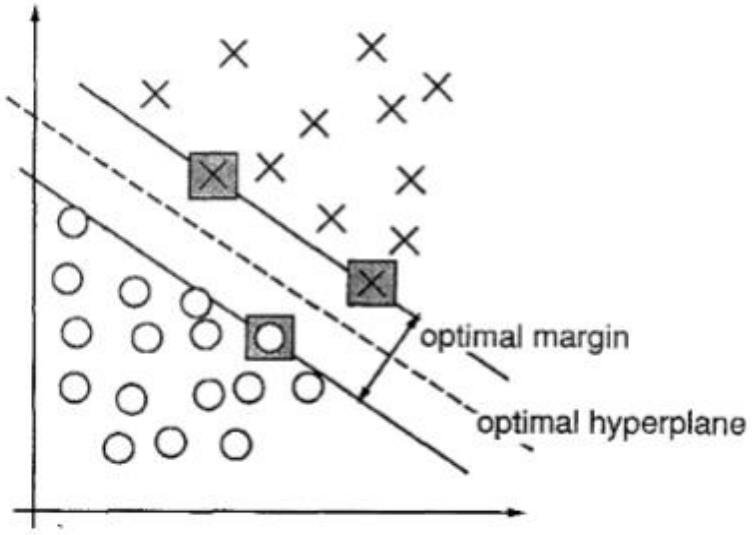
Random Forest

```
print("\nBuilding Model Subtask A...")
classify(vectors_a[1000:2000], labels_a[1000:2000], "RF")
```

```
Building Model Subtask A...
Training Accuracy: 1.0
Test Accuracy: 0.732
Confusion Matrix:
[[159   3]
 [ 64  24]]
```

SVM:

SVM's are one of the powerful machine learning classifiers. It has the capability to discriminate between the classes accurately. SVM, when used with Natural Language Processing, is mostly used for classifying the data because of their capability to work with high-dimensional vectors and their performance. This kind of classifier tends to map the input vector to a high-dimensional space so that the two classes can be split from each other by a separating hyper-plane. These two classes that are split will have an optimal distance from the hyperplane and the support vectors. Margin is considered to be the difference between the hyperplane and the support vectors and will be equal to 1. The separation boundary for linear SVMs will be linear.



Above is the image of SVM splitting between 2 classes in a 2-D space.

Non-linear separating hyperplanes are allowed by making some kernel modifications in the kernel function. There's also a parameter C that allows some tolerable error in the support vectors and incorporates a soft margin distance. If the value of C is higher, the tolerance for incorrect classification is decreased. SVM has to solve for a parameter α in-order for it to work in its Lagrangian dual objective function

$$\sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

yi here is the ground truth vector
xi is an input feature vector
n is the number of training examples
Prediction can be calculated using the formula below,

$$f(x) = \sum_{i}^{n} \alpha_i y_i k(x, x_i)$$

Following screenshot shows the accuracy result for training and testing for SVM Classifier

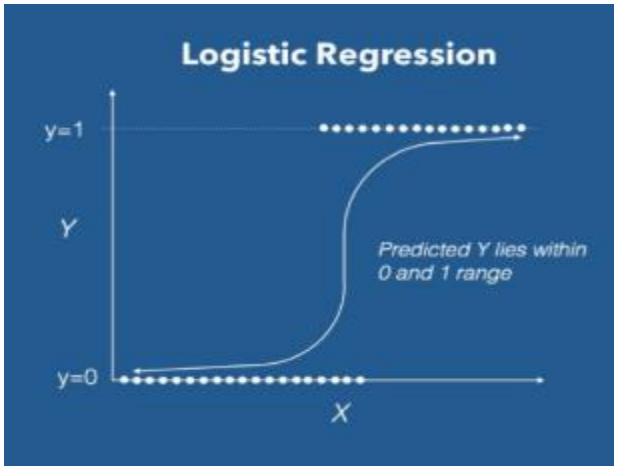
SVM Classifier

```
print("\nBuilding Model Subtask A...")
classify(vectors_a[1000:2000], labels_a[1000:2000], "SVM")
```

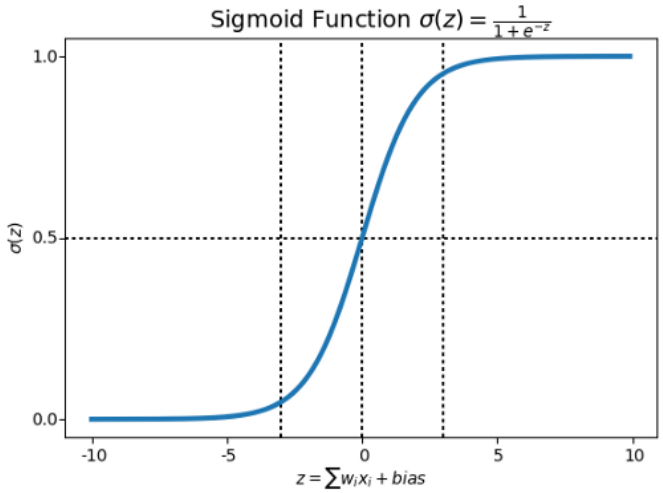
```
Building Model Subtask A...
Training Accuracy: 0.66
Test Accuracy: 0.664
Confusion Matrix:
[[166   0]
 [ 84   0]]
```

Logistic regression

Logistic regression is one of the classification algorithms that is used for assigning observations to a distinct set of classes. A few examples of this classification problem are to classify if it's a fraudulent transaction or not, if it's email spam or not, etc. This classifier transforms the output using a logistic sigmoid function and returns a probability value. Logistic regression is a predictive analysis algorithm and is based on probability.



Linear Regression uses a complex cost function namely 'Sigmoid function' which is also known as a logistic function. The cost of the function is limited between 0 and 1. This function was initially developed in order to describe the properties of population growth that were rising quickly. As can be seen from the figure above it is an S-shaped curve that takes in any real-valued number and maps it into a value that is between 0 and 1.



The Sigmoid function is,

$$f(x) = \frac{1}{1+e^{-(x)}}$$

The estimation of logistic regression coefficients must be done from the training data. This estimation is done using Maximum-likelihood estimation which is used by a number of machine learning algorithms. A value close to 1 would be considered as the best coefficient for default class and a value close to 0 would be the best coefficient for other classes. A search procedure is used to seek coefficients values that are responsible for reducing the error in probabilities predicted by the model.

Following screenshot shows the accuracy result for training and testing for Logistic Regression Classifier

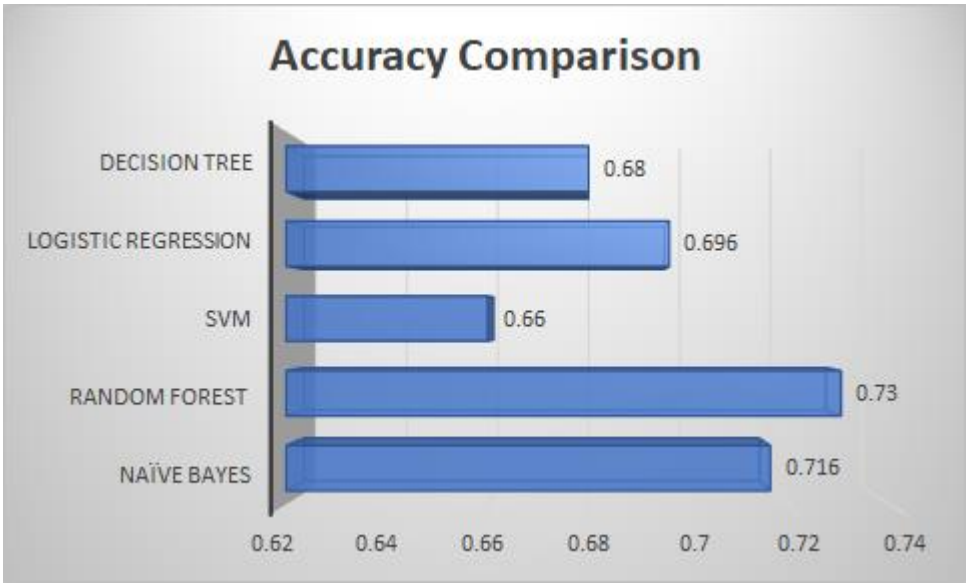
Logistic Regression Classifier

```
print("\nBuilding Model Subtask A...")
classify(vectors_a[1000:2000], labels_a[1000:2000], "LR")
```

```
Building Model Subtask A...
Training Accuracy: 0.9986666666666667
Test Accuracy: 0.696
Confusion Matrix:
[[139  20]
 [ 56  35]]
```

Conclusion:

From the results that we've obtained, it can be seen that the highest accuracy is obtained from the Random Forest classifier with 73% accuracy and the least accuracy is obtained from the SVM classifier with 66% accuracy. Decision Tree, Logistic Regression and Naive Bayes classifiers obtained an accuracy of 68%, 69.6%, and 71.6% respectively.



Future Scope:

Nowadays people are trying various approaches to put the hateful or abusive content on social media in order to not to get detected by the machine learning or deep learning algorithms. One of the ways is posting memes or images. Our implementation does not support detecting abusive content on social media. It is restricted to texts only.

Another scope being our implementation failed to detect some of the tweets which looked abusive or contains abusive words but they are not actually abusive in nature. We need to improve our training model with more accurate features to increase accuracy and detect such tweets.

References:

https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54
https://www.datacamp.com/community/tutorials/random-forests-classifier-python
https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=9949&context=theses