CS464: Introduction to Machine Learning
Project Progress Report-Group 6

Members: Mustafa Efe Tamyapar 21902856, Öykü Erhan 21901541, Zülal Nur Hıdıroğlu 21903125, Selen Görgün 21802674, Berk Saltuk Yılmaz 21903419

Title: Sarcasm Detector
Project Name: Just Kidding!

1. Introduction

We are all being sarcastic from time to time. Expressing sarcasm and understanding it during face-to-face communication is more effortless. However, detecting sarcasm in written communication is not as easy. Moreover, some people are not experts in perceiving sarcasm and may have a hard time understanding if something is ironically written. This project aims to detect texts containing sarcasm/irony to clear any confusion these texts might create. We are using Reddit comments/posts to train our model. The dataset is composed of the comments and their authors, subreddits, scores which is the number of upvotes minus the number of downvotes, upvotes, and downvotes, the creation date, the parent comment, and a label that indicates if the comment is sarcastic or not. We make use of machine learning models such as Random Forest Classifier, Multinomial Naive Bayes Classifier, and Support Vector Machine Classifier.

1.1. Random Forest Classification

Random Forest is a machine learning algorithm that is commonly used to combine the output of a number of decision trees into one result based on a random sampling of the data set. Randomly selected subsets of the data are used to develop the decision trees [1].

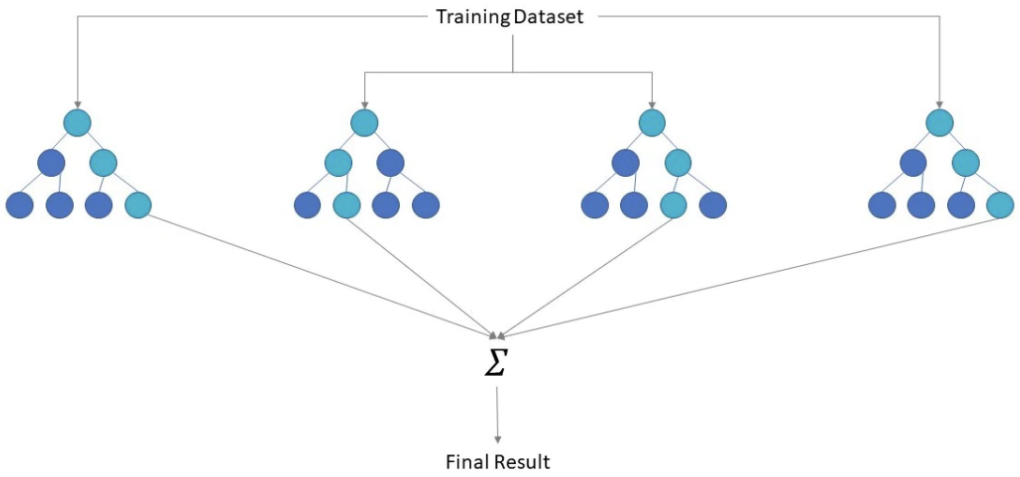


Figure 1: Diagram of Random Forest Classifier [1]

As each tree has the same nodes, gini index is used to determine which feature to be selected for the further split. Gini index is used to determine purity of the splits in the decision tree and the feature with the lowest impurity is selected. To select the feature with the lowest gini index is used.

$$Gini\ Index = 1 - \sum_{i=1}^{n} (P_i)^2$$
$$= 1 - [(P_+)^2 + (P_-)^2]$$

P_+ represents the probability of the positive class and P_+ represents the probability of the negative class in the Gini index.

Random forest classification is chosen because it has a reduced risk of one of the most challenging problems in machine learning, which is overfitting the model. Because of the averaging of the uncorrelated trees this algorithm gives low bias and low variance, therefore the classifier won't overfit [2].

1.2. Naive Bayes Classification

Naive Bayes classifications are based on applying Bayes' Theorem, which states that every feature is classified as independent. The Bayes' rule is

$$P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(Y|X)P(X)}{P(Y)}$$

In Naive Bayes' model, if there are n number of features, the formula is used to estimate the likelihood equation which k is the type in the estimation.

$$P(Y = y_k|X_1, X_2,..., X_n) = \frac{P(Y=y_k)P(X_1,X_2,...,X_n|Y=y_k)}{P(X_1,X_2,...,X_n)} \propto P(Y = y_k)P(X_1, X_2,..., X_n|Y = y_k)$$

$$P(X_1, X_2,..., X_n|Y = y_k) = P(Y = y_k) \prod_{i=1}^{n} P(X = X|Y = y_k)$$

To make the calculations much more easier, the logarithm of the equation is taken and the formula becomes

$$log(P(Y = y_k) \prod_{i=1}^{n} P(X = X|Y = y_k)) = log(P(Y = y_k)) * log(\sum_{i=1}^{n} P(X_i, Y))$$

In our project, the number of times a word is used in a text affects the sarcastic tone of the text. Since it is a necessity of calculating the number of times a word passed in a text, the Multinomial Bayes Model could be used. The Multinomial Bayes Model is based on Bayes' Model. It predicts the tag of a text considering the probability of each text in the text. Then it

outputs the tag with the highest probability. In our project, the likelihood of each feature is calculated as follows

$$P(w_t|Y = y_k) \approx \frac{\sum_{i=1}^{n} X_{i,t} * z_{i,k}}{\sum_{t'=1}^{|V|} \sum_{i=1}^{n} X_{i,t'} * z_{i,k}}$$

In the equation wt represents the feature t. $X_{i,t}$ represents the number of times the feature t is repeated. |V| represents total number of features and $z_{i,k}$ is the label of data i. The likelihood of k type data is formulates as

$$P(Y = y_k) \approx \frac{N_k}{N}$$

where N is total number of data and N_k is the total number of data with label k.

1.3. Support Vector Classification

Support vector machine is based on finding a hyperplane in N dimensional space, N is the number of features in this case, that clearly separates data points on the coordinate system. There are different hyperplane options which can be chosen to classify data points. But the main purpose is to find the hyperplane that maximizes the distance between the data points of both classes. The distance is also called margin. This margin maximization helps us predict the classification of the future data in a better way. [3]

For linearly separable classes, linear classification for support vector machine is in form of:

$$f(x) = w^T x + b$$

In this function, x is the data point, w is the normal to the line which is known as weight vector and b is the bias.

Given the training data (x_i, y_i) for i = 1 to N, with $x_i \in R^d$ and $y_i \in \{-1, 1\}$, linear classification function separates the data as follows:

$$f(x_i) \geq 0 \text{ for } y_i = 1$$
$$f(x_i) < 0 \text{ for } y_i = -1$$

Since $w^T x_+ + b = 0$ and $c(w^T x_+ + b) = 0$ define the same plane, normalization of w can be chosen as desired. So, data is normalized in a way that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$, respectively for positive and negative support vectors. In this normalization form, x_+ is the nearest data point to the linear classifier with the positive label and x_- is the nearest data point to the linear classifier with the negative label. The best w is chosen by maximizing the margin which is the distance between x_+ and x_- to the linear classifier in this case.

Now, the margin is given by:

$$\frac{w}{||w||}(x_+ - x_-) = \frac{w^T(x_+ - x_-)}{||w||} = 2\,||w||$$

Support vector machine can be formulated as an optimation:

$$max\ \frac{2}{||w||}\ subject\ to\ w^T x_i + b \geq +1 \text{ if } y_i = 1 \text{ for } i = 1..N$$

$$max\ \frac{2}{||w||}\ subject\ to\ w^T x_i + b \leq -1 \text{ if } y_i = 1 \text{ for } i = 1..N$$

2. Work Done

The work done until this point consists of data preprocessing, training & testing three machine learning models.

2.1. Data Preprocessing

At first, the CSV file that contains a balanced training sarcasm dataset is preprocessed. The instances that have null “comment” sections are dropped and all of the comments are converted to lowercase. Then, all of the punctuation is removed and they are tokenized. After this step, all words are categorized as adjectives, verbs and nouns. After categorization, words are lemmatized and stop words are also dropped. The final versions of tokens are stored. All this process is done with Natural Language Toolkit.

After storing final versions of words, the data is split into training and test data where 70% of data is reserved for training. Test dataset is further divided into validation and test dataset. The labels are encoded and the preprocessing is completed.

```
# Read the labels and the data
labels = pd.read_csv('train-balanced-sarcasm.csv', dtype=int, usecols= ['label'], nrows = 100000)
sarcasm_data = pd.read_csv('train-balanced-sarcasm.csv', dtype= str, usecols = ['comment'], nrows = 100000)

sarcasm_data = pd.DataFrame(sarcasm_data)
sarcasm_data['comment'].dropna(inplace=True)
sarcasm_data['comment'] = sarcasm_data['comment'].astype(str)

# make all comments lowercase
sarcasm_data['comment'] = sarcasm_data['comment'].str.lower()

# remove punctuations
def remove_punctuations(text):
    for char in string.punctuation:
        text = text.replace(char, '')
    return text
sarcasm_data['comment'] = sarcasm_data['comment'].apply(remove_punctuations)
# tokenization
sarcasm_data['comment'] = sarcasm_data['comment'].apply(word_tokenize)

# tag adjectives, verbs and adverbs for lemmatization
tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV

for i in range(sarcasm_data.shape[0]):
    # Declaring Empty List to store the words that follow the rules for this step
    Final_words = []
    # Initializing WordNetLemmatizer()
    word_Lemmatized = WordNetLemmatizer()
    # pos_tag function below will provide the 'tag' i.e if the word is Noun(N) or Verb(V) or something else.
    for word, tag in pos_tag(sarcasm_data['comment'].iloc[i]):
        # Below condition is to check for Stop words and consider only alphabets
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
    # The final processed set of words for each iteration will be stored in 'text_final'
    sarcasm_data.loc[i,'final'] = str(Final_words)

print("Finished preprocessing")
```

Output:

```
Finished preprocessing
                                                 comment  \
0                                          [nc, and, nh]
1      [you, do, know, west, teams, play, against, we...
2      [they, were, underdogs, earlier, today, but, s...
3      [this, meme, isnt, funny, none, of, the, new, ...
4                  [i, could, use, one, of, those, tools]
...                                                   ...
99995                                   [so, jealous]
99996  [yeah, we, all, know, it, is, the, baby, boome...
99997                          [story, setting, artstyle]
99998  [ebola, virus, to, papyrus, to, the, german, i...
99999                        [ford, is, just, that, good]

                                                   final
0                                          ['nc', 'nh']
1      ['know', 'west', 'team', 'play', 'west', 'team...
2      ['underdog', 'earlier', 'today', 'since', 'gro...
3      ['meme', 'isnt', 'funny', 'none', 'new', 'york...
4                        ['could', 'use', 'one', 'tool']
...                                                  ...
99995                                    ['jealous']
99996  ['yeah', 'know', 'baby', 'boomer', 'cause', 'w...
99997                        ['story', 'set', 'artstyle']
99998  ['ebola', 'virus', 'papyrus', 'german', 'iris'...
99999                                  ['ford', 'good']

[100000 rows x 2 columns]
```

2.1.1. TF-IDF

We also obtained our data in the form of vectors. For this purpose, we used the Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer of Scikit Learn. Term frequency (TF) is basically the ratio of a word's frequency inside a document (Number of word w divided by number of all words in document d). Document frequency (DF) is the frequency of a word inside all documents. TF-IDF is the multiplication of TF and inverse DF.[4]

After obtaining TF-IDF vectors, we have used them to train and test our models.

```
# Vectorize the text data
Tfidf_vect = TfidfVectorizer(max_features=5000)
Tfidf_vect.fit(sarcasm_data['comment'])
Train_X_Tfidf = Tfidf_vect.transform(Train_X)
Test_X_Tfidf = Tfidf_vect.transform(Test_X)
```

2.2. Train & Test

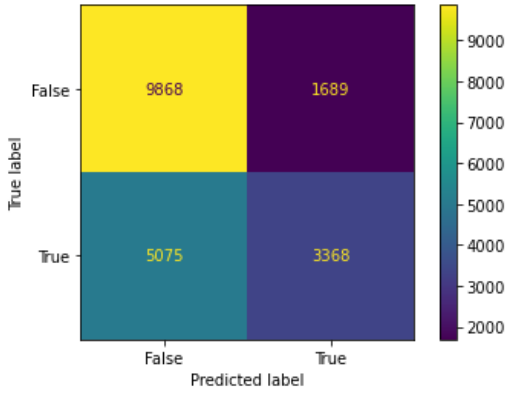
Up until now, we have trained three different models: Multinomial Naive Bayes, Support Vector Machine, and Random Forest.

2.2.1. Multinomial Naive Bayes

We have used the Multinomial Naive Bayes classifier of the Scikit-Learn library.

```
#Naive Bayes
NAIVE_BAYES = naive_bayes.MultinomialNB()
# fit the training dataset on the classifier
NAIVE_BAYES.fit(Train_X_Tfidf, Train_Y)
# predict the labels on validation set
predictions_NAIVE_BAYES = NAIVE_BAYES.predict(Test_X_Tfidf)
# Calc. accuracy
print("NAIVE BAYES Accuracy Score ->", accuracy_score(predictions_NAIVE_BAYES,
Test_Y)*100)
print("Confusion Matrix for RFC:")
print(confusion_matrix(Test_Y, predictions_NAIVE_BAYES, labels=[0,1]))
```

```
NAIVE BAYES Accuracy Score -> 66.18
Precision ->  0.6660075143365631
Recall ->  0.3989103399265664
F1 Score ->  0.498962962962963
Confusion Matrix for RFC:
```

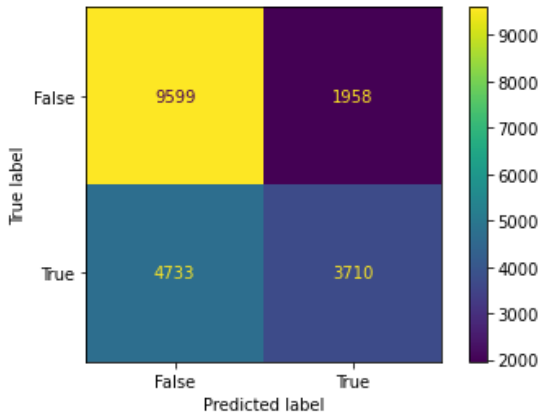


2.2.2. Support Vector Machine

We have used the Support Vector Machine classifier of the Scikit-Learn library.

```
# SVM
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Find the accuracy score
print("SVM Accuracy Score -> ", accuracy_score(predictions_SVM, Test_Y)*100)
print("Confusion Matrix for SVM:")
print(confusion_matrix(Test_Y, predictions_SVM, labels=[0,1]))
```

```
SVM Accuracy Score ->  66.545
Precision ->  0.6545518701482004
Recall ->  0.4394172687433377
F1 Score ->  0.5258309120544256
Confusion Matrix for SVM:
```

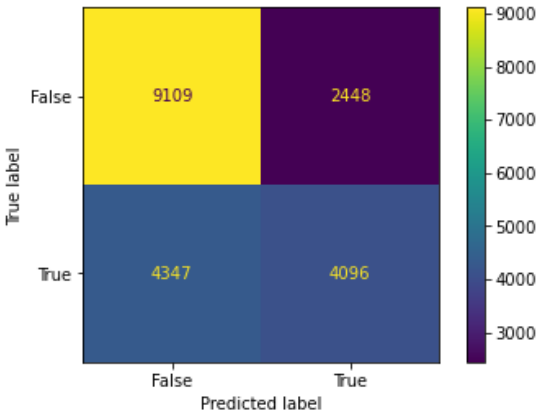


2.2.3. Random Forest

We have used the Random Forest Classifier of the Scikit-Learn library.

```
# Random Forest
RFC = RandomForestClassifier()
# fit the training dataset on the classifier
RFC.fit(Train_X_Tfidf, Train_Y)
# predict the labels on validation set
predictions_RFC = RFC.predict(Test_X_Tfidf)
print(predictions_RFC)
# Calc. accuracy
print("RFC Accuracy Score ->", accuracy_score(predictions_RFC, Test_Y)*100)
print("Confusion Matrix for RFC:")
print(confusion_matrix(Test_Y, predictions_RFC, labels=[0,1]))
```

```
RFC Accuracy Score -> 66.025
Precision ->  0.6259168704156479
Recall ->  0.48513561530261756
F1 Score ->  0.5466070594515247
Confusion Matrix for RFC:
```



2.3. Conclusions So Far

Considering the results of our current models, the order of accuracy from high to low is SVM, RFC, MNB respectively. However, the accuracy and f1 scores are low in general since each model resulted in an accuracy less than 70% and f1 score less than %60.

This accuracy might originate from our unbalanced dataset that is split as 57960 false and 42310 true values. Since there are more than one million instances in our dataset, we have only used the first 100000 instances for the time being. The unbalanced dataset might be the result of the low accuracy.

In addition, the TF-IDF might have decreased the accuracy of the models. The TF-IDF vectorizes the data according to the ratio of a word's frequency inside a document. In our case, we tried to find the most frequent words used in sarcastic comments in order to classify them. According to our observations after the low accuracy rates, the dataset might not contain words that should be classified as sarcastic frequently. Similarly, some words that

were used very frequently in the dataset might not be actually sarcastic. Since the context of the words is not considered, TF-IDF might decrease the accuracy for our dataset.

3. Work Left

We are planning to develop neural network models with higher accuracy rates and f1 scores. Also, we will use more suitable pre-processing techniques for our dataset.

In order to get better results in the pre-processing stage, we have decided to try the Word2vec vectorizer to consider the synonyms. The main difference between TF-IDF and Word2vec is that while TF-IDF vectorizes the words according to their frequency, Word2vec tries to understand the context by looking at the semantic closeness between words.

In terms of neural network models, we will decide what model we are going to use based on our ongoing research. Some of the options are Recurrent Neural Networks(RNN) and Convolutional Neural Networks(CNN), in order to increase the overall accuracy. We will compare the results and try to find the most accurate model.

Currently, we are using the first 100.000 rows of the dataset so the data is imbalanced. We will try to use a more balanced portion of the dataset to improve f1 score. In addition, Laplace smoothing for the naive bayes model will be considered.

4. Division of Work

For the purpose of ensuring an equal workload for all members of the group, the major tasks have been divided among the five members. These parts are related to models we plan to train. During the course of the project, two members of the group will implement and evaluate each classification algorithm. In this way, we will be able to make better decisions when it comes to choosing the algorithm that will be improved. This will allow us to be a better group. There will be a collaborative effort between all five group members in order to develop and test the algorithms simultaneously with the preparation of reports and presentations. The final phase of the project will require all the members of the group to work together in order to prepare the final product and the final details at the end of the phase involving the algorithm development.

The research and implementation of the neural network model will be carried out by Öykü Erhan, Selen Görgün. In the data processing section, the addition of word2vec will be carried out with the whole group. The RFC model will be created by Mustafa Efe Tamyapar and Berk Saltuk Yılmaz. The SVM model will be developed by Öykü Erhan and Selen Görgün. The Multinomial Naive Bayes will be done by Zülal Nur Hıdıroğlu. Finally, BERT's pretrained model will be done by Mustafa Efe Tamyapar, Zülal Nur Hıdıroğlu and Berk Saltuk Yılmaz. The output of each algorithm will be discussed by the developing group, and at the and all outputs will be compared by everyone. With this plan we will ensure that the division of equal workload is maintained.

5. References

[1] IBM Cloud Education, “What is Random Forest?”, *IBM Cloud Learn Hub,* 7-Dec-2020. [Online]. Available: https://www.ibm.com/cloud/learn/random-forest. [Accessed: 26-Nov-2022].

[2] A. Saini, “Random Forest Algorithm for absolute beginners in Data Science,” *Analytics Vidhya*, 26-Aug-2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/. [Accessed: 26-Nov-2022].

[3] S. Ray, “SVM: Support Vector Machine Algorithm in machine learning,” *Analytics Vidhya*, 29-Nov-2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. [Accessed: 27-Nov-2022].

[4] C. Maklin, “TF IDF: TFIDF python example,” *Medium*, 21-Jul-2019. [Online]. Available: https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76. [Accessed: 02-Dec-2022].