Haseeb Chaudhury, Animesh Deb,

Sri Tarun Gulumuru, Al Shafian Bari

CSC 44800 - Artificial Intelligence

12/10/23

Text Classification

Abstract:

Text classification is a machine learning technique that is fundamental to natural language processing (NLP), a field of artificial intelligence, linguistics, and computer science. NLP is the idea that a machine can break down text to understand how humans communicate and convey information. With the rise of social media, text classification is becoming more and more important since the amount of information increases exponentially every second. According to Renolon.com, a website hosted by Renolon, a data and research company, over 867 million tweets(Twitter posts) are sent out daily on Twitter, and this can be further broken down to 10,033 tweets a second. [1] That's just on Twitter alone. Most of this data across the internet can be represented in text form and it is considered 'unstructured' or 'open-ended', as this data can be interpreted in any number of ways. With such insurmountable amounts of this data and the many ways it can be interpreted, it'd take too long for an individual or even a group of people to analyze and understand this data. As a result, having machines analyze and understand this data is much more ideal as they are fast, efficient, and don't tire, saving a lot of time and personnel. In this report, we will be discussing how text classification works and showcase its applications with two supervised learning models and one unsupervised learning model.

Introduction:

As we stated earlier, most shared data is completely unstructured and left for interpretation. Recent studies show that around 80% of all data being shared in text form is unstructured.[2] We can use machine learning to categorize this data efficiently by training it to relate patterns and classify text with a target value. More specifically, this “training” is the process of breaking down text and assigning a category or a tag to each individual piece. These assigned tags provide some insight on the overall meaning of the analyzed text. Machines, however, don’t understand the nuances of human communication, and so we must teach them to understand language structure and phrase structure.

Language structure refers to the way words are structured to properly formulate text and communicate a message. It is based on the eight-word types that could be in a sentence. These are nouns, pronouns, verbs, adverbs, adjectives, prepositions, conjunctions, and interjections. Phrase structure is a set of rules that describe the syntax of a sentence. Combined with language structure, machines can make notes of grammatical patterns to better understand the structure of a message that conveys a certain feeling or tone. These patterns can be based on the words being used, the order they are in, and even the relationship between them. For a machine learning model to even do this, it is given various tools and methods that separate text and extract features from it. One of these methods involves vectorization techniques. These vectorization techniques break down text into smaller individual components known as vectors to make data more digestible for a machine to analyze. Additionally, there are other methods for feature extraction such as Stemming and lemmatization, which involve simplifying a word and its many different forms of usage to its simple root or base word respectively.

Pre-processing:

Vectorization Techniques

As we stated earlier, Vectorization is a process where text is broken down into smaller individual vectors. There are various vectorization techniques available for NLP. Some examples of these various techniques are Bag of Words, Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, GloVe, and FastText. For the sake of simplicity and brevity, we only made use of Bag of Words and TF-IDF Vectorization techniques on my machine learning models. As a result, we will only be going over these two methods in detail.

Bag of Words Vectorization

Bag of Words Vectorization starts off with tokenizing the data. This is the process by which a sentence is separated into vectors, where each vector is a unique word. Those unique words are then taken and analyzed to form a type of vocabulary. Finally, a matrix is created where each unique word is assigned an identification number and given a weight value that represents how frequently it appears in the text. This weight value determines how "important" a word is for context based on how often it is used.

Count Vectorization is a form of Bag of Words Vectorization that we used when training my supervised learning models. The result of Count Vectorization often results in very large vectors, if used on real text data. While we will get very accurate counts of the word content of our text data, it won't provide us with any semantic or relational information. An example of Count Vectorization can be seen in Figure 1 below.

Count Vectorization, while popular, is very simple and this is its major drawback. This simplicity is an issue because some words are used more commonly than others but do not represent

or provide further context to a message. This mainly refers to “stopping words”, which are words that do not convey much information but are commonly used, such as articles and prepositions. This issue is resolved by other vectorization techniques such as Term Frequency–Inverse Document Frequency Vectorization.

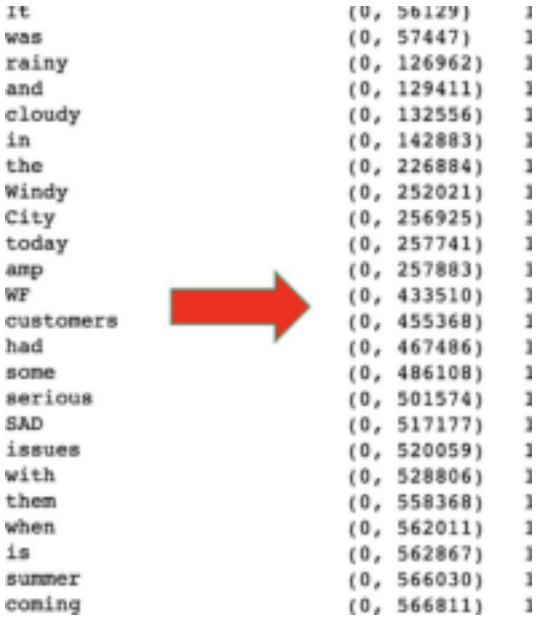


Figure 1: Count Vectorization example made when creating supervised learning models.

Term Frequency-Inverse Document Frequency (TF-IDF) Vectorization

Term Frequency–Inverse Document Frequency (TF-IDF) Vectorization is more of a numerical statistical technique that displays the importance of a word based on its frequency in the sample provided. Like Count Vectorization, it counts how frequently a word appears in a text. However, unlike the simplicity of Count Vectorization, TF-IDF makes use of a combination of different frequency equations to better get a word’s relevancy score. These frequency equations are Term Frequency (TF), Document Frequency (DF), and Inverse Document Frequency (IDF). Term Frequency calculates how frequent a word is in each piece of text. Document Frequency will calculate the number of texts that contain a specific word. Finally, Inverse Document Frequency calculates the inverse of DF. What this actually means is that Inverse Document

Frequency adjusts the weight of a word by lowering its weight if it appears too often between documents. The higher the DF of a term, the lower the IDF value will be. This eliminates the issue of "stopping words" as they are made less relevant because of how prevalent they are. The actual equations for Term Frequency, Document Frequency, and Inverse Document Frequency are as follows:

$$TF = \frac{Freq.\ of\ term\ in\ document}{Total\ \#\ of\ terms\ in\ the\ document}$$

$$DF = \frac{Total\ \#\ of\ documents\ with\ term\ X}{Total\ \#\ of\ documents}$$

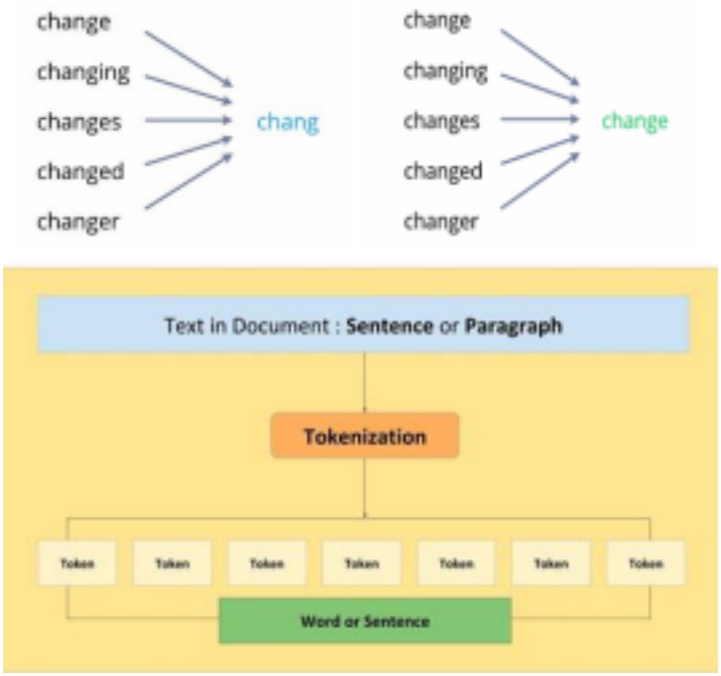
$$IDF = log(\frac{Total\ \#\ of\ documents}{Total\ \#\ of\ documents\ with\ term\ X})$$

Lemmatization, Stemming, and Tokenization

Besides vectorization techniques, there are other methods to simplify text for machines to be able to analyze it. Some methods involve making all text lowercase as well as removing symbols,

punctuation, stopping words, and any other form of unrelated data to the results desired. In addition to this, other techniques such as Stemming, Lemmatization and Tokenization can also be applied for better results.

Stemming is a process by which a word is reduced to its root form. This means that all forms of a word are replaced with the most basic subset of the word. Lemmatization, on the other hand, is a process by which all the different forms of a word are grouped together and reduced to their most common base form. This common base form then replaces all instances and forms of a word with its most found subset. Tokenization is the process of breaking down a given text into smaller units called tokens. Tokens can be individual words, phrases, or even whole sentences.



Figures 2,3 and 4: The above figures detail stemming, lemmatization, and tokenization respectively, and how they work.

Models Tested, Analysis, and Dataset Used:

The models we tested in this assignment are Logistic Regression, Naive Bayes, and a

Recurrent Neural Network (RNN) designed with TensorFlow. Both the Logistic Regression and Naive Bayes models are supervised learning models. When we were extracting features from my dataset, we only used Count Vectorization and TF-IDF Vectorization. The RNN, on the other hand, was much more tedious to use and required far more work and training.

The dataset we used is the Sentiment140 dataset.[3] This dataset was acquired via Kaggle, and it contains a library of about 1.6 million tweets. In addition to its massive library, it contains the tweet ID, the date and time of posting, the user handle of the poster, as well as a sentiment value, where negative statements have a value of 0 (zero) and positive statements have a value of 4 (four). Since we are focusing on Sentiment Analysis, i.e., how positive or negative a statement is, we are testing to determine how well the models we used can accurately classify a Twitter post as a positive or a negative comment. As such, we only focused on the text and target fields of the dataset.

	target	ID	Date	Flag	User	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
...	...	...	...	...	...	...
1599995	4	2193601966	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	AmandaMarie1028	Just woke up. Having no school is the best fee...
1599996	4	2193601969	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	TheWDBoards	TheWDB.com - Very cool to hear old Walt interv...
1599997	4	2193601991	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	bpbabe	Are you ready for your MoJo Makeover? Ask me f...
1599998	4	2193602064	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	tinydiamondz	Happy 38th Birthday to my boo of all time!!! ...
1599999	4	2193602129	Tue Jun 16 08:40:50 PDT 2009	NO_QUERY	RyanTrevMorris	happy #charitytuesday @theNSPCC @SparksCharity...

Figure 5: Preview of the Sentiment140 dataset

Logistic Regression Model

Logistic Regression is the first model we tested. Logistic Regression is a binary classification, and this model is a supervised learning model originally used in the field of statistics. We chose it mainly because we have experience with it using it in class and because it's one of the

most used classification models. We first imported my dataset, specifically choosing the text and sentiment value fields. We then split the dataset such that 80% is to be used in training and the remaining 20% in testing. The dataset was now read, and we started to generate the models.

We generated the model where Count Vectorization was used. After training and testing, the resulting accuracy was 0.7983125. The resulting classifications of this model are shown in Figure 6. As shown below, there are 126,496 true positive classifications and 129,606 true negative classifications. With these results and the accuracy score, we believe it is safe to say that the Logistic Regression model did very well in classifying whether the tweets were positive or negative in sentiment.

We also tested the TF-IDF vectorization technique with the Logistic Regression model, and after training and testing, it performed similarly to Count Vectorization, albeit somewhat better. The resulting accuracy score was 0.802515625. As shown in Figure 7, there are 131,229 true positive classifications and 116,348 true negative classifications. Since this accuracy is already higher than the well-performing Count Vectorization, this model has done very well itself, and it can be said that TF-IDF reduced the effectiveness of "stopping words" and this benefited the Logistic Regression model.

		Actual Target	
		0	**4**
Classified Target	**0**	126496	33825
	4	30073	129606

Figure 6: True positive and true negative classifications of the Logistic Regression model using Count Vectorization

		Actual Target	
		0	4
Classified Target	0	131229	29092
	4	43331	116348

Figure 7: True positive and true negative classifications of the Logistic Regression model using TF-IDF Vectorization.

Naïve Bayes Model

The Naïve Bayes model is one of the most popular models for text classification. This is mainly because this model performs well with high-dimensional data which allows it to better extract features from a provided dataset. Naïve Bayes is known for having a higher bias and lower variance, making it better than Logistic Regression as this model requires less data to analyze. The Naïve Bayes model is based on a culmination of classification algorithms that follow Bayes' Theorem, which is shown below:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

As we did with Logistic Regression, we imported the dataset, selected the fields we wanted to analyze, and split the dataset between 80% training data and 20% testing data. We also tested this model using Count Vectorization and TF-IDF Vectorization. For Count Vectorization, we had a final accuracy of 0.780703125, which was lower than the Logistic Regression model. The resulting classifications of this model are shown in Figure 8. As shown below, there are 131,110 true positives and 118,715 true negatives from this test, which is also less than the Logistic Regression

model. Despite performing a little worse than the Logistic Regression model, the Naïve Bayes model still did reasonably well. For TF-IDF Vectorization, the resulting accuracy was 0.773678125. Something we found interesting here is that TF-IDF performed a little poorly compared to Count Vectorization. This could mean that based on the type of model used, the type of vectorization may be better or worse. The classification results are shown in Figure 9 below and show 131,229 true positives and 116,348 true negatives. Although the accuracy has lowered, the overall accuracy score is still very close to 80% which means that the model is still fairly accurate.

		Actual Target	
		0	**4**
Classified Target	**0**	131110	29211
	4	40964	118715

Figure 8: True positive and true negative classifications of the Naïve Bayes model using Count Vectorization.

		Actual Target	
		0	**4**
Classified Target	**0**	131229	29092
	4	43331	116348

Figure 9: True positive and true negative classifications of the Naïve Bayes model using TF-IDF Vectorization.

Recurrent Neural Network (RNN) Deep Learning TensorFlow Neural Network

The TensorFlow neural network that we designed is an unsupervised learning model. As a result of this, this model took the most training and data breakdown out of all the models used. We had to first clean up the data. What we mean by this is that we removed stopping words, punctuation marks, duplicate words in the text being analyzed, email addresses, URL links, and numbers. After doing that, we further broke down the data using tokenization, stemming, and lemmatization to make it even easier for the TensorFlow neural network to classify the data. We then began to split the dataset as we did for the other models, 80% of the data for training and 20% of it for testing.

The neural network is made up of an input layer that receives the dataset, seven hidden layers to understand the data, and an output layer to output the results. Once all this was done, the time came to test the dataset. We used multiple iterations of the neural network, where each iteration builds off the previous one to improve the model over time. The final accuracy of this model was 0.745550. Figure 10 below shows all the iterations my neural network worked through. This accuracy was the lowest out of all the models and methods we had used. Some things that we can improve upon are requiring more iterations, better training, testing set ratios, or even including more feature extraction methods. These things should increase the accuracy of RNN and better reflect how it performs in terms of text classification.

```
In [46]: model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])

In [47]: history=model.fit(X_train,Y_train,batch_size=80,epochs=6, validation_split=0.1)

Epoch 1/6
WARNING:tensorflow:From C:\Users\chaud\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTe
nsorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\chaud\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

360/360 [==============================] - 114s 312ms/step - loss: 0.5990 - accuracy: 0.6662 - val_loss: 0.5415 - val_accuracy:
0.7234
Epoch 2/6
360/360 [==============================] - 113s 315ms/step - loss: 0.5106 - accuracy: 0.7547 - val_loss: 0.5257 - val_accuracy:
0.7300
Epoch 3/6
360/360 [==============================] - 113s 314ms/step - loss: 0.4933 - accuracy: 0.7634 - val_loss: 0.5248 - val_accuracy:
0.7344
Epoch 4/6
360/360 [==============================] - 112s 312ms/step - loss: 0.5135 - accuracy: 0.7511 - val_loss: 0.5271 - val_accuracy:
0.7406
Epoch 5/6
360/360 [==============================] - 113s 313ms/step - loss: 0.4905 - accuracy: 0.7689 - val_loss: 0.5263 - val_accuracy:
0.7381
Epoch 6/6
360/360 [==============================] - 108s 300ms/step - loss: 0.4663 - accuracy: 0.7801 - val_loss: 0.5394 - val_accuracy:
0.7387

In [50]: accr1 = model.evaluate(X_test,Y_test)

250/250 [==============================] - 14s 55ms/step - loss: 0.5289 - accuracy: 0.7455

In [51]: print(accr1[1])

0.7455000281333923

In [ ]:
```

Figure 10: Testing Iterations of TensorFlow neural network.

Decision Tree

The build of a decision tree starts with a root node, which does not have any incoming branches. That is the top node in the tree where it starts the test. Then we have the decision nodes which are also called the internal nodes. This is where the test of the specific features is taking place and the decision at each node leads to a different path based on the test. Next, we have the branches that make the tree. The branches are used to connect each node together and the different outcomes to come. Lastly, we have the leaf nodes. This is the final node on the tree and the overall final decision of the whole tree.

There are steps that decision trees have to go through in order to get to the final decision. One of these steps is selecting the best features. The algorithm checks the different features and then selects the one that provides the best split. When splitting the decision the tree uses the dataset that

was given and observes the overall features. For example, if there is a dataset that has to do with age the decision tree might split them up into ages greater than forty-five and ages less than forty-five. There is also a recursive process that plays an important role. The process is repeated for each subset which creates additional decision nodes and branches until a reason to stop matches. There is also a process called pruning which is optional. This is a post-processing step to clean out any unnecessary branches or nodes that do not apply a lot to the performance of the tree.

The outputs of the code provide a comprehensive overview of the decision tree model's performance on a randomly selected subset of 10 rows from the dataset. The accuracy on the subset reveals the percentage of correctly predicted instances, giving you an initial impression of how well the model generalizes to this specific portion of the data. The confusion matrix takes the evaluation a step further by breaking down predictions into true positives, true negatives, false positives, and false negatives. This matrix offers insights into specific areas where the model excels or struggles, aiding in the identification of potential biases or areas for improvement. Additionally, the display of 10 random rows alongside their actual and predicted values allows for a qualitative assessment of the model's predictions on individual instances, offering a closer look at its performance at a granular level.

```
Accuracy on subset: 100.00%
Confusion Matrix on subset:
[[2]]
                                                     Text  Target  Predicted
541200             @chrishasboobs AHHH I HOPE YOUR OK!!!       0          0
750      @misstoriblack cool , i have no tweet apps  fo...       0          0
766711   @TiannaChaos i know  just family drama. its la...       0          0
285055   School email won't open  and I have geography ...       0          0
705995                              upper airways problem       0          0
379611         Going to miss Pastor's sermon on Faith...       0          0
1189018           on lunch....dj should come eat with me       4          4
667030    @piginthepoke oh why are you feeling like that?       0          0
93541      gahh noo!peyton needs to live!this is horrible       0          0
1097326  @mrstessyman thank you glad you like it! There...       4          4
```

Figure 11: Matrix on Subset

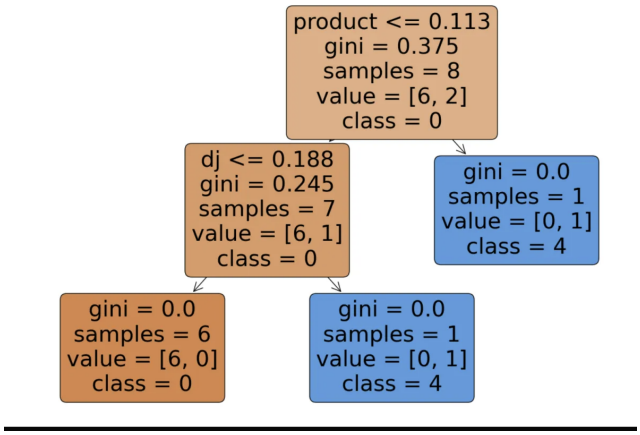


Figure 12: Decision Tree

The decision tree plot visually represents the decision-making process of the model, illustrating the hierarchy of features and their importance in predicting the target variable. Each node in the tree represents a decision based on a specific feature, and the branches depict possible outcomes. The color and size of nodes convey information about the predicted class and the number of samples, respectively.

Summary:

Text classification is one of the fundamentals of Natural Language Processing. It allows machines to be able to read and understand how humans communicate with each other via text. It is vital as humans can't do it ourselves in as efficient or timely a manner. For machines to be able to classify text, we must first break down the data and to expedite this we can remove unnecessary words and details. In addition to this, there are methods to break down the text even further. There's tokenization, which breaks down the text into tokens, stemming and lemmatization, which break down words and the various forms they are used into their common root and base word respectively. There are also vectorization techniques like Count Vectorization and Term Frequency–Inverse Document Frequency which count how many times a word appears in a text or document and then

assign a weight value to them.

The models that we tested on are Decision Tree, Logistic Regression, Naive Bayes, and a Recurrent Neural Network designed with TensorFlow. The Logistic Regression and Naïve Bayes models are supervised learning models and we used Count Vectorization and TF-IDF Vectorization to classify the data and then compare the results. The methods and models performed similarly. The Neural network we designed is an unsupervised learning model and thus requires the most work. We first cleaned up the data, and removed stopping words, URLs, emails, and numbers and then applied tokenization, stemming, and lemmatization to further break down the data even further. Overall, the Logistic Regression model with TF-IDF Vectorization did the best with an accuracy score of 0.802515625. The neural network performed the worst, with an accuracy score of 0.7457500100135803. This may improve with better training ratios and feature extraction. For Logistic Regression, TF-IDF Vectorization performed better and for Naïve Bayes, Count Vectorization performed better. What this tells me is that certain methods may work better or worse with certain models.

Sources used

1) How Many Tweets per Day 2022 (Number of Tweets per Day) (renolon.com)

2) Structured vs Unstructured Data: What is the Difference? | AltexSoft

3) Sentiment140 dataset with 1.6 million tweets | Kaggle