Classifying Tweets as Football or Soccer using Natural Language Processing and Random Forests

Abstract

This project is focused on the fundamentals of natural language processing (NLP) and using random forests as a classification model for text analysis. More specifically, our project involves preprocessing large, text datasets and representing this data as a feature matrix using a bag-of-words model. Then, using this feature matrix to create a decision tree classifier and later, a random forest. After finishing this exercise, you will have a clear understanding of the importance of preprocessing text and the methods used to do so. Furthermore, you will be able to understand the what a decision tree is and how it extracts the most important features from a dataset. Lastly, you'll learn about the overfitting in decision trees, and see how Random Forests overcome overfitting simply by averaging the results of an ensemble of decision trees. We used the concepts explained in this final project and implemented a text analysis program which classifies whether a tweet is talking about the National Football League (NFL) or the English Premier League (EPL) or broadly speaking: American football or soccer. Using our own dataset of tweets scraped from twitter, we implemented a binary classifier using random forests with an error rate < 4%.

Natural Language Processing (or NLP) is a subfield of artificial intelligence with the purpose of understanding, interpreting and manipulating language. NLP addresses the problem that computers are unable to interpret language like humans can. We see NLP everyday in digital personal assistants like Alexa, in text prediction like autocomplete, and translators like Google Translate. Applications like these weren't formed in a vacuum. In fact, language processing dates back to the 1930s with a russian translation machine. The first chatbot was developed in 1966 by MIT professor Joseph Weizenbaum which had scripted responses and often mimicked user inputs. In 2006, IBM's Watson beat a panel of Jeopardy contestants using NLP. Today, machine learning is instrumental in NLP applications and we will use a machine learning technique called random forests to accurately label whether or not a tweet is about profession American football (NFL) or profession soccer (EPL).

The goal of our classifier is to use large data sets to accurately make predictions. The first step is getting a large set of text data, which in our example is tweets. The first thing we do to process this data is a method called ***tokenization***. Tokenization is the process of breaking up text into words, phrases, symbols, or other meaningful elements called tokens in order to keep track of the word frequency. For our example, the tokens will be words found in tweets.

However, raw text data can be large and can contain lots of noise which negatively affects the model. This noise includes variations in case, word forms, common words, punctuation and special characters. Consider the tweet: *"He scores a GREAT goal, again!!!!!!"* For a classifier that's trying to distinguish between NFL and EPL tweets, the tokens *He, a,* and *again* as well as the comma and exclamation points contribute no meaningful information for classification. These are ***stop words*** and should be removed from the tweet dataset along with punctuation, leaving only the relevant tokens *scores, GREAT,* and *goal.* During tokenization, the data will converted into a uniform case which is commonly lowercase. This would change the token *GREAT* into *great* and ensures that any case variation does not result in different tokens.

Another common variance in tokens of the same word occurs with different forms of the same root. For example the tokens *scores, scored,* and *scoring* should all be counted as the same token. One naive method to overcome this variance is ***stemming*** which removes suffixes from words. After stemming, the tokens in the example above would all be *scor*. While this is usually effective enough, its logic is rather crude. What we can do instead is **lemmatization**

which takes parts of speech into account like verbs, nouns, and adjectives. So instead of crudely chopping off the suffix and returning *scor*, lemmatization will attempt to return the base word which in this case would be *score*. These are the most important and most commonly used steps in preprocessing large text data. Additionally preprocessing often includes removing infrequent words and extra-long words as they do not provide much meaning and take up unnecessary space and processing time.

Now that we have our text data all preprocessed, we need to represent it as feature vectors so that we can train our classifier. There are a couple of methods to do this. The most common and also the one we've chosen is a Bag-of-words model (BoW). It's a matrix where the rows are the documents or tweets in our example and the columns are distinct tokens from the entire corpus. So for each tweet, we count the instances a token appears in it.

	field	*score*	*kick*	*touchdown*	*great*	*goal*
Tweet 1	1	1	0	1	1	2
Tweet 2	1	1	0	0	0	1
Tweet 3	0	1	0	2	0	0

Bag of Words model

Decision Trees

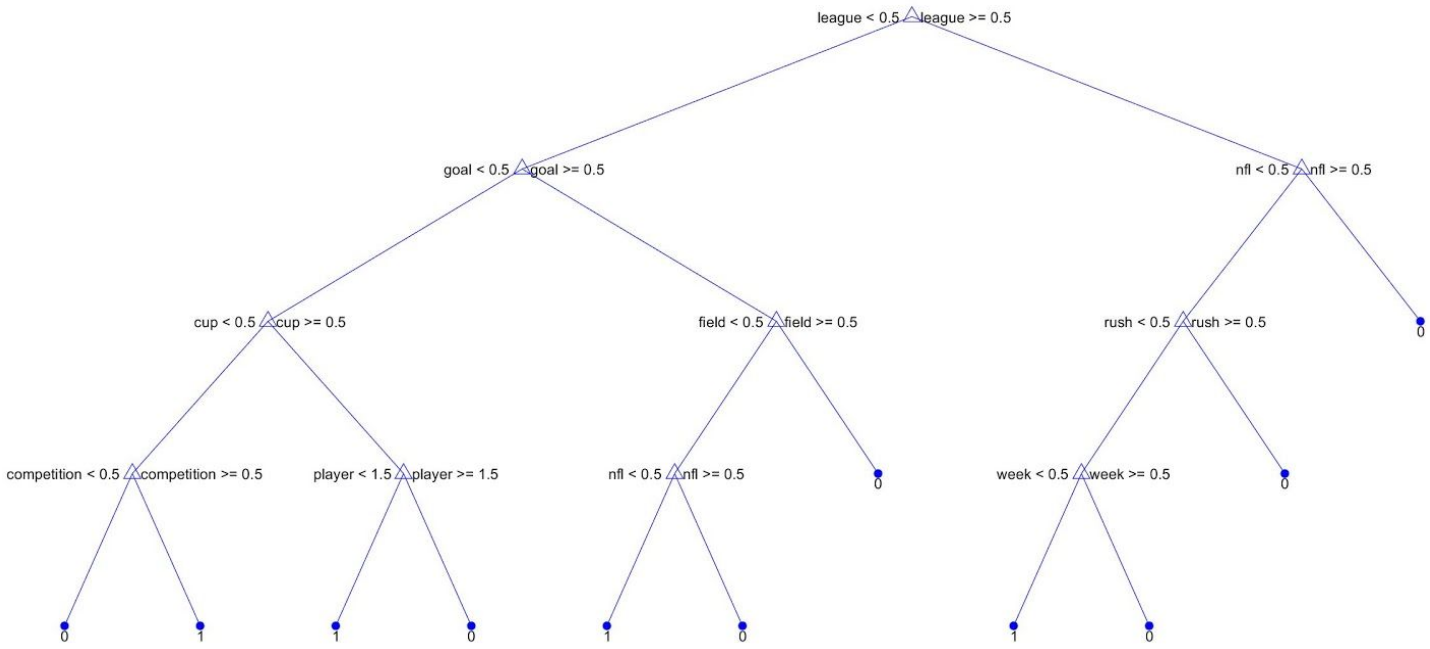
One way to design a classifier is using a ***decision tree***. These are binary tree structures where each node represents a feature and has a corresponding threshold. To test new vectorized data, starting at the root, the model checks if the corresponding feature of the new data is greater or less than the threshold of the node. From there, it moves to the left or right node depending on the result. This process is recursively repeated until a leaf node is reached which contains a label, classifying the test data. So, to arrive at an estimate, we used a series of questions, with each question narrowing our possible values until we were confident enough to make a single prediction.

Choosing the most important features:

Matlab's decision tree implementation uses the **CART Algorithm** (Classification and Regression Trees). To obtain feature importance, the algorithm calculates the Gini Impurity which is a cost function used to evaluate how good a split in the dataset it. If the Gini score is 0, we have a perfect split. On the other hand, an impurity score of 0.5 is the worst possible score.

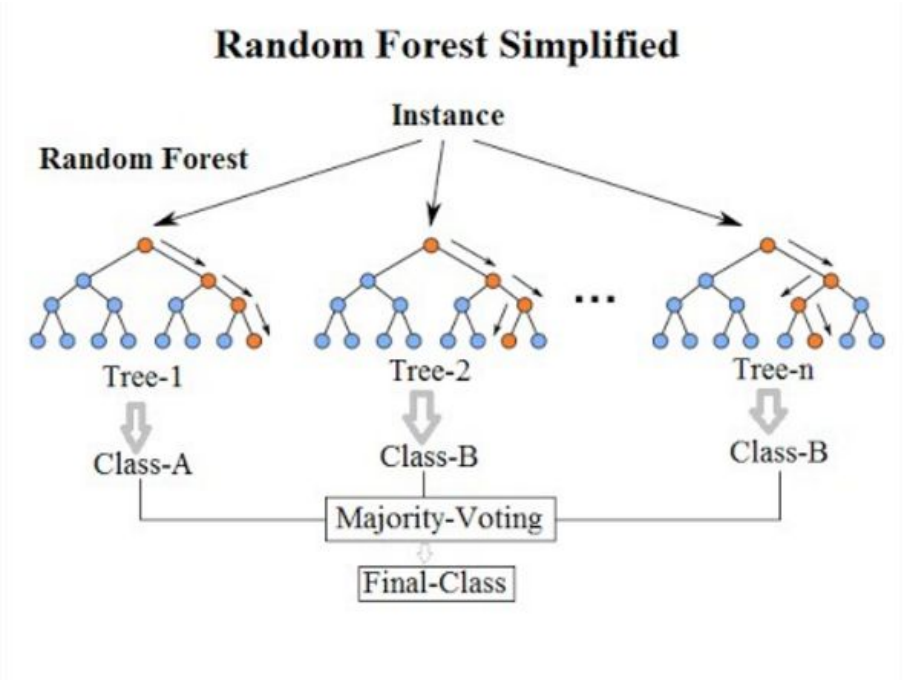
$$Gini\ Impurity\ =\ \sum_{i=1}^{C} -f_i(1-f_i)\ :\ f_i\ is\ the\ frequency\ of\ label\ i\ at\ a\ node\ and\ C\ is\ the\ number\ of\ unique\ labels$$

A decision tree trained with our dataset



Random Forest

Our prediction using one decision tree is probably wrong. There are too many tokens to fit into the tree. Hence, decision trees tend to overfit data. That means the decision tree classifies the training data perfectly, ending up with branches with strict rules of sparse data. Thus this effects the accuracy when predicting samples that are not part of the training set. ***Random Forests*** overcome this overfitting. Random forests are collection of several decision trees into a single model. Each single decision tree is trained using a random subset of the original data. What this does is average out the inconsistencies of overfit trees, resulting in a prediction that is closer to the right prediction. In our example, where our model is assigning a label to a tweet rather than calculate a range, the selected label would be the majority.



Warm up

1. Create a Bag of Words model for the following tweets:
   Note: You will need to perform all preprocessing steps i.e. lowercase, remove stop word/punctuation and stemming.

Tweet 1	Messi scored an amazing goal today!!
Tweet 2	Today we saw Jake score a touchdown, and score a field goal.
Tweet 3	jake is not as amazing as messi!

2. Open *warmup.m* and the run section labeled **Question 2**. Answer the following question: You are given word-clouds of 2 datasets. Assuming your datasets are classifying between the NFL and EPL, which dataset would be better and why? What did that dataset do differently?

3. Using your bag-of-words from warm up question 1, classify the 3 tweets with the simple decision tree below. Comment on the label of Tweet 3. Do you think it is accurate?

Main Activity

Note: *main.m* processes the dataset and trains a random forest classifier with it (the program might take upto 30 sec on some devices). You will need to run the file to answer the next questions. The Label 0 is NFL(football), and 1 is EPL(soccer)

For questions 1-3, run the section named Question 1-3 of main.m

1. After running section 1 of *main.m*, you will be greeted with two windows. For now, focus on the decision tree. Maximize it so that you can clearly see the features. This is one of the decision trees of the random forest.
    a. According to the decision tree, what is the most important feature in deciding if a tweet is related to NFL or EPL?
    b. What is the shortest path to a leaf node (label)?
    c. What do these features you pass over on the shortest path represent for the classifier?

2.
    a. The value of **numTrees** is initially set to 2. Run it for this condition. Now, change the value of **numTrees** to 10, 25, 50, and 100. Comment on the performance of the classifier for each change you make to numTrees.
    b. If you were designing the classifier, what number of trees do you think would be optimal considering performance time and error rate?

3. NumSplits is the maximum allowed number of decisions in each decision tree. For example if NumSplits is set to 1, then the decision tree will consist of the root node, a decision query and the leaf nodes.
    a. Change the value of **numTrees** to 2 and **numSplits** to 1. The error rate hovers around the 50% mark. Why do you think this is?
    b. Keep **numTrees** as 2 and set **numSplits** as 100. What is the error rate now and how is the performance affected?

4.

a. Run the section named ***Question 4a***. There are two bag-of-words using the same data will be displayed in the form of word-clouds. Their only differences come in a preprocessing technique. Can you identify the difference in preprocessing methods used?
b. Now run the section named ***Question 4b***. Two models will be trained and tested using the stemmed and lemmatized datasets in part a. Charts for the error rates of both models will be displayed. Compare the smallest error rate for both models. What is the difference between them. If lemmatization is computationally costly, is it worth it to lemmatize the data?

(If you would like classify some tweets yourself, try running the final section in *main.m* and putting the tweet in the variable *test_tweet* ! If it fails, mention the tweet in the comment section along with NumTrees and NumSplits. But please don't dock points :))

References

NLP

https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.removewords.html

Decision Trees & Random Forests

https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

https://medium.com/@srnghn/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

https://www.mathworks.com/help/stats/decision-trees.html

https://www.mathworks.com/help/stats/treebagger.html

Warm-up Solutions:

Question 1:

	messi	score	amazing	goal	today	saw	jake	touchdown	field
Tweet 1	1	1	1	1	1	0	0	0	0
Tweet 2	0	2	0	1	1	1	1	1	1
Tweet 3	1	0	1	0	0	0	1	0	0

Question 2:

Dataset 1 is better than Dataset 2 because it features more relevant words like season, game, goal, etc. while Dataset 2 has mostly common words like the, in, of, a, etc. Dataset 1 removed stopwords, while Dataset 2 did not.

Question 3:

Tweet 1: Soccer
Tweet 2: Football
Tweet 3: Soccer

The nodes of the decision tree do not have any features that were extracted from Tweet 3, so the label of the tweet is fully accurate.

Main Activity Solutions:

Question 1:

a. The most important feature will be the feature in the root node. This could be different on each run because each tree has a random subset of the entire dataset.
b. This will be the depth of the shallowest leaf node (should be around the 1st, 2nd or 3rd level if the root is level 0).
c. The features you pass are the features that the model thinks are strongest. If the data contains all of these features, then the model can quickly classify it.

Question 2:

a. The performance when numTrees = 2 is quite poor. It often will hover around 0.42 - 0.5. This is slightly better than a guess.
   The performance when numTrees = 10 is much better than before. It will hover around 0.15.
   The performance when numTrees = 25 is slightly better than 10. This will improve to around 0.1.
   The performance when numTrees = 50 is still better than before. Again, it improves to 0.05.
   The performance when numTrees = 100 is practically the same as 50. This will often be around 0.045.
b. Setting numTrees to 50 would optimal as it halves the number errors from 25, but sees little improvement when jumping to 100.

Question 3:

a. With numTrees set to 2 and num splits to one. We will have the average of 2 decision trees each having a 50% error rate as the tree is just a root node and two leaf nodes.
b. The performance greatly improves. This makes sense, since each tree can now make up to 100 decisions before it makes a classification. However having such a low numTrees will increase the error rate.

Question 4:

a. The difference between the datasets is that Dataset 1 used stemming while Dataset 2 used lemmatization. This can be seen because Dataset 1 has

truncated words like *sinc* and *leagu* while Dataset one had full english words like *since* and *league*.

b. The difference between the error rates is often around 0.01 where the lemmatized dataset is better. However, considering the only slight improvement after lemmatizing the dataset, it does not seem worth it to spend processing time and power.