Scraping and Classifying Sports Tweets Using Scikit-Learn

Jazli & Umair

December 2022

1 Goal

The intent of this project is to scrape sports tweets using Twitter's API. Using these tweets we intend to teach a machine learning model to be able to identify the sport a tweet is talking about. Our process is as follows: First we intend to scrape the tweets giving us a large json file. Second, we plan to add another key-value-pair that will contain the actual sport the tweet refers to. This will be done by taking advantage of twitter's context annotations. These context annotations provide insight on the actual sport the tweet is discussing. Next, we will convert the json file into a csv to make the machine learning process easier. Then, we will clean the tweets within the csv using a variety of methods. We will need the data to be split into then training and testing data sets to train our machine learning model. Finally, we will cross validate a couple classifying machine learning models to determine the most appropriate one for our use case.

2 Methodology

1. Acquiring the Data

   We acquired the data using twitters api the code can be found within the TwitterScrape.py. This file is responsible for looking for sports tweets specifically tweets that relate to the following sports: Soccer, Hockey, Basketball, American Football, and Baseball. We scraped 2000 tweets and stored them in the file SportsTweets.json.

2. Identifying the Sport

   Once the tweet data has been saved into a json there is a lot of data that can be used. Our approach for determining the actual sport being discussed was using the context_annotations field within the json file. This field gives context clues of what the tweet refers to, one of which being sport. We used this field to create a new key value pair that contained the sport. This was done within SportClassification.py.

3. Converting to CSV

   Now that the required information is put into the json file we decided to get rid of the fluff and focus on the important parts of the data. That being the tweet and the sport being discussed. We converted the json file into a csv with only two columns one for the tweet and one for the sport. While converting we also removed any emojis within the tweets.

4. Data Cleansing

   Now that our data was neatly put into a csv file we began cleaning the data. The cleaning operations we decided to perform on the tweets were:

   - Removing @ handles
   - Removing hashtags
   - Removing URLs
   - Making the tweets lower case
   - Removing the RT keyword to signify retweets
   - Removed parentheses, exclamation mark, and question mark
   - Expanded contractions

5. Model Training

   Since we are using Python to train our model on, we decided to use scikit-learn as it is the most intuitive for performing classification tasks as TensorFlow would work better for using neural networks but we wanted to use a more straightforward machine-learning model.

   Before creating any models we need to pre-process the data into a format that the model can read. First we had the categorical variable sports where we needed to convert into numeric values. These were encoded using the Label Encoder from scikit-learn's preprocessing library. These are the numbers each correspond to:

   - 0: Corresponds to American Football
   - 1: Corresponds to Baseball
   - 2: Corresponds to Basketball
   - 3: Corresponds to Hockey
   - 4: Corresponds to Soccer

   This label was then assigned to the y/output variable to be observed.

   In order to analyze these tweets and break them down for the computer to understand we needed to use Natural Langauge Processing or NLP for short. We decided to do this using scikit-learn although we realized that TensorFlow, or even PyTorch might be better for this process.

   We decided to use the method TF-IDF for the model to use via the Tfidf Vectorizer given by the scikit's feature extraction library. TF-IDF will find

a word's importance to the output then return a weighting factor for each word that appears in its dictionary. The "TF" stands for "term frequency" which refers to the frequency of a term in the document. "IDF" refers to the "inverse document frequency" which looks at the logarithmic inverse proportion of documents where the term occurs across the whole corpus. The TFIDF score multiplies both the TF and IDF scores, to get a final weighted value.

It was given the argument (1, 2) for the ngram_range parameter to allow for the use of n-grams, specifically, unigrams and bigrams when it creates the vector holding the sequences of words.

Then a variable called X_fit was created which held the fitted and transformed values from the TfidfVectorizer.

To find our the best model to be used for this analysis we performed cross validation across five different models and decided to compare their averages.

The Stratified K Fold cross validation technique was used to compare these models as it is perceived one of the most accurate k-fold techniques We used 10 folds as the n_splits parameter since this is generally accepted as the most optimal amount of folds needed for cross validation.

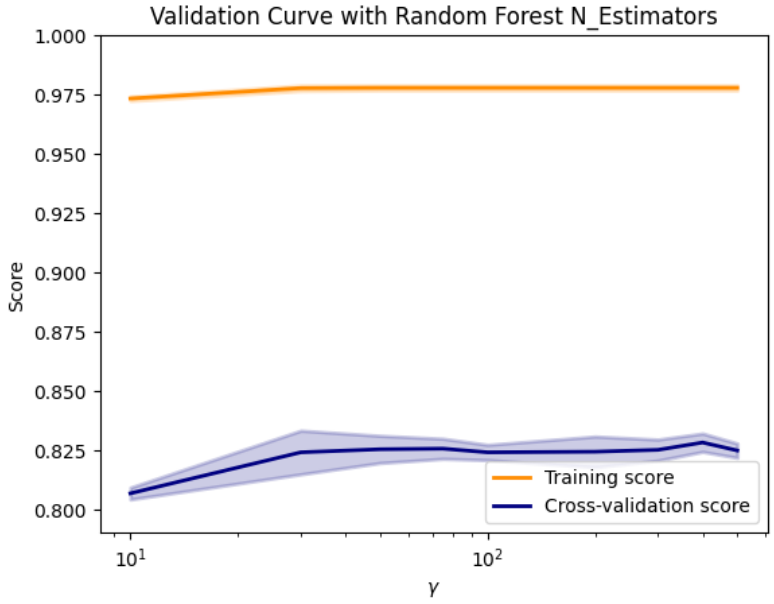
We tested the Linear SVC model, the RBF Kernel SVM model, the K-Nearest Neighbors model, Logarithmic Regression model, and the Random Forest model using the cross_val_score passing in the stratified k fold object which used the training set.

A function was created to find the highest performing classifier and usually the best performing was the **Random Forest** model.

Afterwards, we decided to fine tune the hyperparameters by using the validation_curve function to find the best arguments for n_estimators, max_depth, max_samples_split and finally min_samples_leaf to find the best performing values to increase the accuracy of the model.

For each hyperparameter, we used the validation_curve function on each and calculated their mean which also used the training set.

Here is an example of a graph that shows these scores that were assessed for the n_estimator hyperparameter. The solid blue line at the bottom shows the actual mean score and the outer purple bubble surrounding the line represents the standard deviation that is plus minus from the calculated mean.



We then created a new RandomForestClassifier object by passing in the hyperparameters that performed the best in the validation_curve tests where it was finally used to train itself on the data

Using that classifier it was then trained on the training data set again but then it was used with the **untouched testing data set** to generate predictions for the sport corresponding to the tweet.

3 Results

To assess the results we used the classification_report function that compared the predicted sport (output) result of the tweet was to the actual real sport of the tweet.

A rough average of 83% was achieved after several tests of the model.

	precision	recall	f1-score	support
0	0.84	0.66	0.74	191
1	0.96	0.73	0.83	71
2	0.78	0.98	0.87	461
3	0.98	0.70	0.81	82
4	0.89	0.67	0.77	163
accuracy			0.83	968
macro avg	0.89	0.75	0.81	968
weighted avg	0.84	0.83	0.82	968

To conclude the project, it was an overall success as an accuracy of 83% is decently accurate for the project.

Some improvements we could have used was to scrape more tweets in order to create a greater data set for the model to be trained on. Another improvement we could've performed would be to use TensorFlow or PyTorch where the former especially specializes in Neural Networks where this could be used to perform Natural Language Processing better. This could greatly increase the accuracy of the model. A different technique other than TF-IDF could have been used like the bag of words or word2vec technique.

Future projects we could perform would be to create a model that can formulate a tweet that corresponded to each of those sports maybe via TensorFlow. We could also try to classify tweets of other sports as well.