Brandon Huegerich

May 8th, 2019

ITCS 4156-001

800909207

Prepping and Classifying the GAP Dataset

**Abstract**

The Gender Ambiguous Pronouns or GAP dataset was provided to Kaggle community members by Google AI Language as a part of a Kaggle competition. The goal of this competition is to provide a model that is fair and accurate at assigning the correct noun to pronoun.

My solution to this problem will use two main libraries to SpaCy and scikit-learn. SpaCy is where we will do all our natural language processing or NLP. SpaCy is the natural language processor or NLP that I choose to work with to convert the text into meaning. I used scikit-learn in order to pick a classification model without having to implement the whole model from scratch.

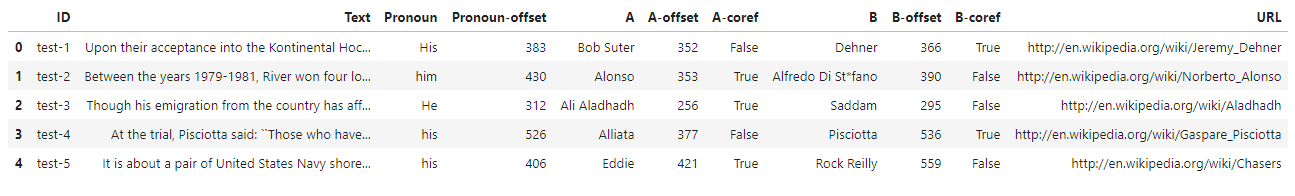
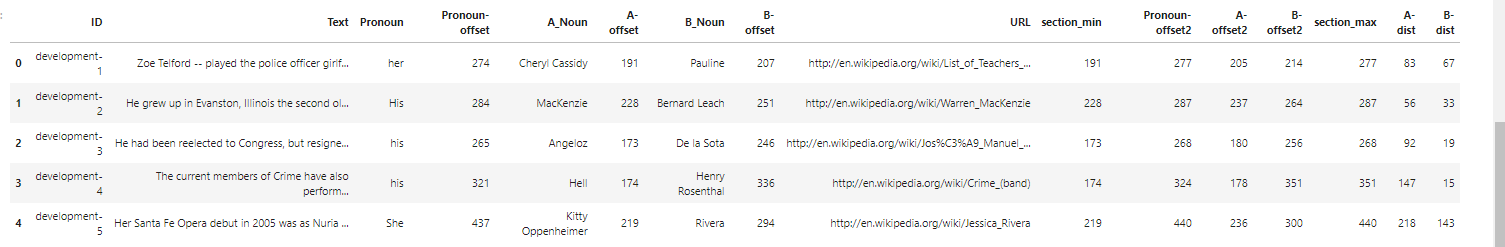
**Introduction**

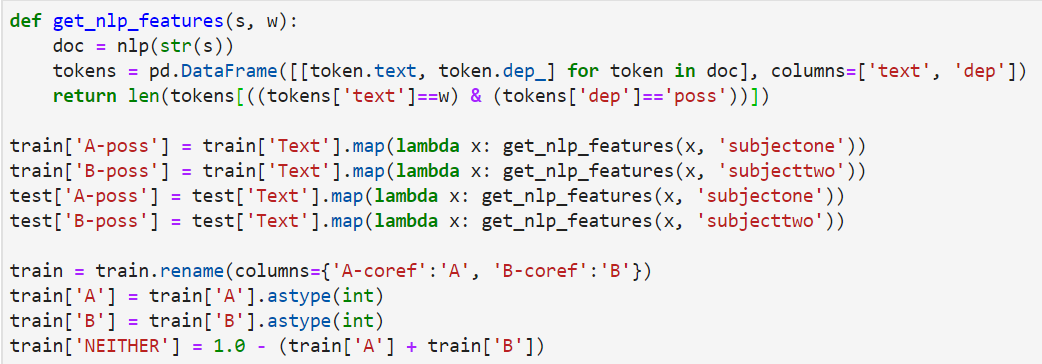
First, let’s talk about why I choose to do SpaCy over the other NLPs or pre-trained models. When starting out the project I was using BERT the pre-trained model released by Google AI language. After I started fine tuning the model, I realized how long it took to change the model and the large size of the model.

So, what is SpaCy? SpaCy is a free, open source library to help developer work with large amount of text and figure out more information about the text. While SpaCy doesn’t support a ton of languages our dataset is taken from English Wikipedia pages meaning everything will be in English which is supported. The benefits are that it is the fastest NLP framework out of the bunch. It is also easy to learn because each task you would want to do has a single highly optimized tool. We will use SpaCy for extracting text, tokenization, and linguistic dependency

After the text has been processed, we will then split our dataset into a test-train split and then select a classification model. This is where scikit-learn comes in they have a wide array of estimators to select from. In the methods selection we will look at the different options to selecting a method for the right use case through the flow chart that scikit-learn provides. Then, we will use the estimator that provides us the most accurate results in a quick manner.

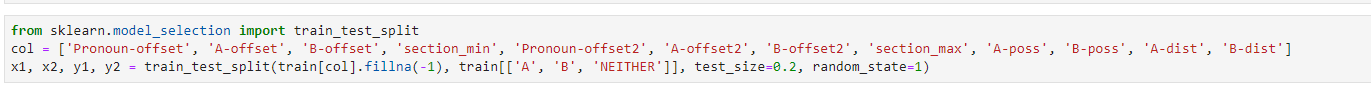
**Methods and Results**

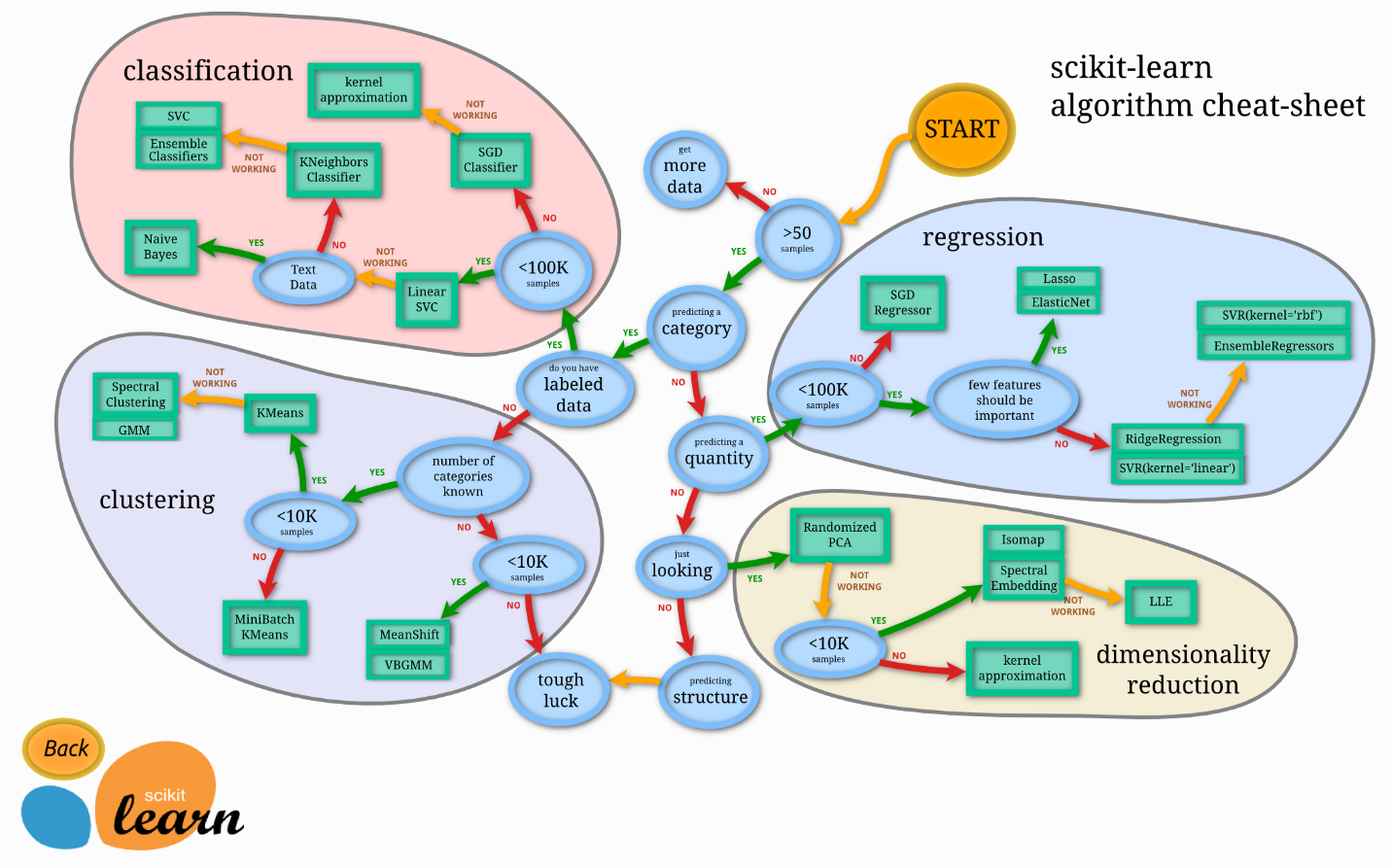
First, we will start with extracting the important information from our dataset. During this processing phase we will be working with the GAP test, train, and validation set provided. We have one main function that will extract the main features of the dataset and make it easy for our NLP and classifier to train with. This is the current table before changes and this is the function that will extract new features out of the dataset.  After we do feature extraction this is what the table looks like. 

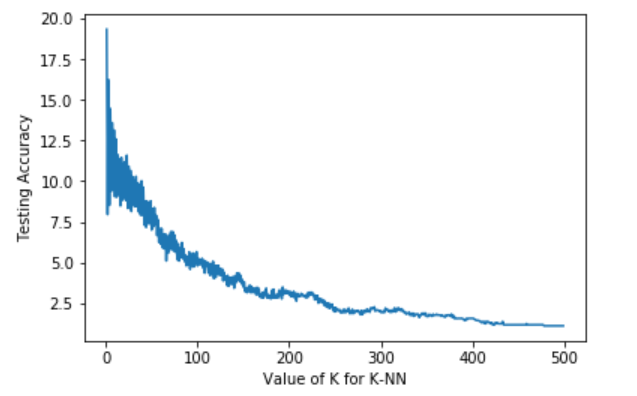
Now that we have extracted some more features from the dataset let’s move on to the NLP portion of the code. Given some string s where this string is a logical sentence or sentences. The function nlp(s) takes that string and creates a “token” for each word. Each word or token now holds a lot more information than before. The two that we will be looking at is token.text and token.dep\_. token.text is self-explanatory it returns the text of the token. While token.dep\_ is the linguistic dependency of each word. This will be the key ingredient when it comes to name-entity relationships. 

Now that we know what tokenization and what token.dep\_ and token.text the last line in the method is where we search for where we search for our noun and it adds one if that noun is found to be possessive. This information is stored in A-poss, B-poss for both the training set and test set.

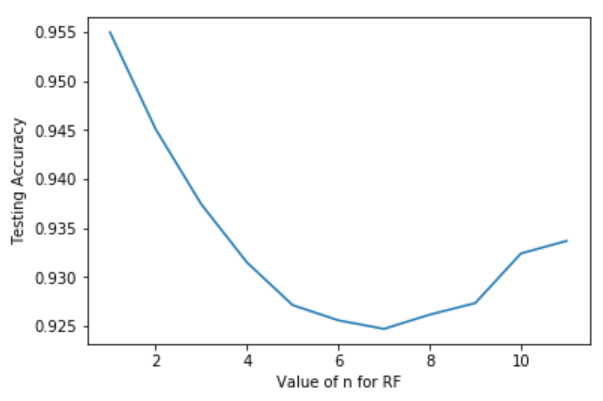
We have now prepped the data for training and extracted new information. It’s time to do the test-train split and pick our classification model. This is where sciKit-learn. So, we have two datasets right now test and train. We will split the trainset and create the model and fit it to our training set. Then we will use the model that was fitted to predict our test dataset. We are going to do a 20% test size. The code for our test-train split looks like the following:



Scikit-learn has provided this easy cheat sheet to follow when trying to pick out an estimator provided below. 

We already know our goal is going to be a classification task, but let’s follow through the paths provided. It is more than fifty samples. It is categorical. The data we have is labeled it less than a hundred thousand samples. Now that we have converted all our data to numbers with A- poss and B-poss all the data is now numerical. So, our first stop is seeing if K nearest neighbors(K-NN) is good enough. K-NN is what we call a lazy algorithm meaning it does not make any generalizations. It uses feature similarity rather so how closely a new point resembles the current trainset. K is the number of nearest neighbors you can see. We will be going iterating through 500 values of K. From this graph you can see the accuracy over these values of K. The accuracy of this graph is the log loss function. 

I wasn’t happy with a score of 1.5 – 2. Random forest builds multiple decision trees and merges them together for a more accurate and reliable answer. Random forest also adds randomness to the model. Unlike a decision tress instead of searching for the most important feature while splitting a node. It searches for the best feature among the random subset of features. So, first thing I had to figure out was where was the best place to prune to get the most accurate results, so I went through all the depths again using log loss as accuracy. (n = branch length)



With this random state it looks like the highest accuracy came when the branch length was 7 with accuracy of .926

**Discussion**

Now that I have explain the way I approached the problem. I would like to advert the attention to another method that might be more accurate but requires a lot more memory. BERT or Bidirectional Encoder Representations and Transformer was released by Google AI language, which remember is the team that started this competition. BERT is an embedded. The job of the programmer is fine tuned the model to fit the specific use case of the model.

The problem with the approach is that the just the pre train model is that it will take more than a GB of free space to function. On machines that are lacking in space or space is crucial it could be worth it to go with a model such as mine. BERT includes a fully-connect neural network layer with GEL and normalization and it does a softmax on top of that which can start adding up when it comes to computational time. With K-NN and random forest the longest computation times comes from fitting the model to the training data. This time can also be reduced by decreasing the training size of the model

**Conclusion**

I believe in terms of speed and space that my method blows BERT out of the water at the loss of some accuracy. However, the use of these pre-trained models is still way better than making the model yourself and spending hours, days, or weeks training a model for this competition. I believe NLP is moving in a positive direction but as we find more complicated and diverse solutions the space and computational cost will just keep increasing. This is due to the complexity of semantics in languages.

**Sites used in the making of this project are:**

<https://www.kaggle.com/c/gendered-pronoun-resolution/overview>

<https://www.kaggle.com/shujian/ml-model-example-with-train-test>

<https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>

<https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb>

<https://spacy.io/usage/spacy-101#annotations>

<https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>

<https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>

<https://uncc.instructure.com/courses/89871/files/5641154?module_item_id=1762773>