Medium Article Author NLP Classification

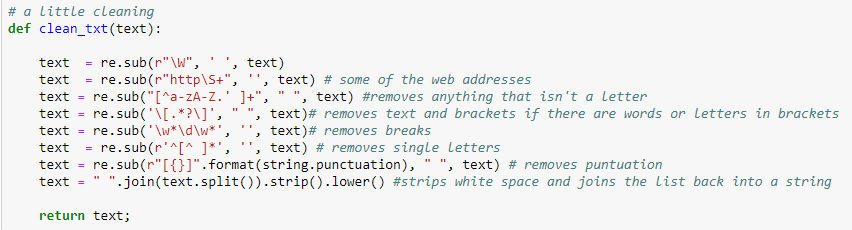
Unit 4 Capstone

In this project we will attempt to classify authors based on their writings of Medium.com articles using supervised and unsupervised methods. The articles will all be based on artificial intelligence, machine learning, or data science. The data is a set of 279577 articles written by 31021

unique authors. There is a mixture of articles written in different languages and on several different topics. The data contains features such as number of claps(Medium.com's rating), the link to the article, the title, and the article itself. For our purpose we will only focus on a portion of the data, the author column and the article column.

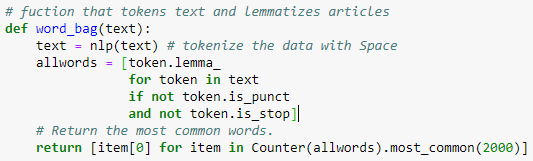
Well start off by selecting only the articles that are all written in the same language. We will use english. The articles also come in a variety of sizes. We want to make sure that the articles are long enough to add value to the corpus. The data will be selected such that the remaining articles with be greater than the median word count. After that we select only the articled with the “artificial intelligence” label. There are several articles that are written by more than one author. It would be difficult to know how well the model is doing if we don’t know who the specific author is so we select rows whose label is only one author. From those authors we select 10 and of those 10 authors we select their first 10 works. This gives us a corpus of 10 authors and 100 articles.

Next we move on to cleaning the data. From here we just write a function with some regular expression patterns that in general eliminates punctuation, words in brackets along with those brackets, escape characters and line breaks, and there were also a ton of reference URLs at the ends of the articles that we want to get rid of. The regex expressions were search for on Stack Exchange and then implemented in the code.

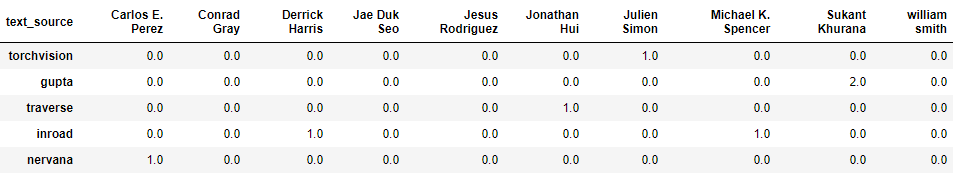


After cleaning the text we used train\_test\_split() to split the data keeping 25% as a holdout set. From here we will begin feature extraction from the text.

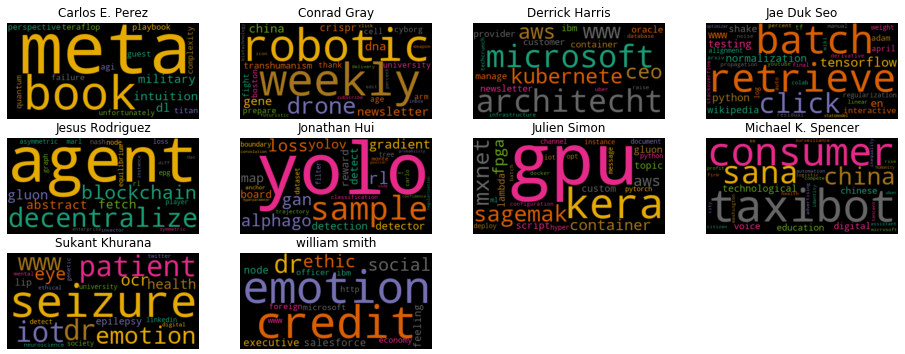
For this project three types of feature extraction method with be used; Bag of Words, TF/IDF vectorizer and Latent Semantic Analysis. To create the Bag of Words features we must first tokenize the data. We’ll use the SpaCy python library for this purpose. Tokenizing the corpus with SpaCy will allow us to extract features from each word such as lemma and will also allow us to exclude stop words and punctuation from by using SpaCy’s built in vocabulary. To do this we define a function and then apply that function to each document in the corpus.



From here we create a Pandas dataframe where each column is a word and each row is a document, a document term matrix.. The intersections are filled with the number of time that that word appears in that particular document. In order to make life easier the matrix was transposed so that the authors names were on top. The unique authors were also combined so that a list of words used by each author could be made.

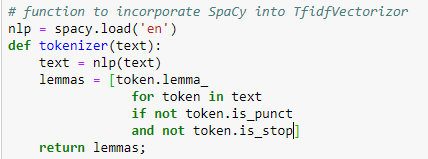


The Counter() function was then used to see how many times the word occurs in the list. A word can only occur in the list once per author so if we use the most\_common() attribute we can see which words are used by how many authors. We can then use this information to add more words to the stop word vocabulary. Also, within that stop word list the names of the authors were added so that no information is leaked when conducting the supervised classification tasks. For this project if a word was used by over 6 authors then that word was added to the Stop word list. Finally we create some word clouds to see if our work is making sense.

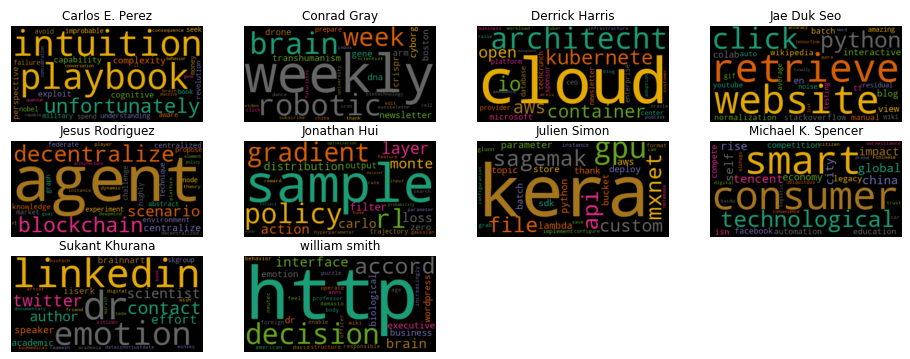


It seems our work is paying off. There is some distinction between the authors and you can somewhat gather common topics each author writes about even though all 10 documents of each author are combined and the overall subject is the same, artificial intelligence.

Next, in a similar fashion we will extract TF/IDF(*term frequency/inverse document frequency*) features from the corpus. First we define a function so that we can incorporate SpaCy into TfidfVectorizer. This will take care of converting the words into lemmas, excluding the stop words and excluding and punctuation that may have been missed during cleaning.

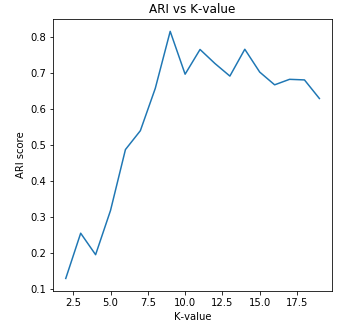


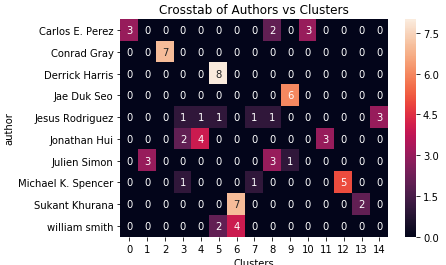
Next we instantiate the TfidfVectorizer and it does the hard work for us. We fit\_transform() the training set and transform() the test set. From here we can make another matrix with the values as the TF/IDF scores instead of counts of each word. In a similar fashion as BOW we can create word clouds with the top words of each author from the matrix

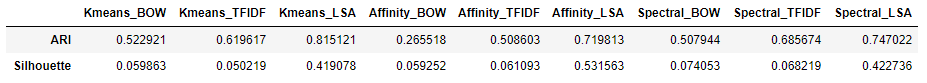


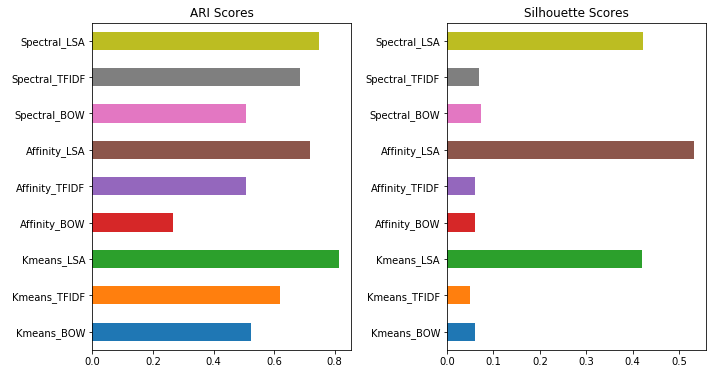
Obviously our there should be some more text cleaning since the ‘http’ should have been eliminated. We won’t worry about that for now.

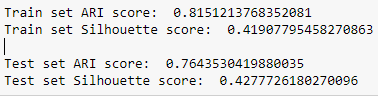
Creating the Latent Semantic Analysis is easy from here since we already calculated the TFIDF matrix. All we do is run Truncated Singular Value Decomposition (SVD) on the TFIDF matrix. This will reduce or feature space and hopefully some of the noise created by our not-to-great cleaning job. For this project we will set the SVD to 10 components then fit and transform the train set and transform the test set and we’re done. Now that we have all of out features we can start making some clusters

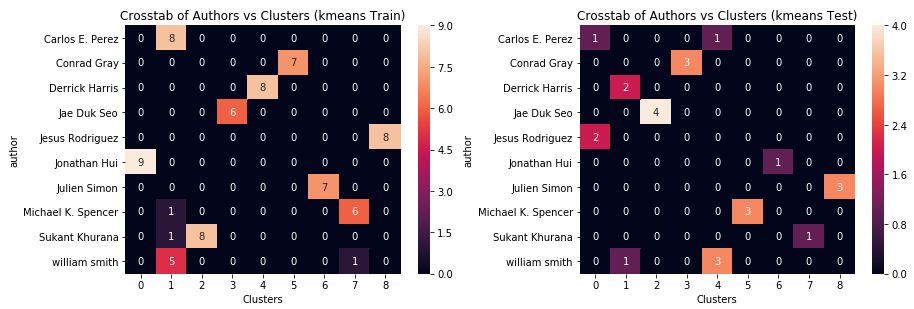
Now that we have our features processed let's build some clusters. The clustering algorithms that will be used are Kmeans, Affinity propagation, and Spectral clustering. I chose these algorithms because we know, generally, how many clusters we are looking for which is 10 for the number of authors we have in our target. These Kmeans and Spectral clustering allow us to specifically choose k number of clusters and with Affinity propagation we can adjust the damping value and run it to see if the number of clusters each value predicts matches the number of clusters we are looking for.

For each clustering algorithm we will run each set of features, BOW, TF/IDF, and LSA. For Kmeans and Spectral clustering we will iterate over a range of clusters that begins a little under our target number of clusters and a little after. For each iteration we will record the ARI score in an array and then plot it. We will finalize the model with the number of clusters that maximizes the ARI score. We will also keep track of the silhouette score as a secondary metric. For each algorithm we will also plot a confusion matrix so we can see which authors are in which clusters. Ideally there would be 10 clusters with each point in the cluster representing a document by the same author but 10 clusters may maximize the classification of authors. After every algorithm is run we will return the scores. 

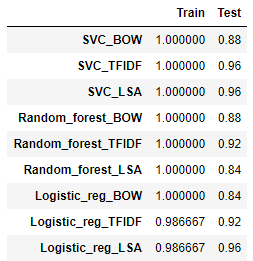


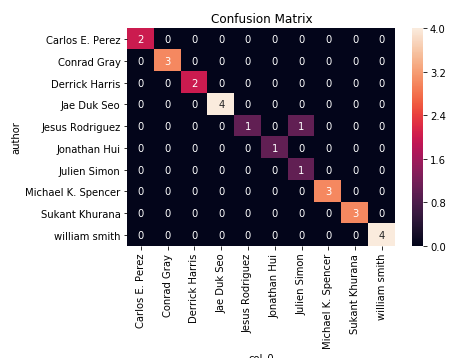


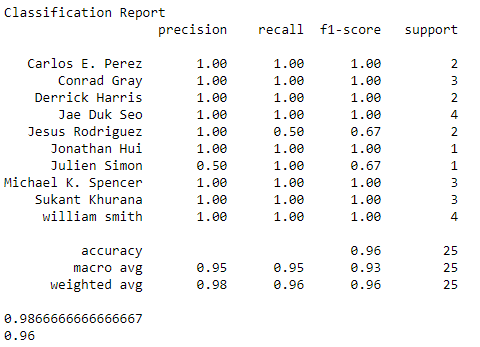
Kmeans with LSA comes back with the best ARI score which is the metric we are using because our goal is classification.This was achieved using k=9 clusters. You can see that all of the algorithms using LSA have much higher silhouette scores than the scores using the other features. Since all of the articles fall under "artificial intelligence" the data is tightly grouped together and it is difficult to cluster. Better cleaning of the data or possibly creating a longer list of domain specific stop words could help raise those scores. However LSA reduces the dimension for us and makes it easier for the algorithms to find better clusters whose members are more closely related and each of those members father from the members in other clusters. From here we will return the test set of the data on our best clustering model and see if it is consistent.

It looks as if or model is performing as we hoped and expected it would. Let’s compare the confusion matrices of the train and test set of Kmeans with LSA.

You can see that some of the same authors in both matrices are in the same cluster. This is most likely because the topic that the author wrote about in the articles that are close together were the same even though the overall topic was artificial intelligence. Let’s see how some supervised models do to classify our authors.

The models that will be used to classify our authors are Linear Support Vector Classifier ,Random Forest model and the Logistic Regression. Before we plug the data into the models we will also do some feature selection on the TF/IDF and the BOW feature sets. To achieve this we will use sklearn's SelectPercentile feature selection tool to select the best features within a certain percentile of the original feature set. Our feature space is very large so we will set the percentile to 10% and let the tool do the rest of the work. Once we have finished we can enter our data into the models and record the results into a dataframe. The best performing combination of is the SVC with LSA. But It looks like it could be overfitting. The training and test scores for the Logistic Regression model with LSA seem to be more stable. 





From the results we have seen here it seems that the supervised classification along with some unsupervised feature extraction it a little more accurate at classifying the texts with the correct authors. If you have labeled data and you want to train a model to be able to accurately classify you the author of a document then supervised classification would be the way to go for this task. Clustering isn’t bad for this task either but it would be better used to see which authors writing styles are the same or if the topics they write about are similar. If you had no labeled data to compare clusters to then you could use clustering for analyzing the topics of the documents and organizing them that way. The supervised learning route doesn't really allow you to analyze the nature of the text itself. Clustering is a little more versatile in looking for patterns but sacrifices accuracy while the supervised models are accurate but perform a very specific task.