# Data 71200 - Final Project

## 1 Choosing the Datasets

My main objective to learn about machine learning is to learn to work with textual data. So when I started to select the datasets for my project, I focus on this area. I did my exploration on Kaggle and I decided to choose the Disaster Tweets datasets. Twitter has become an important communication channel in times of emergency. The emerge of smartphones enables people to announce an emergency they are observing in real-time. Because of this, more agencies are interested in programmatically monitoring Twitter (i.e. disaster relief organizations and news agencies). The main goal of this Kaggle challenge is to predict which Tweets are about real disasters and which ones are not. I have the access to a dataset of 10,000 tweets that were hand classified. It is labeled and It can be both used for the supervised learning experiment and the unsupervised learning experiment. In the process, I also can learn to process textual data and how to vectorize textual content.

## 2 Working on the Data

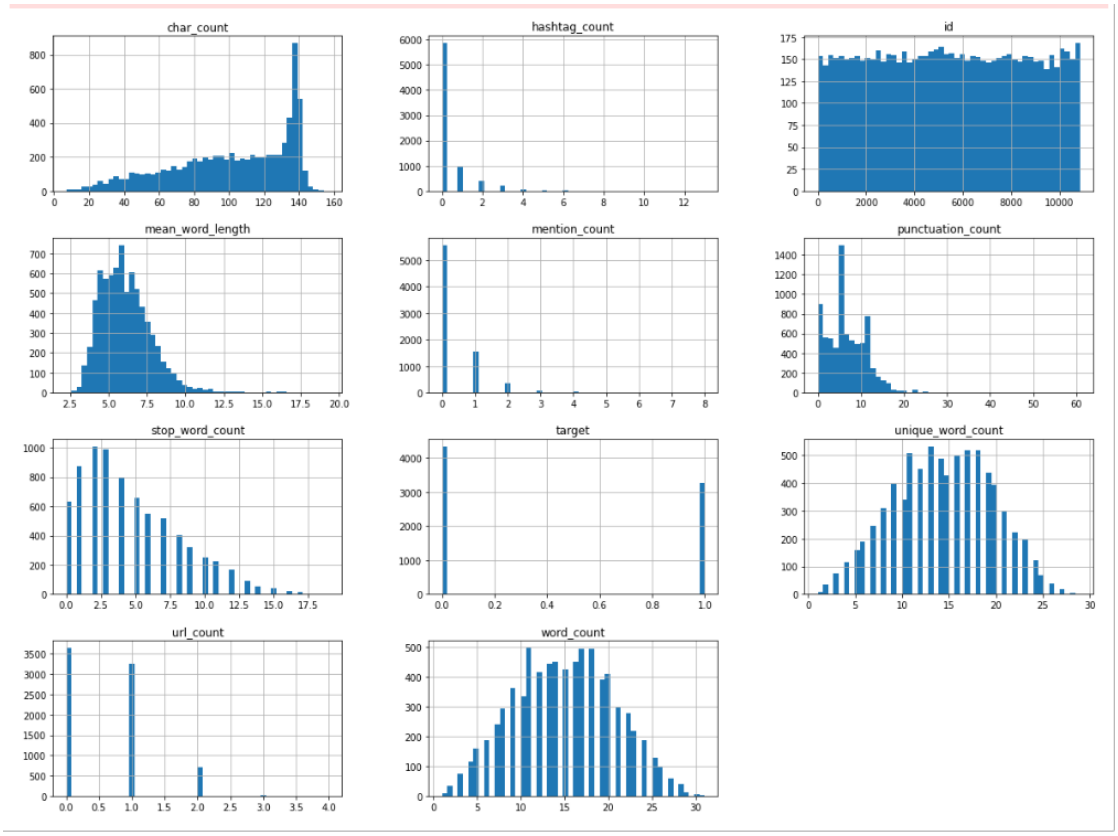
Taking a first look at the data, we have 7613 observation in the training dataset and 3263 observations in testing datasets. We have four features about the twitter content: id, keyword, location, and text. Using the info() function, we can see that there are missing values in keyword and location. We can see the main feature is the text.

To understand the data better, I generated several features on the textual content. One feature I want to talk about is the stop_word_count. One of the major forms of pre-processing is to filter out useless data in the text. The idea of Natural Language Processing is to do some form of analysis, or processing, where the machine can understand, at least to some level, what the text means, says, or implies. In natural language processing, useless words (data), are referred to as stop words. A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that contain very few information. So it is very important to include some features on this issue. Besides stop_word_count, I also create features to count unique words, url, characters of the text, punctuations in the text, hashtag in the text and “@” in the text. I also generated the mean word length feature.

After gathering those basic features of the text, I started to clean the data. I removed the html, urls, emojs and punctuations. Then I get the cleaner version of my raw data. We can see from the below histograms of features: The mode of characters count is about 137. Most of the texts don’t have any mention @. Stop words are around 2.5 and the target (whether it is true disaster twitter text) is about equally distributed. And the unique word count and word count are similar which suggests the texts are kind of short and the reoccurred words are not many.

Then I constructed the scatter plots to see the correlation between features. Most of the patterns are predictable since word count clearly relates with other specific type word count. I also got the

binned chart of some features. We can see that once we processed the data with binned technique, the distribution is clearer for us to see.




I also applied some transformation to the data. I took the log of word_count feature and characters_count feature to make their distribution closer to normal distribution. I also took the square root of hashtag data and mentioncount data since they are very scattered.

I use describe function to get a basic idea on the distribution of each feature.

```
df_train.describe()
```

| | id | word_count | unique_word_count | stop_word_count | url_count | mean_word_length | char_count | punctuation_count | hashtag_count | mer |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7613.000000 | 7 |
| mean | 5441.934848 | 14.903586 | 14.340733 | 4.672928 | 0.620255 | 6.128494 | 101.037436 | 6.839485 | 0.446999 | |
| std | 3137.116090 | 5.732604 | 5.277160 | 3.559228 | 0.664104 | 1.675464 | 33.781325 | 4.608758 | 1.099841 | |
| min | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 2.250000 | 7.000000 | 0.000000 | 0.000000 | |
| 25% | 2734.000000 | 11.000000 | 11.000000 | 2.000000 | 0.000000 | 4.875000 | 78.000000 | 3.000000 | 0.000000 | |
| 50% | 5408.000000 | 15.000000 | 14.000000 | 4.000000 | 1.000000 | 5.928571 | 107.000000 | 6.000000 | 0.000000 | |
| 75% | 8146.000000 | 19.000000 | 18.000000 | 7.000000 | 1.000000 | 7.058824 | 133.000000 | 10.000000 | 0.000000 | |
| max | 10873.000000 | 31.000000 | 29.000000 | 19.000000 | 4.000000 | 19.333333 | 157.000000 | 61.000000 | 13.000000 | |

However, I felt these features containing limited information for the algorithm to learn. The most important feature: text is not well studied. I then started to use text vectorization to get more information from my raw data. I used GloVe for Vectorization. I referred to the notebook on Kaggle to conduct my procedure.

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. There are pre-trained word vectors that can be used with easy access. I used pre-trained word vectors to work on the twitter raw text. To be specific, I use Wikipedia 2014 + Gigaword 5 datasets. It is available in 3 varieties :50D ,100D and 200 Dimentional.I try 100 D in my project. I downloaded the dictionary and built up de embedding dictionary. Then I tokenized both the training and testing datasets.

```
def create_corpus(df):
    corpus=[]
    for tweet in tqdm(df['text']):
        words=[word.lower() for word in word_tokenize(tweet) if((word.isalpha()==1) & (word not in stop))]
        corpus.append(words)
    return corpus
```

```
corpus=create_corpus(df)
```

```
100%|██████████| 10876/10876 [00:01<00:00, 8637.11it/s]
```

```
embedding_dict={}
with open('glove.6B.100d.txt','rb') as f:
    for line in f:
        values=line.split()
        word=values[0]
        vectors=np.asarray(values[1:],'float32')
        embedding_dict[word]=vectors
f.close()
```

## 3 Supervised Learning

To start the supervised learning experiment, I settled my target feature first My target feature is the “target” feature, which denotes whether a tweet is about a real disaster (1) or not (0). My target variable is a binary variable.

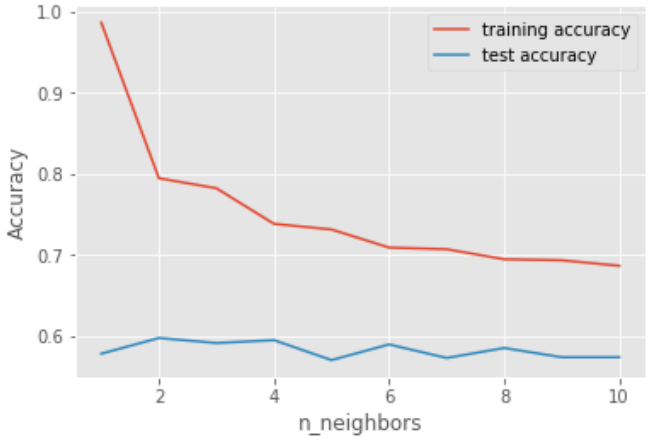
I decided to use two supervised learning algorithms: (1) k-Nearest Neighbors and (2) Random Forest. I chose to use k-Nearest Neighbors is that it did not have any requirement on the distribution of the data, and it is very easy to understand. Also, my dataset is relatively small, and my data is not that sparse. The reason to use Random Forest is that it also does not have any requirement on data distribution. It is also very easy to be interpreted and it works well with mixed data. However, it does have the caveat that it may over fit the data. To deal with this issue, I use cross-validation technique to improve my decision on parameters.

To evaluate the results, I used below four metrics:

1. Accuracy: measures the fraction of the total sample that is correctly identified.
2. Precision: measures that out of all the examples predicted as positive, how many are actually positive.
3. Recall: measures that out of all the actual positives, how many examples were correctly classified as positive by the model.
4. F1 Score: harmonic mean of the Precision and Recall.

### 3.1 k-Nearest Neighbors

I started with default parameters with 5 neighbors. I got following results. I used 10-fold cross validation. The average score is 0.58. The recall rate and precision rate are similar. Then I started to explore best n_neighbors parameter. We can see from the figure that after 6 the accuracy rate remains flat. I used grid search to locate best parameters. It turns out to be: {'metric': 'manhattan', 'n_neighbors': 20, 'weights': 'distance'}. Then I used same procedure as my first trial and got a improved results. The cross-validation score moved up to 0.617 and the accuracy on test set turned to be 0.617. Fitting the testing dataset into the best model I got, I got the feedback from Kaggle that the score for my best KNN model is 0.62269.




```
cv_scores:
[0.60030864 0.60030864 0.58487654 0.5935085  0.58268934 0.59969088
 0.57496136 0.57032457 0.62229102 0.56346749]
cv_scores mean:
0.5892426993333495
Accuracy on test set: 0.617
              precision    recall  f1-score   support

           1       0.63      0.78      0.69       639
           0       0.59      0.42      0.49       503

   micro avg       0.62      0.62      0.62      1142
   macro avg       0.61      0.60      0.59      1142
weighted avg       0.61      0.62      0.60      1142
```

### 3.2 Random Forest

I took similar procedure with random forest method. I used grid search to find best parameters: {'max_depth': 25, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 500}. We can see that after parameters tuning, the performance of the model improved. Submitting the results to Kaggle, the score increased to 0.62746, which is higher than KNN method.

```
cv_scores:
[0.66975309 0.69135802 0.6558642  0.66769706 0.70170015 0.67387944
 0.63833076 0.68315301 0.64551084 0.64086687]
cv_scores mean:
0.6668113450386919
Accuracy on test set: 0.651
              precision    recall  f1-score   support

           1       0.64      0.88      0.74       639
           0       0.70      0.36      0.48       503

   micro avg       0.65      0.65      0.65      1142
   macro avg       0.67      0.62      0.61      1142
weighted avg       0.67      0.65      0.62      1142
```

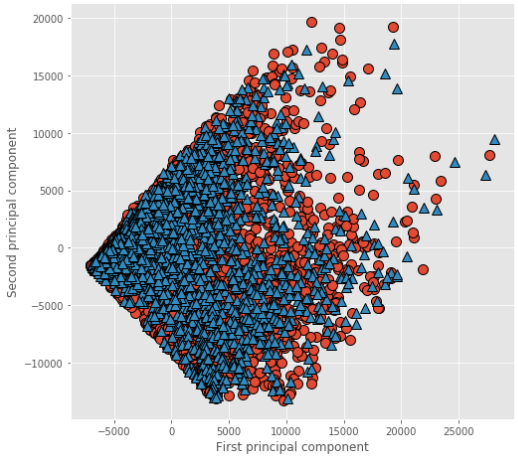
### 3.3 Summary

Even though random forest model improved the results, the overall performance of two methods are not that well. It is caused by the underlying datasets. Unlike some datasets we used during class, such as the housing price datasets, the text data is more complex and just using vectorization method is not enough. Learning from Kaggle, I found most people use sequential model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. Sequence models, in supervised learning, can be used to address a variety of applications including financial time series prediction, speech recognition, music generation, sentiment classification, machine translation and video activity recognition. The only constraint is that either the input or the output is a sequence. In other words, you may use sequence models to address any type of supervised learning problem which contains a time series in either the input or output layers. I think it would be my future learning goal.

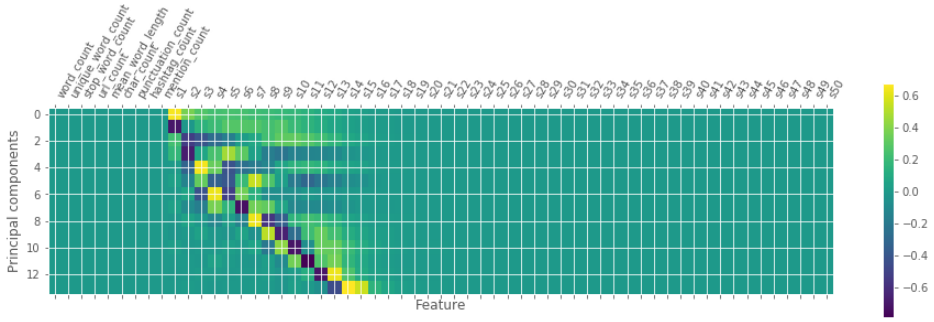
## 4 Unsupervised Learning

### 4.1 PCA Procedure

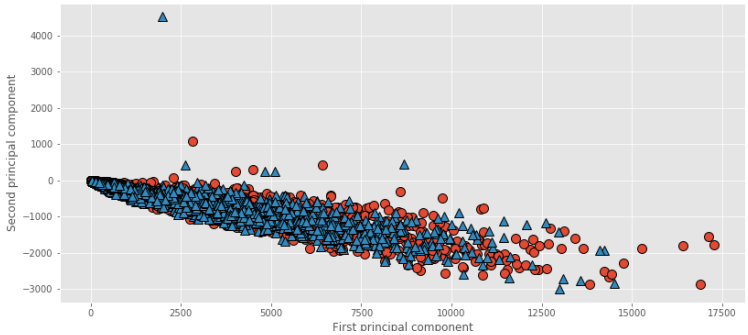
After combining my text vectors and project 1 generated features, I start to use PCA method on my data. I tried to capture 95% variance. Fitting the model on unscaled data, we can see we have 14 features included under the model. Judging from the plot, the clusters are still stacked together. For very large components, we can see slightly division.



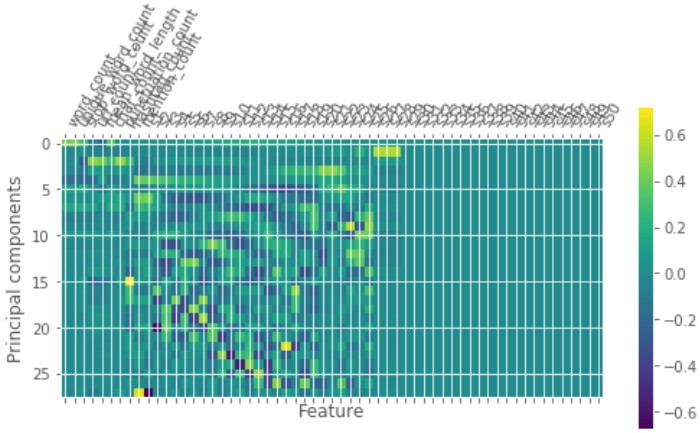
We can see from the gca plot that most generated features are not included in components.




Then I conducted the same procedure on scaled data. We can see that the model includes more features on scaled data. We have 28 featues for 95% variance explianed. And we can see data are more squeezed into the middle. And for scaled data, generated features are used.

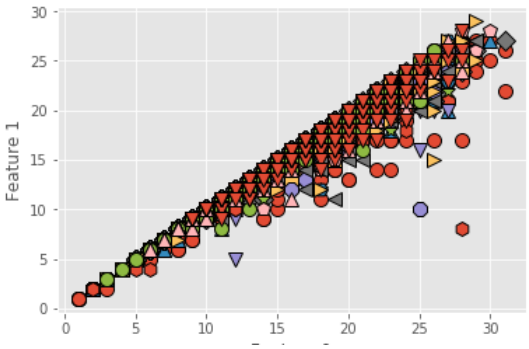



Then I used the random forest model I developed in project 2 and used it on unscaled data, unscaled pca-processed data, scaled data, and scaled pca-processed data. For no pca processed data, we can see the scale improves the performance. The accuracy on test set improves from 0.682 to 0.701. However, the PCA doesn't improve the performance of prediction. For unscaled data it has 0.628 scores under PCA and for scaled data it has 0.695 scores under PCA. Both are lower than no pca processed results. My guess is that the information is very sparse and keepping 95% variance actually lose information, resulting in lower accuracy score.

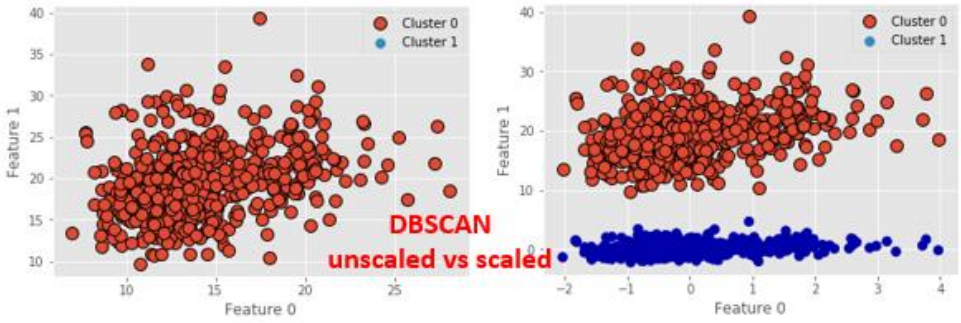


**4.2 Clustering with Kmeans, AgglomerativeClustering and DBSCAN**

I tried on my own data but its visualization indicates that my data is not suitable for this task so I switched to use the breast_cancer data from scikit-learn。 I think the main reason is that my dataset's features are too sparse so it is very unlikely to summarize most of the information by 95% variance.




After getting the X and y, I used Elbow plot to find optimal number of clusters for both scaled data and unscaled data. It turns out to be 4 for unscaled and 3 for scaled data. We can see the scaled data has way better performance when I used dbscan model.



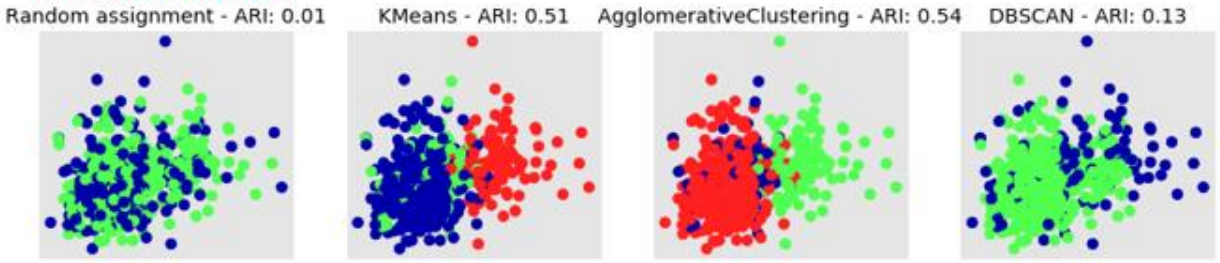

I compared three clustering algorithms with and without ground truth. I got the ARI and Silllhouette score for both scaled data and scaled data processed with PCA. We can see that PCA improved the results and among the three methods, agglomerative clustering seems to work best.

## 5 Final Thoughts

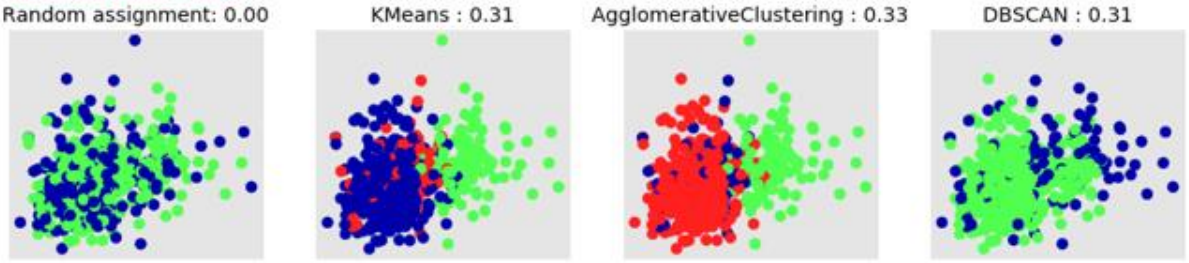
I think the most important gain from the course is that I became familiar with the whole process of machine learning project, especially surprised learning. I also learned how to vectorize textual data and how to deal with raw data in a general way. In addition, I learned many data visualization techniques. All of them are very helpful for my future research. I think I should include more exploration and visualization on the text data to give audience a better understanding. Overall, I really appreciate this course and the lecture notes we get surely will serve me well for my future python machine learning path.

**Scaled Data without PCA**

**1. adjusted_rand_score**

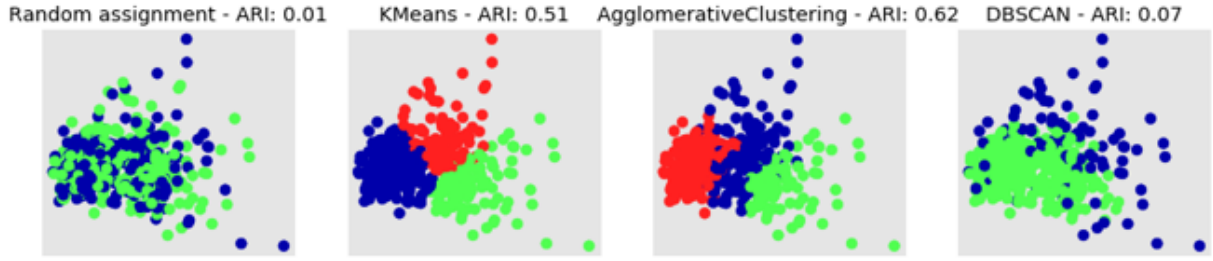



**2. silhouette_score**




**Scaled Data with PCA**

**1. adjusted_rand_score**




**2. silhouette_score**