Project 1 Milestone Report

**Introduction**

Web content categorization is important for online industries in many perspectives and has a wide range of applications. Firstly, it can help with ‘related content’ recommendations. For example, publishers like NEW YORK TIMES will want to recommend articles of the same topic as compared with the topic of an article a reader just read and shows interests in. For another example, game publishers like PlayStation can recommend ‘similar games’ based on the content of the game descriptions. For another example, e-commerce shopping websites like amazon can recommend related products according to the similarities between product descriptions.

Web content categorization also can help customers to avoid risky contents. For example, in online advertising industry, advertisers especially big and good brands like NIKE, Cocacolar, Nestle would not want their advertisement shown on webpages containing inappropriate contents like hatred speech, violation or sexual. To the opposite, they want customers see their brands on healthy, optimism webpages. Social media websites need web content categorization to detect and remove harmful user generated contents, such as contents to teach people how to commit a crime, child abusement. For example, since COVID-19 happens, social media websites such as facebook and youtube try to remove misleading contents like drinking bleach to kill the virus in our body.

In the realm of online media publishing, categorizing web contents is helpful for publishers to organize their own contents. As a result, they can group all articles under right topics easily, without reading and identifying articles one by one.

To understand the potential consumers, we also need a good web content categorization. Consumers interact with contents by viewing, reading, clicking, purchasing and so on. The topics extracted from the contents people interacted with can indicate their interests. By knowing what type of topics a consumer visit more frequently, we can infer the consumer’s own inherent interests. This can be used widely. For example, online advertising targeting where advertisers target a consumer’s topical interests; for another example, doing personalization in the ecommerce applications, where ecommerce platforms want to understand their consumers’ purchase interests and match products with consumers’ interests.

However, to categorize webpages according to their contents is not an easy job. Usually there are too many webpages for human labor to label and suitable human editors are hard to find. Meanwhile, user generated contents are created every single minute, thinking about President Trump tweeted 200 times on Jun 5, 2020. It is impossible for human curators to do real time monitor and track every piece of contents. Efficiency is another thing we need to consider. Costumers might need fast, automatic solutions to detect topics from web contents. Like social medias have to provide real time topic trends.

Machine learning techniques are undoubtedly the best way to that people will resort for help.

In this project, I will build a machine learning model to categorize webpages based on their contexts.

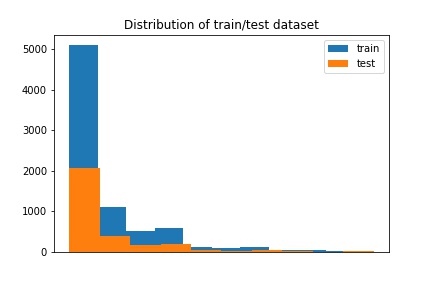
**Dataset and Pre-processing**

The data I use is a built-in dataset of a collection of Reuters articles from NLTK library, which is a famous library for Natural Language Processing.

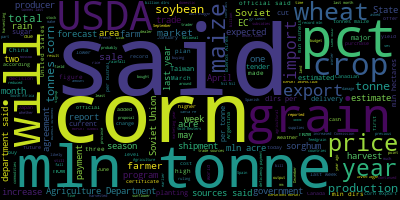
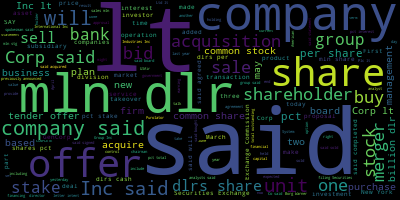
The documents in this collection appeared on the Reuters newswire in 1987.

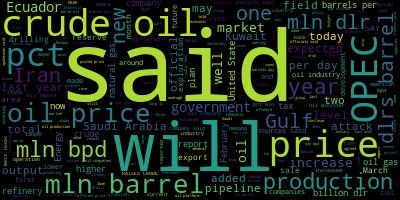
The documents were assembled and indexed with categories by personnel from Reuters Ltd. It contains 10788 news articles from Reuters labeled with 90 categories according to their topic, such as Politics, Economics, Sports, and Business.

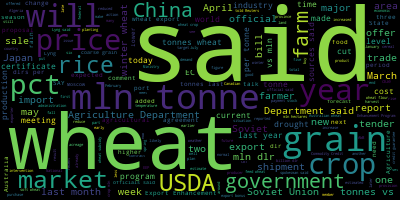
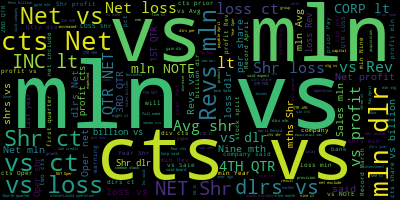
The data has been pre-seperated into training data and testing data by a ratio of 7:3 with similar distribution.

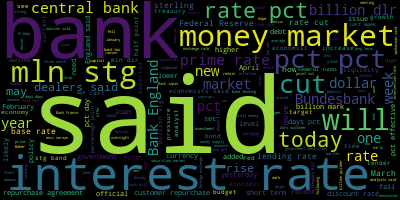
Since the dataset shows a highly skewed distribution, and there are only a few articles in the topics other than the top10 most frequent topics, I kept only top10 topics and put all articles in other topics into a new category as ‘other’.

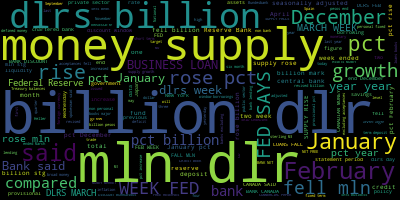
The following pictures show the most frequent words in each category.











As we can see, words like ‘said’ takes a large area in the word cloud analysis pictures. That is caused by the stop word problem, we will talk about this in the following session.

Remove stop word

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that takes up a lot of valuable space and processing time but barely carries much information. Here, I used NLTK package in python to remove the stopwords in all the articles.

Word stemming

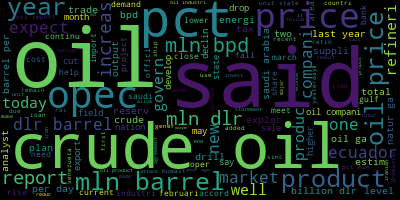
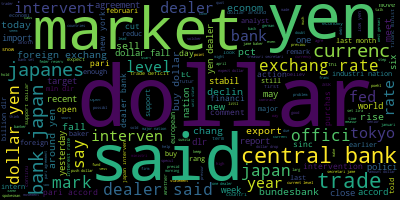
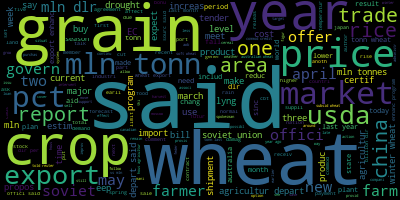
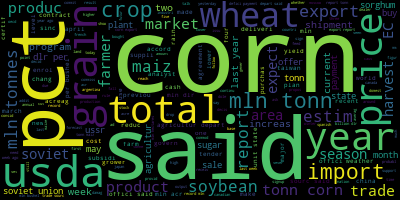
Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. For example, words in different time tense will be changed back to original form.

Word tokenization

Word tokenization is the process of splitting a large sample of text into words. Word tokenization can help us to split words in a right way and keep as much information as possible.

Data exploration

After all these steps, I used word cloud to show the most frequent words in each topic.

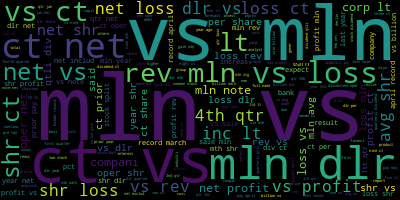


Corn

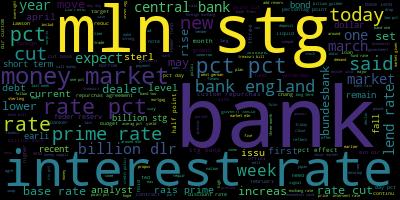
Grain

Dollar

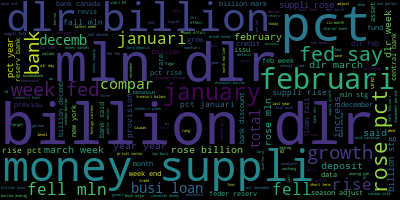
Crude



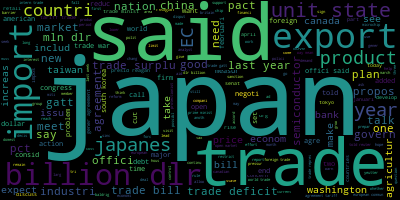
Earn



Interest



Money-supply



Trade

As we can see from the figure, after the pre-procession of the data, the most frequent words in each category are different and closely around the topics.

Here we can also do……

**Feature Engineering/Word Embedding**

For natural language processing, one of the most important challenges is how to transfer sentences and words into features and keep as much information as possible.

Here, I tried several different ways to transfer the articles into features.

**Count Vector**

Count vector is representing in an article by counting the number of times each word shows in the it.

**TF-IDF Vector**

short for term frequency–inverse document frequency

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

The tf–idf value increases [proportionally](https://en.wikipedia.org/wiki/Proportionality_(mathematics)) to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

TF-IDF can be done on different levels, and here are things that I have tried:

* + Word level
  + n-gram level: n-gram is n consecutive words in a sentence
  + Character level

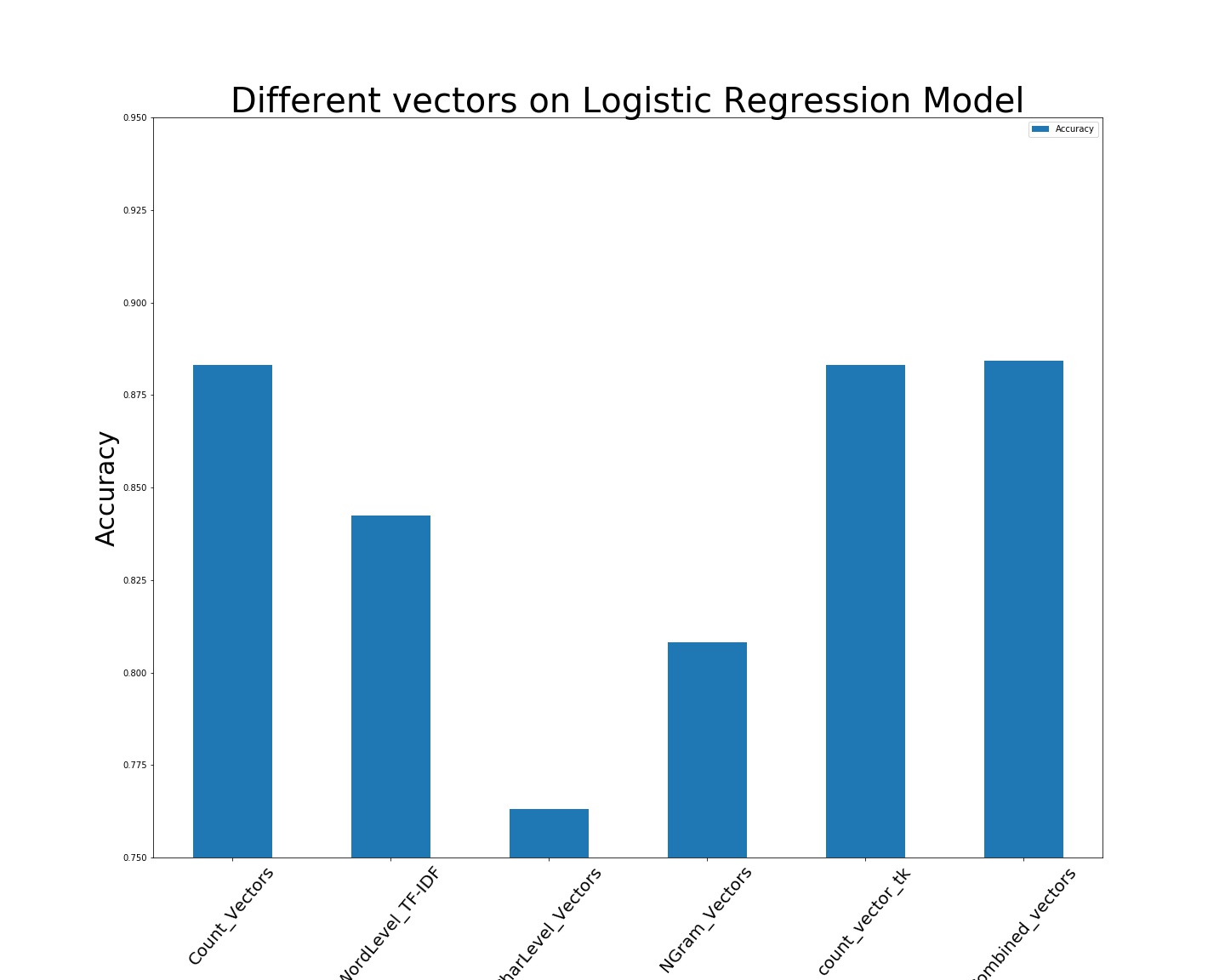
**Word Embedding**

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

**Text based features**

Text based features are features about the texts, like the number of character， the average word density, the number of punctuation, upper case, and the frequency distribution of part of speech tags, like noun count, verb count, adjective count, adverb count, pronoun count.

**Combine different vectors to make better of the model performance**

****

The figure shows the performance of different vectors on the baseline model. As we can see, different vectors may have a big difference on representing the topic of the articles.

**Model Selection**

For to find the best model, I try models from following perspectives: Linear model, tree model, svm, resembled model.

Logistic regression:

According to Wikipedia, logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. Using logistic regression to build a multi-classes model is one of its extensions.

Random Forest:

Is an ensemble learning method based on decision tree, but usually it has a better performance than decision tree.

Naïve Bayes:

In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

XgBoost:

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.----XGBoost Documentation

Support vector machine(SVM)

SVM is a machine learning algorithm that used for classification.

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

The figure shows the performance of different algorithms with features processed in different ways. So that we can see at the same time which feature and algorithm work the best. The figure shows that with the same feature, different algorithms give different accuracy.

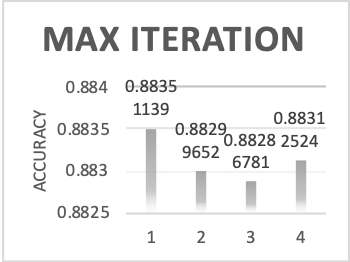
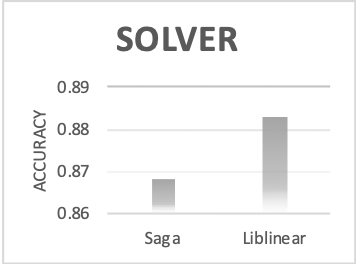
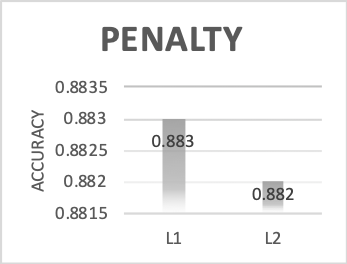
Different models have a big different runtime to train the same model. As we can see, Xgboost takes much much more time than other algorithms, although it’s accuracy is one of the best.

The runtime of algorithms is important for us to consider when we choose the model, because sometimes we hope we can build up a model quickly to solve our problems, say, realtime problems. Or when it is a big size dataset to train and predict, it might take even longer time. If the model takes too much time to run, even if it has a high accuracy, it might not be the right choice.

At last I choose logistic regression as the final model, because it gives a good accuracy and at the same time, it runs fast.

Model parameter tuning

Penalty, solver, max iteration are the most important parameters for logistic regression. To find out the best values of these parameters for my model, I try various values for the model on the dataset. And here are the choices, penalty = l1, solver = ‘liblinear’, Max-iter=10.



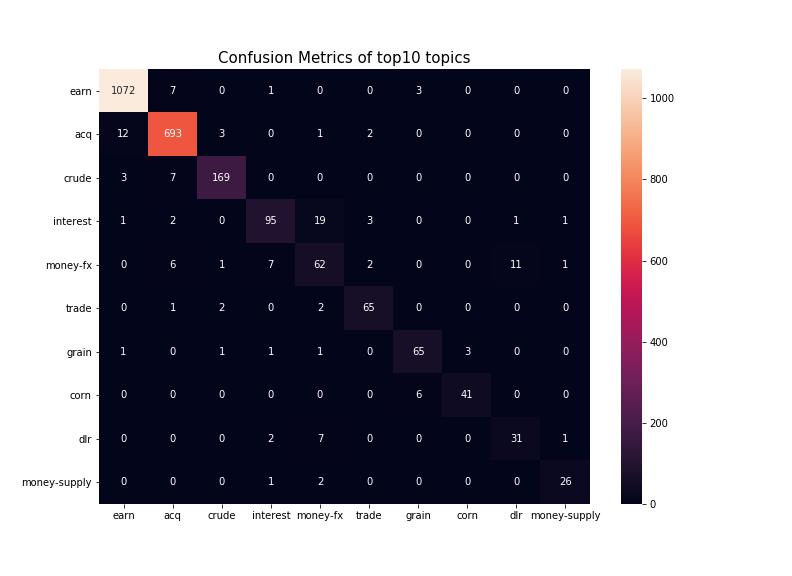
Model Evaluation

As we can see from the confusion matrix, the model did a good job on predicting the test data on all of the topics.

After testing the model on testing data, I test the model on data out of the given dataset. I searched and downloaded several articles from Reuters website using the main topics of the categories as keys words, such as ‘trade’, ‘corn’. And then I process the article the same way as I process the training and testing data, so that transform the articles to features good for the model.

The result turns out good. The articles titled ‘ U.S. top trade official lauds USMCA as model for future trade deals’ , ‘EU seeks reset in trade talks with U.S. trade chief Hogan’ are both put into the category of trade.

When I use articles from other news websites such as BBC, CNN, the prediction of the model is not very satisfying. This can be understood that I used Reuters articles to train the model and The Reuters have their own specific way to report a news.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **Earn** | **0.99** | **0.98** | **0.99** | **1094** |
| **Acq** | **0.96** | **0.96** | **0.96** | **721** |
| **Crude** | **0.9** | **0.85** | **0.87** | **191** |
| **Interest** | **0.77** | **0.86** | **0.81** | **111** |
| **Money-fx** | **0.61** | **0.59** | **0.6** | **100** |
| **Trade** | **0.78** | **0.79** | **0.78** | **75** |
| **Grain** | **0.81** | **0.85** | **0.83** | **73** |
| **Corn** | **0.81** | **0.91** | **0.86** | **43** |
| **Dollar** | **0.67** | **0.69** | **0.68** | **42** |
| **Money-supply** | **0.9** | **0.81** | **0.85** | **32** |
| **Others** | **0.87** | **0.88** | **0.87** | **537** |



