Capstone Final Report

1. Abstract

Natural Language Processing (NLP) is a very hot topic in the Artificial Intelligence, and NLP includes plenty of the areas, and I focus on the part of the news classifications in my Capstone Project, and there are four labels in my dataset, politics, entertainment, us and world. The dates of these news are between July 2019 to March 2020. I implement this classification problem in five algorithms, Naïve Bayes, Logistics Regression, XGBoost, Forward Neural Network, and Recurrent Neural Network (RNN).

It is very important to handle text or word features in machine learning and deep learning models, and people always implement the Word Embedding to make the words to vectors. It also called the words vectorization, and I implement words vectorization by Word2vec method and term frequency–inverse document frequency (TF-IDF) method.

2. Data Collection

I collected the news data from the website, and I used web crawler technology to get the data which comes from Fox News website. The website crawler sometime called the spider or spiderbot, and crawler can valid the hyperlink and HTML code, there are some different technology to load the contents from the website, for instance, Regex, Beautiful Soup, and XPath, and I used the XPath technology to load the contents of the news from the website.

XPath is the built under the lxml[1] module in python libraries, and lxml module is very strong, and easy to learn, and it is faster than Beautiful Soup, but a little bit slower than Regex. In the Fox News website, people should click the load more button to load more news, and Selenium can simulate this behavior. But Selenium can't simulate this behavior by itself, and it relies on the specific version of browser driver, and I used google chrome with 81 version to crawl data so I need download the corresponding version of the ChromeDrive, and official offer corresponding version in ChromeDriver-WebDriver for chrome website[2]. In the Selenium, there is a function named click, and this function can simulate the click button behavior, so I use the

[1] https://lxml.de/index.html

[2] https://sites.google.com/a/chromium.org/chromedriver/downloads

XPath to find the load more button in the HTML code, and then I use click function implement the click behaviors. But there was an issue that the Selenium was very slow while progressing. By my calculations, it got six news from the Fox News website per minute, and I got 9620 news in total, and it took me a lot of time the crawl the data from Fox News.

Then I choose the json format to store the data, and the same data stored in json format is much smaller than the excel format, and excel always got the plenty of gibberish characters when I stored the data in the excel although I set the code to UTF-8. There are four features in my dataset, title, content, date and category, and they are all stored in string format. I created four txt files to store each category of the data in json format.

3. Data Cleaning and Preprocessing

I read these four Json format files into the four data frames format by the function read_json in pandas. There are lots of noises with plenty of different kinds of the forms after I crawled the data from website, for instance, there are some noises in date feature, and the date is obviously out of the range, and in the category of us, there is a kind of the news only have one sentence: "Also on this day:". There are also some null values in these four features. At first, I deleted the null values directly, and there were only a few null values in the dataset so the deletion of the null values didn't affect the whole dataset. There were still some anomalies in the dataset after I deleted the null values, especially, most of the anomalies were in the date feature. Secondly, I choose the regex to handle these anomalies, and the regex is both offered by pandas and re modules in python. The regex was good to detect the anomalies, and I replaced these anomalies by null values. Then the null values of the date are replaced by the previous data because the date is a continue feature.

It was hard to do some operations on the date feature if they were in string format, for instance, it is impossible to directly compute the range between two date so I need change the date feature from string format to the date format, and python offers a lot of modules and functions about time series topic, and the parse function in the dateutil module is a good chance. Parse function can directly change the string format of the date into the date format properly. After these operations, the dataset was clean with no anomalies. At last, I concatenated these four data frames into one data frame with resampling method in order to rearranged the data.

4. Text Feature Extraction

Text analysis is a major application for machine learning. However, we can not use the raw text data into the algorithms. First, I need to introduce some important concepts:

- tokenizing: strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators[1].
- counting: the occurrences of tokens in each document.
- text normalization: means converting it to a more convenient, standard form. For example, most of what we are going to do with language relies on first separating out or tokenizing words from running text.
- lemmatization: is another part of text normalization, the task of determining that two words have the same root, despite their surface differences[2].
- stop words: are words like "and", "the", "him", which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality[3].

At first, I set every words in the lowercases, and then I removed the stop words because I think stop words were no longer of any use in text classification, and it would affect the tag clouds. After removing the stops words, I printed out the tag cloud for each category.

[1] https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[2] Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition; Daniel Jurafsky, James H. Martin

[3] https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

The orders of these four imagines are us, entertainment, politics and world. The result of these tag clouds look good, for instance the, in tag cloud of the politics news, the date of these politics news were between January 2020 to March 2020. In this period, the hottest news in politics was the Impeachment of President Donald Trump. We can see the words like impeachment, trump in the tag cloud.

In order to implement algorithms in machine learning I needed to vectorize the text features. I should make the corpus into a very high dimensional sparsity matrix, and the corpus stands for the text data which I can train in the machine learning algorithms. I need to introduce some concepts at first, then I will explain these later:

- Corpus: The dataset which stores our documents
- Bags of words: Vectors of word counts or frequencies
- Bags of n-grams: Counts of word pairs (bigrams), triplets (trigrams), and so on
- TF-IDF vectors: Word scores that better represent their importance[1]

The Scikit Learn Library offers the TF-IDF method, and I used the TF-IDF vectorization function named TfidfVectorizer to transform text feature into a high dimensional sparsity matrix. Next I will introduce how TF-IDF method work for the text feature which cited from Scikit Learn website.

TF means term-frequency while TF-IDF means term- frequency times inverse document-frequency: td-idf(t,d) = tf(t,d) × idf(t).

$tf(t,d)$ means the term frequency, the number of times a term occurs in the given document, is multiplied with idf component. Thich is compute as:

$$\text{idf(t)} = \log\left(\frac{1+n}{1+df(t)}\right) + 1$$

where n is the total number of words in the document set, and $df(t)$ is the number of words in the document set that contain term t. The resulting TF-IDF vectors are then normalized by the Eclidean Norm:

$$V_{\text{norm}} = \frac{v}{||v||^2} = \frac{v}{\sqrt{v1^2+v2^2+\cdots+vn^2}}$$

[1] Natural Language Processing in Action Understanding, analyzing, and generating text with Python; Hobson Lane, Cole Howard, Hannes Max Hapke, Arwen Griffioen

This was originally a term weighting scheme developed for information retrieval that has also good in text classification[1].

N-gram is also an import concept in the text feature, and it means compute the probability of the next N words given a sentence, for instance, suppose given a sentence "its water is so transparent that" and we want to know the probability that the next word is "the":

P(the | its water is so transparent that)

In previous example, this is the one-gram instance that we just compute the probability of one word given by a sentence. We also compute the two-gram, three-gram, etc. In my project, I didn't set the specific N-gram and I used one- gram, and one-gram is also the default number in the Scikit Learn Library. With a large enough corpus, it is easy to compute the probability. I have 9620 items in my dataset, and it is enough to count words in corpus and compute the probability[2].

So far, I finished text feature extraction part. I remove stop words, and drew the tag clouds, and I mapped my corpus in a high dimensional matrix, and it is the time to build the machine learning models to predict the classifications of the news.

5. Naïve Bayes

Classification is the heart both in human and machine intelligence, and classification is a supervised problem in machine learning. Naïve Bayes algorithm is a classic and popular supervised classification algorithm because it is very fast and with high accuracy, and it performs very well in text classification.

As its name, Naïve Bayes is built on Bayes Theorem, and Bayes Theorem computes the posterior probability given a prior probability. Here is the formula of the Bayes Theorem:

$$P(A|B) = \frac{P(A|B) \times P(B)}{P(A)} = \frac{P(A|B) \times P(B)}{P(A|B) \times P(B) + P(A|B^c) \times P(B^c)};$$

Where A is posterior probability and B is prior probability, and $B^c = 1 - B$.

[1] https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[2] Natural Language Processing in Action Understanding, analyzing, and generating text with Python; Hobson Lane, Cole Howard, Hannes Max Hapke, Arwen Griffioen

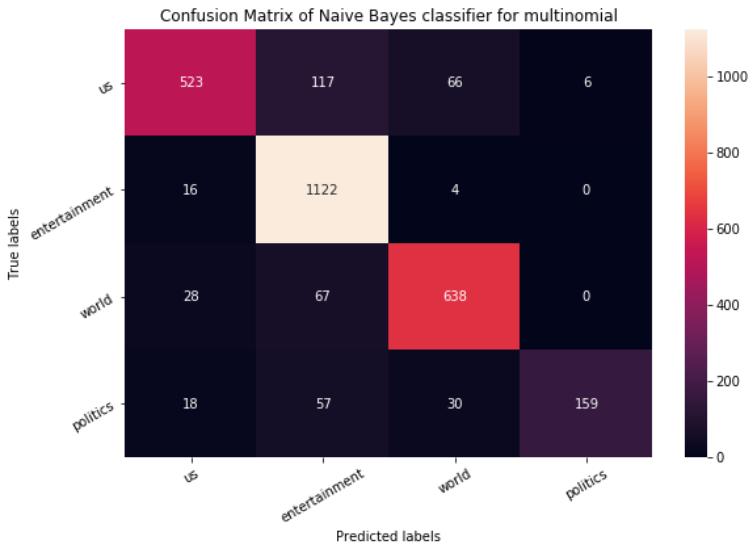
Naïve Bayes assumption is the conditional independence assumption that the probability $P(f_i|c)$ are independent given the class c and hence can be "naively" multiplied as follows:

$$P(f_1, f_2, \ldots, f_n \mid c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots \cdot P(f_n|c)$$

Where $f_1, f_2, \ldots, f_n$ are the set of the features of the document, and c is the class of the document. The final equation for the class c chosen by naïve Bayes classifier is thus:

$$c = argmax_{c \epsilon C} P(c) \prod_{f \in F} P(f|c)$$ [1]

This is how Naïve Bayes works mentioned above, and I implemented Naïve Bayes in my project to predict the classes, and I got 85.7 percent accuracy, and it took 0.13 second to train the model. The result was pretty good with high accuracy and high speed. Here is the confusion matrix of naïve Bayes.



From this confusion matrix, we can see naïve Bayes perform very well in news classification problems, especially, it got very high accuracy in the politics. But I didn't know whether this model is stable, that means can this model get good and stable result on another test sets, so I used k-fold cross validation method to test this model. I will introduce how k-fold cross validation works in machine learning.

Cross validation is a popular and useful method to test how the this model work in different test sets, for instance, in our dataset, we could divide our dataset into ten sub dataset, and then we used each of the ten sub dataset to be the test set, and others nine sub dataset

[1] Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition; Daniel Jurafsky, James H. Martin

combine to one dataset , and it is training set. We can get ten results and we can see how the model works on these ten different test sets. Scikit Learn offers the k-fold cross validation method to the researchers. Here is the result of the cross validation in ten-fold cross validation and I also compute the mean accuracy of ten results.

```
Score of each Validation is: [0.86645636 0.84315789 0.85263158 0.84526316 0.86842105 0.86842105
 0.85368421 0.86421053 0.86947368 0.86631579]
Mean of score is: 0.8598035309092922
```

From above results, we can see Naïve Bayes algorithm is very stable in news classification. In summary, naïve Bayes works very fast, with high accuracy and stable in news classification problem.

5. Logistic Regression

Next, there is a very important and interesting algorithm in text classification problem named logistic Regression. It performs very well in binary classification, and it also works well in the multiple classifications problem. Logistic Regression is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class and otherwise it predicts that it does not. This makes it a binary classifier. In multiple classifications problems, estimator compute the probabilities of each class, and it makes the decision of the class with the biggest probability.

Let use the binary classification problem to be a example to introduce how Logistic Regression works. Logistic Regression works like a Linear Regression model, a Logistic Regression model computes a weighted sum of the input feature, but instead of outputting the result directly like the Linear Regression model does, it outputs the logistic of this result[1]. The following equation shows how Logistic Regression estimate probability:

$$\hat{p} = h_\theta(x) = \sigma(X^T\theta)$$

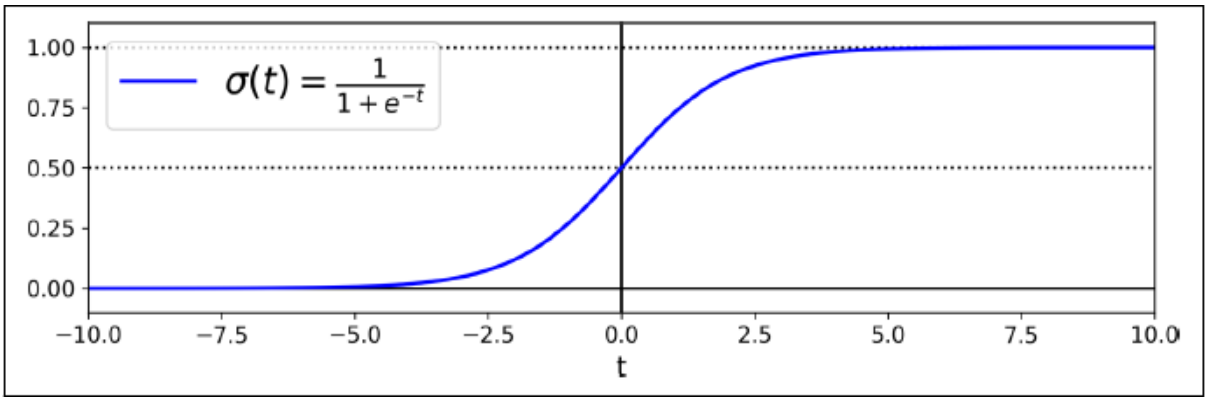
Where X is the matrix stored input data, and θ is the vector stored the weights of each of the feature.

[1] Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron

Let define a sigmoid function $\sigma(t)$, and sigmoid function can make the output of any functions between 0 to 1. Here is the formula of the sigmoid function:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

and the domain of sigmoid function:



In the binary classification problem, notice that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$. Logistics Regression train to set the parameter vector θ so that the model estimates high probabilities for positive instance ($y = 1$) and low probabilities for negative instances ($y = 0$). This idea makes the cost function shown in the following equation for a single training instance x:

$$c(\theta) = \begin{cases} -\log(\hat{p}) & if\ y = 1 \\ -\log(1 - \hat{p}) & if\ y = 0 \end{cases}$$

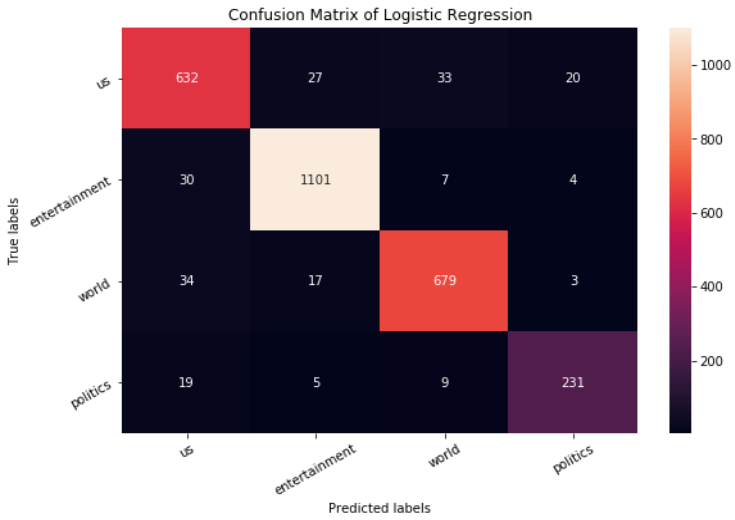
We can see t will approach 0 if $-\log(t)$ grows very large, so the cost will be large if the model estimates a probability close to 0 for positive instance, and it will also be very large if the model estimates a probability close to 1 for a negative instance.[1]

Then we optimize to minimum the loss function $J(\theta)$:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{p}^{(i)})]$$

This is a brief introduction to show how Logistic Regression model works, and I implement Logistic Regression model which is offered by Scikit Learn library. The accuracy of Logistic Regression is much better than the Naïve Bayes, and it got 92.7 percent score in the test

[1] Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron



set, and it improve 7 percent points than Naïve Bayes model. But Logistic Regression is much slower than Naïve Bayes, and it took 10.9 seconds to train the model.

From the confusion matrix, Logistic Regression performed very well in each class. We can see it performed much better than Naïve Bayes in each class. Logistic Regression is more suitable than Naïve Bayes in news classification even though it much slower than Naïve Bayes.

Here is the result of the 10-fold cross validation, and it is very stable:

```
Score of each Validation is: [0.93270242 0.91578947 0.93368421 0.92421053 0.91894737 0.93684211
 0.92631579 0.92210526 0.92947368 0.93157895]
Mean of score is: 0.9271649786927888
```

We can see Logistic Regression works pretty good in news classification because it is stable and high accuracy, except the speed, everything is much better than Naïve Bayes.

6. XGBoost

XGBoost is a hot and popular algorithm in these years, and it implements machine learning algorithm under 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[1].

Boosting refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor[2]. There are many boosting methods, and Gradient Boosting is one of the hottest boosting method, and XGBoost evolves from Gradient Boosting. I implemented XGBoost in xgboost library which is very similar to Scikit Learn.

[1] https://xgboost.readthedocs.io/

[2] Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron

The core ideas of XGBoost are:

1. Keep adding the trees, and keep split the features to finish a tree, and keep add a tree that means create a new function $f(x)$ to fit the residual.
2. When we get k trees in total, we need to predict a score of a single sample, according to the features of this sample, every sample would go to the bottom node of the tree, and it would get a sore by this node.
3. At last, we need to add every score of the tree together, and this total score is the prediction by XGBoost.

Obviously, our goal is closer and closer from prediction value y_i' to the real value y_i,

And it starts from constant prediction, and add a new function each time.

$$\hat{y}_i^{(0)} = 0$$

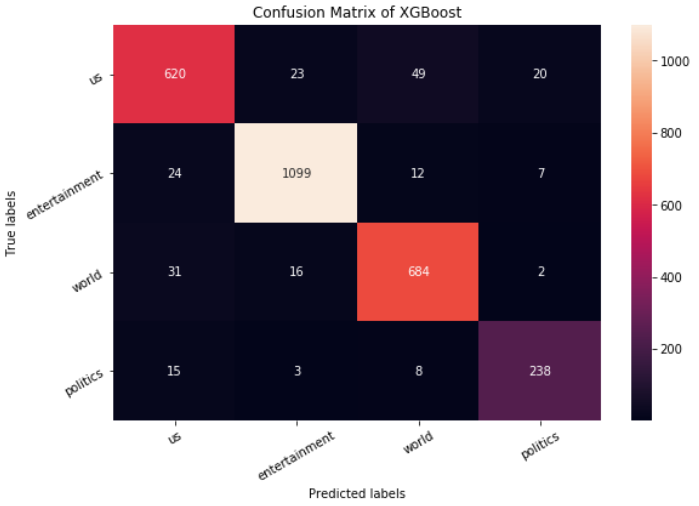
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_1(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

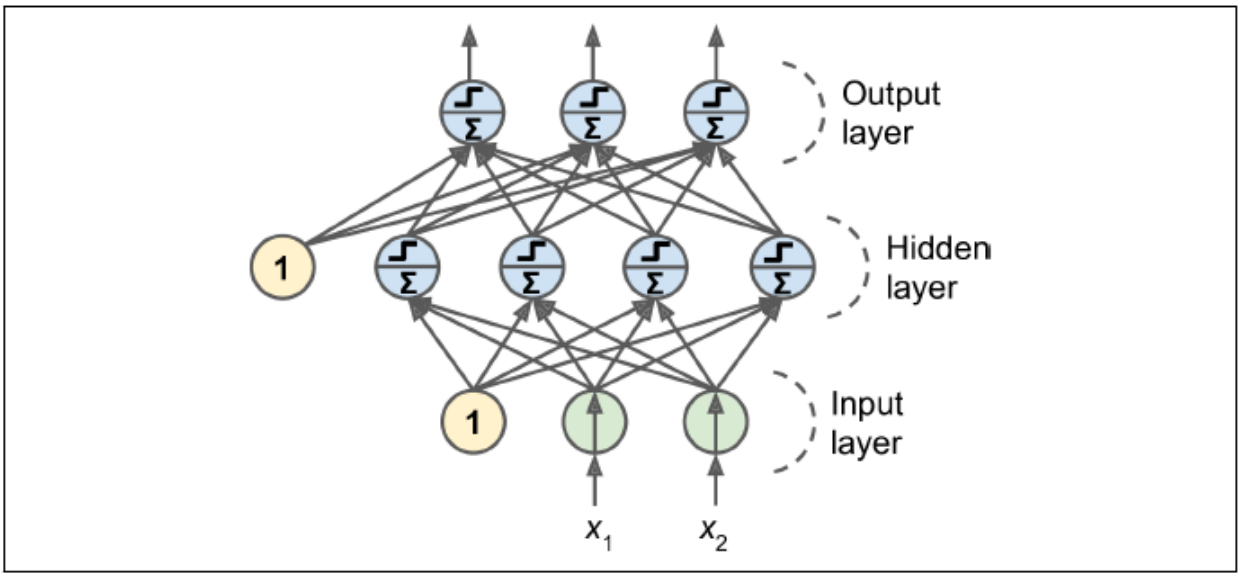
I implemented XGBoost algorithm by a python library named xgboost. The slow speed of the XGBoost left me a very deep impression, and it took 270 seconds to train the model. It much longer than Naïve Bayes and Logistic Regression. The accuracy of XGBoost is very good, and it reached 92.6 percent accuracy. Compared with Naïve Bayes and Logistic Regression, the speed of the XGBoost is much smaller than both of Naïve Bayes and Logistic Regression, and the accuracy of XGBoost is much better than Logistic Regression, but it a little bit worse than Naïve Bayes. We



can see XGBoost get a good result from confusion matrix, and all classes were predicted very well.

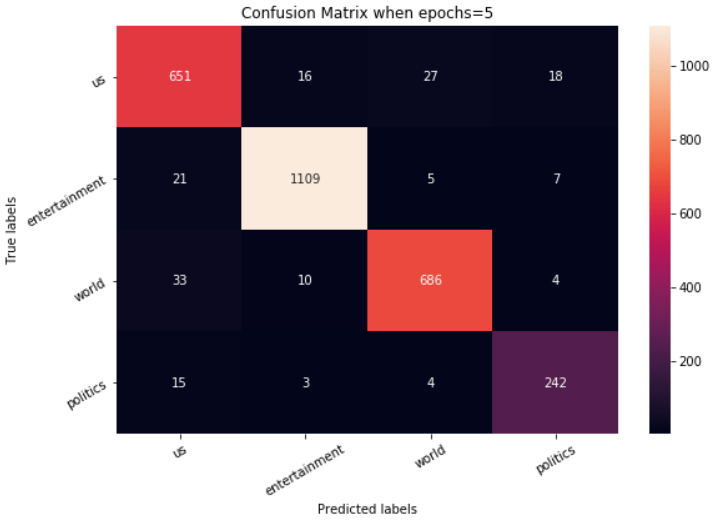
7. Forward Neural Network

Deep Learning is a very hot topic in recent years, and it performs very well in image recognition and natural language processing areas. In my project, I built two kinds of nerual network, forward nerual network and LSTM. First, I built a forwar nerual networks with two hidden layers, and the following imagine show how the forwad neural network works.



In the above imagine, there are two functions in the hidden layers. $\sum$ is the weight function for each parameter. ε is the activation function that make the output of the $\sum$ into a specific range.I chose Relu activation function, and it is the most popular activation. Relu makes the output of $\sum$ fall into range between 0 to infinity.

In my project, I built a four layers forward neural network, one input layer, two hidden layers and one out layer. I trained this model in two times, and in the first time, I set epoch equal to 5, and epoch equal to 10 at the second time. Normally, it will get the better result when people set the epoch larger because larger epoch makes model fit better, but too larger of epoch will get a overfitting model. The resluts of these two different epochs are very close in my project, and the accuarcy is 94.3 percent when epcoh equalled to 5, and this is a little bit tiny better than when epoch equalled to 10. I got 94.2 percent accuracy in 10 times epoch. The accuracy of forwad neural network is much better than Logistic Regression and XGBoost.

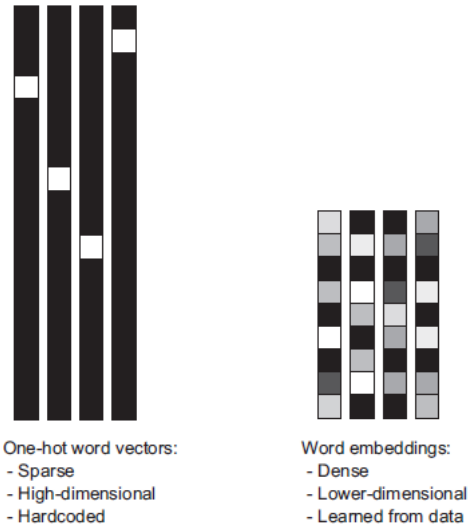


This is the confusion matrix of the forward neural network when epoch equalled to 5. This model performed very well in all of the lables. This is the best model for news classification until now, but one problem is that it takes a lot of time to train a forwad neural network model, and the speed of training model depends on researchers' devices, especially GPU or TPU, and this is the common problem for neural network because there are too many parameters created by weighted function $\sum$.

Next, I will introduce another neural network, and it built on the Recurrent Neural Network (RNN) named Long short-term memory (LSTM). I used another word vectorization method replace TF-IDF when I implemnt LSTM, because the dimension of TF-IDF was too high, and individual device couldn't work, and this words vectorization method is Word Embedding.

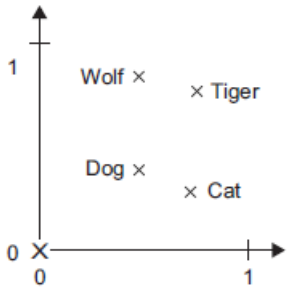
8. LSTM and Word Embedding

Let me introduce word embedding first. One-hot encoding are binary, sparse (mostly made of zeros), and very high dimensional, word embedding are low dimensional floating-point vectors (that is, dense vector, as opposed to sparse vectors). Unlike the word vectors obtained via one-hot encoding, word embedding are learned from data. In a word, word embedding pack more information into far fewer dimension[1]. In word embedding, the geometric relationships between words vectors should reflect the semantic relationships between these words. Word



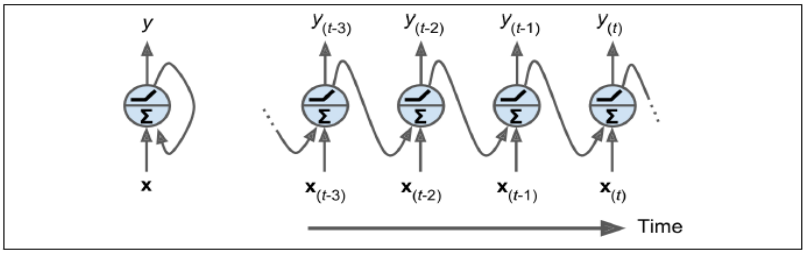
[1] Deep Learning with Python; Francois Chollet

embedding are meant to map human language into a geometric space. For instance, in a reasonable embedding space, people would expect synonyms to be embedded into similar word vectors, in the other word, people would expect the geometric distance(such as L2 distance) between any two word vectors to relate to the semantic distance between the associated words[1].



In the left image, there are four words embedded in the left 2D space: cat, dog, wolf and tiger. With the vector representations we chose here, some semantic relationships between these words can be encoded as geometric transformations. For instance, the same vector allows us to go from cat to tiger and from dog to wolf: this vector cloud be interpreted as the "from pet to wild animal" vector. Similarly, another vector lets us go from dog to cat and from wolf to tiger, which could be interpreted as a "from canine to feline" vector. Stanford University offers a word embedding set with 100-dimensional embedding vector for 400,000 words named Glove[2].

Next, I will introduce a new neural network named LSTM which built on Recurrent Neural Network (RNN), and how LSTM works on word embedding. RNN is a type of ANN where the input size is variant and the features output depends on previous outputs, and LSTM is a more complicated and deeper RNN. A recurrent neural network looks very similar to a forward neural network, except it also has connections pointing backward. Let's look at the simplest possible RNN, composed of one neuron receiving inputs, producing and output, and sending that output back to itself like following image. At each time step t, this recurrent neuron receives the inputs $X_{(t)}$ as well as its own output from the previous time step, $y_{(t-1)}$. Since there is no previous output at the first step, it is generally set to 0. We can represent this tiny network against the time axis, as shown in the following image. This is called unrolling the network through time (it's the same recurrent neuron represented once per time step)[3].



[1] Deep Learning with Python; Francois Chollet

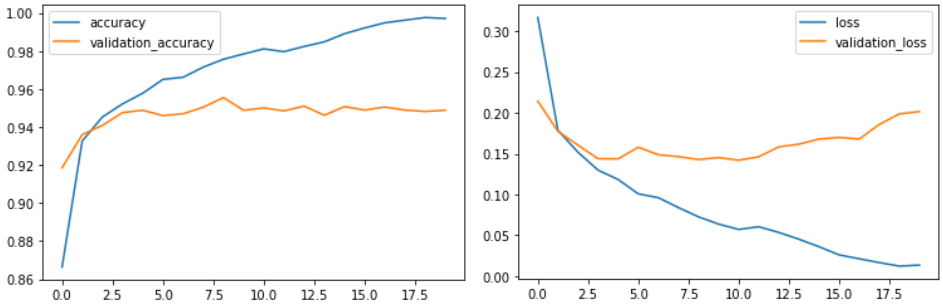
[2] https://nlp.stanford.edu/projects/glove

[3] Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron

Each recurrent neuron has two sets of weights: one for the inputs x_t and the other for the outputs of the previous time step, $y_{(t-1)}$. Let's call these weight vectors w_x and w_y. If we consider the whole recurrent layer instead of just one recurrent neuron, we can place all the weight vectors in two weight matrices, W_x and W_y, and b is the bias vector and $\varphi(\cdot)$ is the activation function (that is Relu)[1]. Here is the equation of output of recurrent layer for a single instance:

$$y_t = \varphi(W_x^T X_{(t)} + W_y^T y_{(t-1)} + b)$$

I implemented LSTM by Keras library with a single LSTM layer model. The result looks pretty good. It got 95 percent accuracy, and it is a little bit better than forward neural network.



RNN is a good neural network on news classification although we can find LSTM has overfitting problem from accuracy and loss images above, and forward neural network has the same problem. But it can not cover up theirs good performances.

9. Latent Dirichlet Allocation (LDA)

LDA algorithm can extract the topic from the document. LDA creates a semantic vector space model. You manually allocated words to topics based on how often they occurred together in the same document. The topic mix for a document can then be determined by the word mixtures in each topic by which topic those words were assigned to. This makes an LDA topic model much easier to understand because the words assigned to topics and topics assigned to documents tend to make more sense. LDA assume that each document is mixture (linear

[1] Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron

combination) of some arbitrary number of topics that you select when you begin training the LDA model. LDA also assume that each topic can be represented by a distribution of words (term frequencies). The probability or weight for each of these topics within a document, as well as the probability of a word being assigned to a topic, is assumed to start with a Dirichlet probability distribution. That is where the algorithm gets its name[1].

The idea of LDA is a really complicated but interesting. In short, LDA came up with the idea by flipping dice for the head. The researchers imagined how a machine that could do nothing more than roll dice could write the documents in a corpus you wan to analyze because you are only working with bags of words, they cut out the part about sequencing those words together to make sense, to write a real document. They just modeled the statistic for the mix of words that would become a part of a particular Bag of Words model for each document.

They imagined a machine that only had two choices to make to get started generating the mix of words for a particular document. They imagined that the document generator chose those words randomly, with some probability distribution over the possible choices, like choosing the number of sides of the dice and the combination of dice you add together to create a D&D character sheet. Your document "character sheet" needs only two rolls of the dice. But the dice are large and there are several of them, with complicated rules about how they are combined to produce the desired probabilities for the different values you want. You want particular probability distributions for the number of words and number of topics so that it matches the distribution of these values in real documents analyzed by humans for their topics and words[2].

In my project, I implemented LDA algorithm by genism and nltk libraries, and the results look not bad, there are plenty of key words of the topics of the news in the result, but there are a lot of irrelevant words appearing in the results, but these irrelevant words always occurred in the news. But in a word, this was an interesting try.

---

[1] Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition; Daniel Jurafsky, James H. Martin

[2] Latent Dirichlet Allocation; David M. Blei, Andrew Y. Ng, Michael I. Jordan

10. Recommendation System

We always can see there are some recommendation of the news when we read a new, and they are always very similar and relevant. At first, recommendation system makes each of the new into a vector, and then it will compute the measurement of similarity or we can say the distance between the target new to other news. We already finished word vectorization by TF-IDF method. We just need to find a suitable function to compute the similarity between each vector, and there are numerous of formulas who can compute the similarity, for instance, Manhattan Distance, Euclidean Distance, Jaccard Coefficient, etc. I chose the Cosine Similarity to compute the similarities between two documents because the performance of Cosine Similarity performs very good and it is easy to implement. Here is the equation of Cosine Similarity:

$$cos(x, y) = \frac{\mathrm{x} \cdot \mathrm{y}}{||\mathrm{x}|| \cdot ||\mathrm{y}||}$$

Where x and y are two document vectors, $\cdot$ indicates the vector dot product, $\mathrm{x} \cdot \mathrm{y} = \sum_{k=1}^{n} x_k y_k$, and ||x|| is the length of vector[1].

11. Summary

This project gives me numerous of interesting experiences, and I learned lots of things about natural language processing because this is the first time that I worked on the natural language processing. I tried to build some new algorithms which I never used before, for instance, XGBoost, and LSTM. I learned how to handle the document feature by word embedding method that I never tried before. I tried to build the LDA model, and the idea of LDA is very interesting and it makes me impressive. Finally, compared with these five algorithms, I know Logistic Regression and LSTM perform pretty good in document classification. Thank you professor!

[1] Introduction to Data Mining; Pan-Ning Tan, Michael Steinbach, Vipin Kumar

Reference

1. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition; Daniel Jurafsky, James H. Martin

2. Introduction to Data Mining; Pan-Ning Tan, Michael Steinbach, Vipin Kumar

3. Hands on Machine Learning with Scikit Learn & Tensorflow; Auelien Geron

4. Deep Learning with Python; Francois Chollet

5. Natural Language Processing in Action Understanding, analyzing, and generating text with Python; Hobson Lane, Cole Howard, Hannes Max Hapke, Arwen Griffioen

6. Latent Dirichlet Allocation; David M. Blei, Andrew Y. Ng, Michael I. Jordan

7. https://lxml.de/index.html

8. https://sites.google.com/a/chromium.org/chromedriver/downloads

9. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

10. https://xgboost.readthedocs.io/

11. https://tensorflow.google.cn/