**CSYE 7374-Parallel Machine Learning & AI**

**Final Project Report**

**Predict Political Advertisements from Facebook by Learning from the Dataset Collected from the Facebook Users**

**Team Number:**

No.4

**Team Members:**

Yi Ren 001050300

Yuchen Zhao 001089667

**Instructor:**

Dr. Liu Handan

Content

[I. Instruction: 3](#_Toc47644739)

[ⅰ. Background: 3](#_Toc47644740)

[ⅱ. Motivations: 3](#_Toc47644741)

[ⅲ. Goal: 4](#_Toc47644742)

[Ⅱ. Methodology: 4](#_Toc47644743)

[ⅰ. Method Introduction: 4](#_Toc47644744)

[ⅱ. Specific Implementation: 8](#_Toc47644745)

[i) Naive Bayes Algorithm: 8](#_Toc47644746)

[ii) TensorFlow: 12](#_Toc47644747)

[iii) K-fold Cross Validation & Data Parallelism: 13](#_Toc47644748)

[iv) Neural network & GPU data parallelism: 15](#_Toc47644749)

[Ⅲ. Data Sources (download link): 17](#_Toc47644750)

[Ⅳ. Results and Analysis: 17](#_Toc47644751)

[Plan I: Linear regression under the Tensorflow framework 17](#_Toc47644752)

[ⅰ. Data Processing: 18](#_Toc47644753)

[ⅱ. Linear Regression: 18](#_Toc47644754)

[Plan II: Visualization & Comprehensive Methods 19](#_Toc47644755)

[ⅰ. Visualization: 19](#_Toc47644756)

[ⅱ. Data Processing: 23](#_Toc47644757)

[ⅲ. Modules: 24](#_Toc47644758)

[Plan III: Naive Bayes and K-fold (Cross-validation) [CPU] 26](#_Toc47644759)

[i. Naïve Bayes: 26](#_Toc47644760)

[ii. K-fold (Cross-validation) & data parallelism: 27](#_Toc47644761)

[Plan Ⅳ: Neural Network [GPU] 28](#_Toc47644762)

[Ⅴ. Conclusion: 29](#_Toc47644763)

[ⅰ. Linear regression under the Tensorflow framework: 29](#_Toc47644764)

[ⅱ. Visualization & Comprehensive Methods: 29](#_Toc47644765)

[ⅲ. Naive Bayes and K-fold (Cross-validation) [CPU]: 29](#_Toc47644766)

[ⅳ. Neural Network [GPU]: 30](#_Toc47644767)

[ⅴ. Gain and experience: 30](#_Toc47644768)

[Ⅵ. Reference: 31](#_Toc47644769)

# I. Instruction:

## ⅰ. Background:

In November this year, the United States will have a new presidential election. Therefore, our group chose to study and predict whether the ads on Facebook are political. The data set selected is Kaggle’s research on related issues. The data level is 3 GB. The data format is CSV, which contains 24 feature values.

This database, updated daily, contains ads that ran on Facebook and were submitted by thousands of ProPublica users from around the world. We asked our readers to install browser extensions that automatically collected advertisements on their Facebook pages and sent them to our servers. We then used a machine learning classifier to identify which ads were likely political and included them in this dataset.

The reason why our team conducts research on the prediction of political advertisements on Facebook is that on the one hand, it mainly combines current political factors. On the other hand, there are relatively few cases of this type of data analysis before, so we can gain new knowledge through learning and innovation.

Content:

This database is created by Andriy Samoshin, he collected the database using the browser extensions. The newest date is on 03-27-2019. This database is very authoritative and reliable.

The data is explained by the following factors:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | HTML | Political | Not\_political | Title | Message |
| Thumbnail | Created\_at | Updated\_at | Lang | Images | Impressions |
| Political\_  probalitity | Targeting | Suppressed | Targets | Advertiser | Entities |
| Page | Lower\_page | Targetings | Paid\_for\_by | Targetedness | Listbuilding\_fundrasing\_proba |

## ⅱ. Motivations:

Our team wants to predict Political Advertisements from Facebook by Learning from the Dataset Collected from the Facebook Users. The 2020 US election is about to take place, so this forecast also has certain political timeliness. We want to use many different methodologies to finish this research.

## ⅲ. Goal:

We hope to use machine learning data with parallel computing to successfully predict whether Facebook ads are political.

# Ⅱ. Methodology:

## ⅰ. Method Introduction:

At first, we use some ways to make the data visible in order to distinguish the features which can be used in our research.

After seeing the 24 features ​​of this data set, we discussed whether to use classifier or regression analysis to learn and predict the problem. After repeated practice and repeated determination, we decided to use both regression, which the label remains a value of probability, and classifier, which the label of probability is set to 0 which refers to not political and 1 which refers to political.

**TensorFlow:**

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.‍

**Snowball Stemmer:**

The stemming method in natural language processing. In our application, we can use this to extract the stem and finally get the desired keyword and keyword ratio. It is an efficient tool.

**Word Cloud:**

Before talking about word cloud, we must first understand a noun, word cloud diagram, what is a word cloud diagram?

A word cloud image, also called a word cloud, is a visual display of the "keywords" that appear frequently in the text. The word cloud image filters out a large amount of low-frequency and low-quality text information, so that the viewer only needs to scan the text at a glance. Understand the main point of the text.

**k Nearest Neighbors:**

In the field of pattern recognition, the nearest neighbor method (KNN algorithm, also translated as K-nearest neighbor algorithm) is a non-parametric statistical method for classification and regression [1]. In both cases, the input contains the k closest training samples in the Feature Space.

In k-NN classification, the output is a classification group. The classification of an object is determined by the "majority vote" of its neighbors. The most common classification among the k nearest neighbors (k is a positive integer, usually small) determines the category assigned to the object. If k = 1, then the category of the object is directly assigned by the nearest node.

In k-NN regression, the output is the attribute value of the object. This value is the average of the values ​​of its k nearest neighbors.

The nearest neighbor method uses the vector space model to classify. The concept is that the cases of the same category have high similarity to each other, and the possible classification of cases of unknown categories can be evaluated by calculating the similarity with cases of known categories.

K-NN is an instance-based learning, or a local approximation and lazy learning that postpones all calculations until classification. The k-nearest neighbor algorithm is one of the simplest among all machine learning algorithms.

**Decision Tree:**

Decision Tree and its variants are another algorithm that divides the input space into different regions with independent parameters. Decision tree classification algorithm is a case-based inductive learning method, which can extract a tree classification model from a given disordered training sample. Each non-leaf node in the tree records which feature is used to judge the category, and each leaf node represents the category finally judged. The root node to each leaf node forms a classified path rule. However, when testing a new sample, it is only necessary to start from the root node, test at each branch node, recursively enter the subtree along the corresponding branch and then test until reaching the leaf node. The category represented by the leaf node is the predicted category of the current test sample.

**Random Forest:**

Like other models, random forest can explain the effects of several independent variables (X1, X2, ..., Xk) on dependent variable Y. If the dependent variable Y has N observation values and K independent variables are related to it; When constructing the classification tree, the random forest will randomly reselect N observation values in the original data, some of which are selected many times and some are not selected. This is Bootstrap resampling method. At the same time, the random forest randomly selects some variables from K independent variables to determine the nodes of the classification tree. In this way, the classification tree may be different each time. In general, random forests randomly generate hundreds to thousands of classification trees, and then select the tree with the highest degree of repetition as the final result (Breiman, 2001).

**Logistic Regression:**

Logistic regression is a generalized linear model, so it has many similarities with multiple linear regression analysis. Their model forms are basically the same, and they all have w 'x + b, where w and b are parameters to be solved. The difference lies in their different dependent variables. Multiple linear regression directly regards w' x + b as dependent variable, i.e. Y = w 'x + b, while logistic regression corresponds w' x + b to a hidden state p, p = L (w 'x + b) through function L, and then determines the value of dependent variable according to the size of p and 1-p. If L is a logistic function, it is logistic regression, and if L is a polynomial function, it is polynomial regression. [2]

**Naive Bayes:**

The Naive Bayes algorithm is an intuitive method that uses the probability that each attribute belongs to a certain class to make predictions. You can use this supervised learning method to model a predictive modeling problem probabilistically.

Given a class, Naive Bayes assumes that the probability of each attribute belonging to this class is independent of all other attributes, thus simplifying the calculation of probability. This strong assumption produces a fast and effective method.

Given an attribute value, the probability that it belongs to a certain class is called conditional probability. For a given class value, multiply the conditional probability of each attribute to obtain the probability that a data sample belongs to a certain class.

We can calculate the probability that the sample belongs to each class, and then select the class with the highest probability to make predictions.

Usually, we use categorical data to describe Naive Bayes, because it is easy to describe and calculate by ratio. A more useful algorithm that meets our purpose needs to support numerical attributes and assumes that every numerical attribute obeys a normal distribution (distributed on a bell-shaped curve). This is another strong assumption, but it can still give a robust result.

**SGD Classifier:**

Gradient descent method (SGD) is a simple and effective method for judging classifiers (SVM or logistic regression) using convex loss function. Even though SGD has existed in the machine learning community for a long time, it has been widely accepted in recent years.

**Linear Regression:**

The linear regression output is a continuous value, so it is suitable for regression problems. Regression problems are very common in practice, such as predicting continuous values such as housing prices, temperatures and sales. Different from the regression problem, the final output of the model in the classification problem is a discrete value.

**K-fold Cross-Validation:**

In machine learning, the data set A is divided into training set B and test set C. Under the condition of insufficient sample size, in order to make full use of the data set to test the algorithm effect, the data set A is randomly divided into k packets, one packet is taken as the test set at a time, and the remaining k-1 packets are taken as the training set for training.

**Multi-process parallel computing:**

We want to use Multi-process parallel computing to save time and cost. In theory, putting more resources into a task will shorten the task completion time and reduce the potential cost. Solving larger/more complex problems, utilize non-local resources and make better use of the underlying parallel hardware.

**Pytorch uses GPU efficiently:**

At present, GPU has developed to a more mature stage. Using GPU to train deep neural networks can give full play to the capabilities of its thousands of computing cores. In scenarios where massive training data is used, the time spent is greatly reduced and fewer servers are occupied. If a proper deep neural network is reasonably optimized, a GPU card is equivalent to the computing power of dozens or even hundreds of CPU servers. Therefore, GPUs have become the industry's preferred solution for deep learning model training.

How to use GPU? Nowadays, many deep learning tools support GPU computing, and only need to be configured simply when using them. Pytorch supports GPU. You can transfer data from memory to GPU memory through the to(device) function. If there are multiple GPUs, you can also locate which GPU or GPUs. Pytorch generally applies GPU to data structures such as tensors or models (including some network models under torch.nn and models created by themselves).

**Neural Network:**

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. (Neural networks can also extract features that are fed to other algorithms for clustering and classification; so you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

## ⅱ. Specific Implementation:

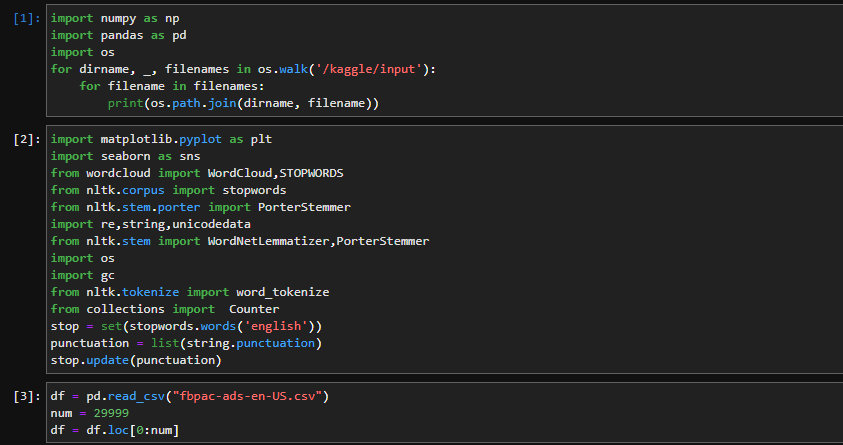
### Naive Bayes Algorithm:

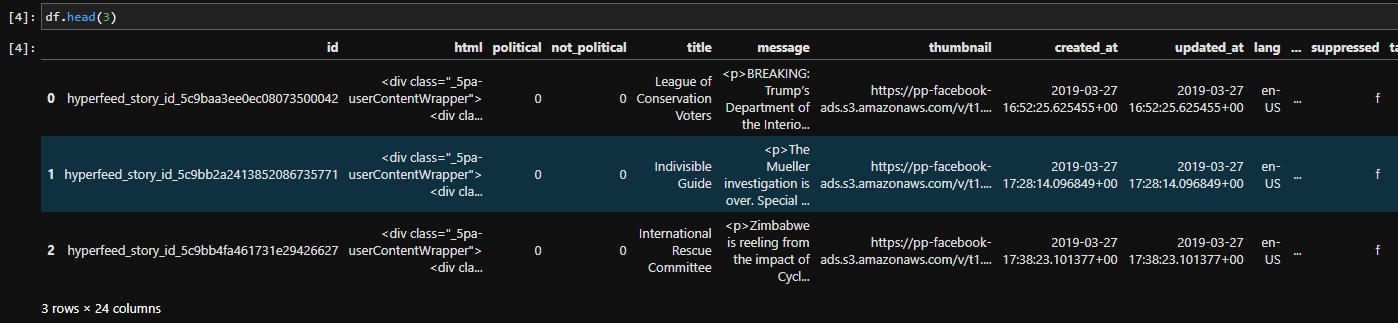
When our group used the naive Bayes algorithm, we did a lot of reading and learning, and we summarized a set of methods for using the algorithm.

There are five steps:

**Step1: Processing data:**

First, several necessary packages are imported and the csv file is read through the pandas method “read\_csv”, and only 30000 rows are used to avoid memory errors:



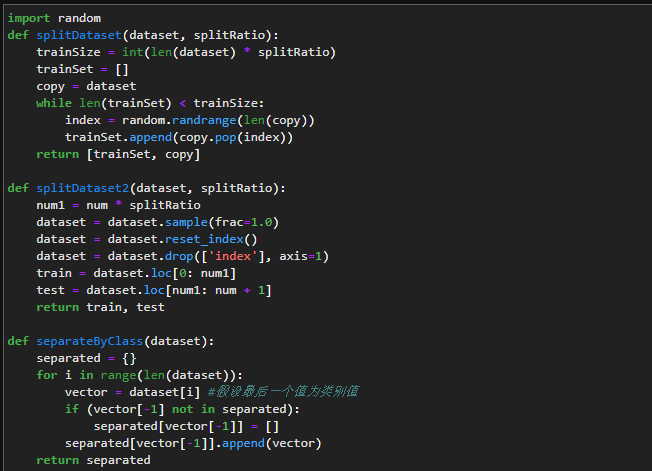


**Step2: Extract data features:**

(1) Divide data by category

First, the samples in the training data set are divided into categories, and then the statistics of each category are calculated. We can create a mapping from a category to a list of samples belonging to this category and classify the samples in the entire data set into the corresponding list.

The following SeparateByClass() function can accomplish this task:

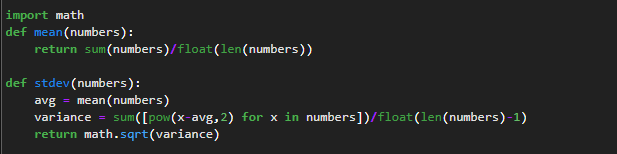


(2) Calculate the mean and standard deviation

We need to calculate the mean value of each attribute in each class. The mean is the midpoint or central tendency of the data. When calculating the probability, we use it as the median of the Gaussian distribution.

We also need to calculate the standard deviation of each attribute in each class. The standard deviation describes the deviation of the data spread. When calculating the probability, we use it to describe the expected spread of each attribute in the Gaussian distribution.

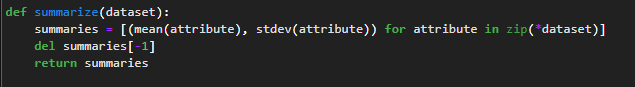
The standard deviation is the square root of the variance. The variance is the average of the squared deviation of each attribute value from the mean. Note that the denominator we use N-1 (the denominator of the unbiased estimate of the sample standard deviation is N-1), that is, when calculating the variance, the number of attribute values ​​minus 1.



(3) Extract the features of the data set

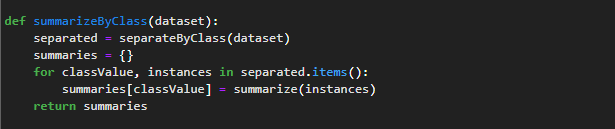
Now we can extract the data set features. For a given list of samples (corresponding to a certain class), we can calculate the mean and standard deviation of each attribute.

The zip function groups data samples into lists according to attributes, and then calculates the mean and standard deviation for each attribute.



(4) Extract attribute features by category

To merge the code, we first divide the training data set into categories, and then calculate the summary of each attribute.



**Step3: Forecast:**

We can now use the summary obtained from the training data to make predictions. Making predictions involves calculating the probability of a given data sample belonging to each class, and then selecting the class with the largest probability as the prediction result.

We can divide this part into the following tasks:

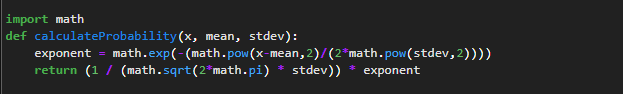
1. Calculate the probability density function of Gaussian distribution (normal distribution)

Given the mean and standard deviation of the known attributes from the training data, we can use the Gaussian function to evaluate the probability of a given attribute value.

Knowing the attribute characteristics of each attribute and class value, under the condition of a given class value, the conditional probability of a given attribute value can be obtained.

For the Gaussian probability density function, you can check the references. In short, we have to integrate the known details into the Gaussian function (attribute value, mean, standard deviation), and get the likelihood that the attribute value belongs to a certain class (translator's note: probability).

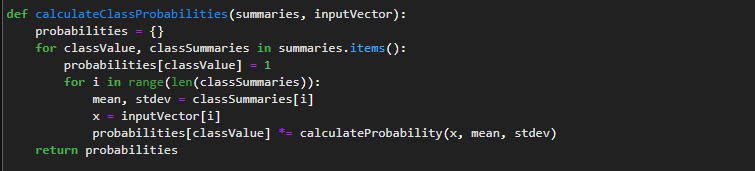
In the calculateProbability() function, we first calculate the exponential part, and then calculate the backbone of the equation. This can be well organized into 2 rows.



(2) Calculate the probability of the class

Since we can calculate the probability that an attribute belongs to a certain class, then combine the probabilities of all the attributes in a data sample, and finally get the probability that the entire data sample belongs to a certain class.

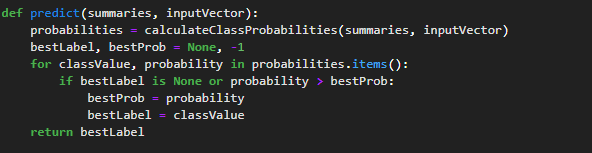
Use multiplication to combine probabilities. In the calculClassProbilities() function below, given a data sample, the probability of each category it belongs to can be obtained by multiplying its attribute probabilities. The result is a mapping from class value to probability.



(3) Single prediction

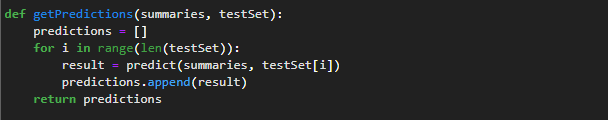
Now that we can calculate the probability that a data sample belongs to each class, we can find the maximum probability value and return the associated class.

The following predict() function can accomplish the above tasks.



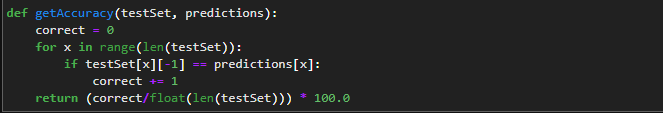
(4) Multiple predictions

Prediction of multiple data samples in the test data set.

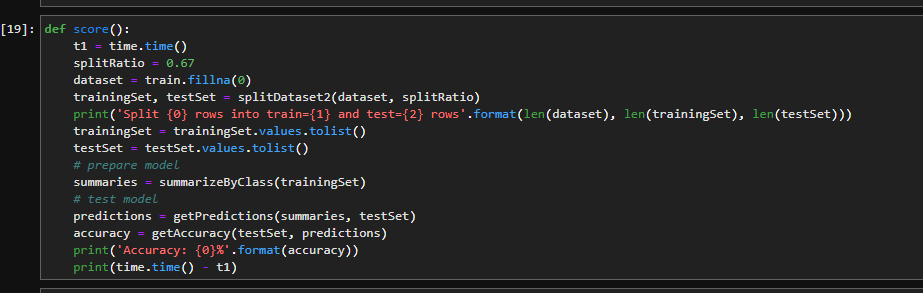


**Step4: Evaluation accuracy:**

The predicted value is compared with the category value in the test data set, and an accuracy rate between 0% and 100% can be calculated as the accuracy of the classification. The getAccuracy() function can calculate this accuracy rate.



executable code:



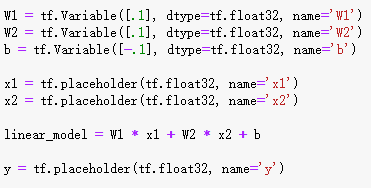
**Step5: Conclusion:**

In our practice of naive Bayes algorithm, our results are not very ideal, but our learning, trying and summarizing process has benefited us a lot, and we will also cross Naive Bayes and K-fold in the follow-up The verification method has been combined with experiments, and some results have been obtained.

### TensorFlow:

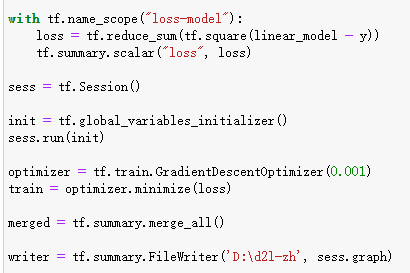
**Step1: Initialize parameters:**

The model we chose to use in this project is y = W1 \* x1 + W2 \* x2 + b. W1, W2 and b are set to a specific value as the one that would be used at first. y, x1 and x2 are given the name but no specific value, and the value will be added after.

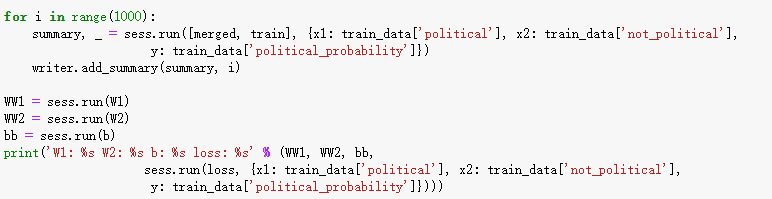


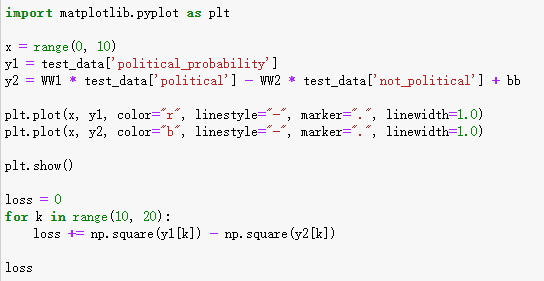
**Step2: Define the model:**

The loss model is defined as the variance of the data. The learning rate is set to 0.001. TensorBoard is also implemented to record the variance with the number of training. Once the variance curve is generated, the training time we need can be decided:



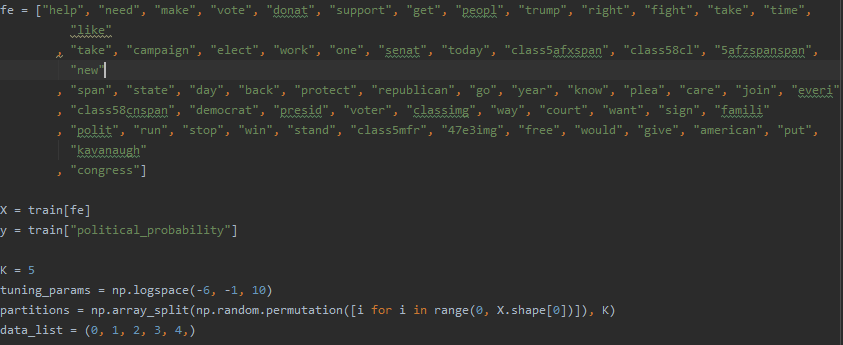
**Step3: Training model and printing the results:**





### K-fold Cross Validation & Data Parallelism:

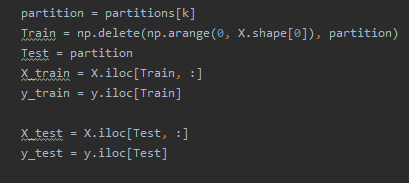
**Step1: Initializing parameters:**



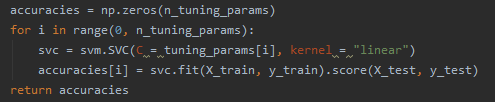
Defining tuning parameter:



Defining testing set and training set:



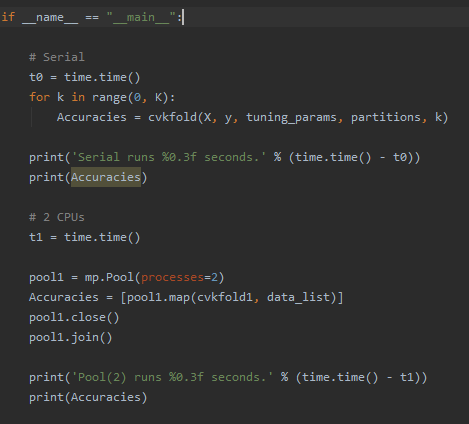
**Step 2: Training model:**



**Step3: Data Parallelism:**

The multiprocessing pool is used to parallelize the operation:

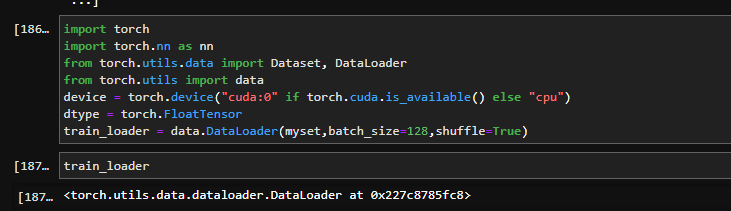
We use parallelize to cut down the time we will use in the calculation. And we set k=5 to separate the data to five pieces. At the meantime, we also use the parallelism to cut down the data processing time. In this class we also learnt how to use parallelism to process the data.



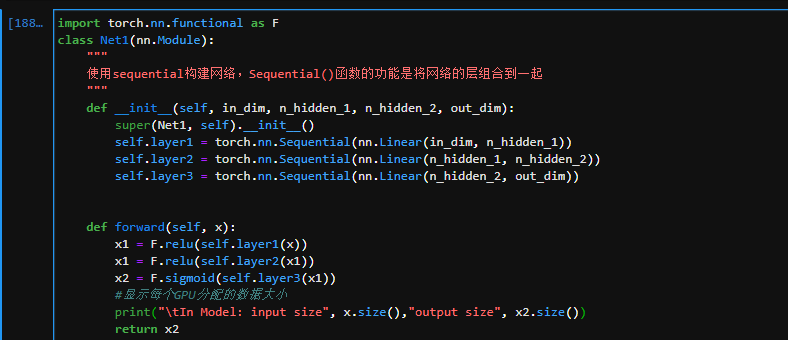
### Neural network & GPU data parallelism:

The DataParallel function, which is mainly used by single-machine multi-GPU, instead of DistributedParallel, is usually used for multi-host multi-GPU, and of course it can also be used for single-machine multi-GPU.

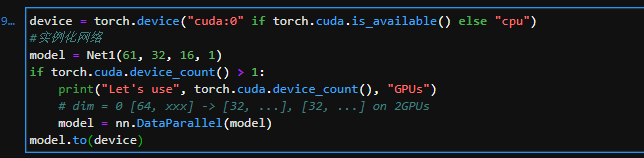
Import the package corresponding to torch, some related tools, and data loading:



Define the neural network which contains one input layer, one output layer and two hidden layers:



Convert the model to a multi-GPU concurrent processing format, the number of the neurons of each layers are set to 61, 32, 16, 1:



Choose optimizer and loss function, the learning rate is set to 0.01 and the loss is set as the Binary Cross-Entropy Loss:

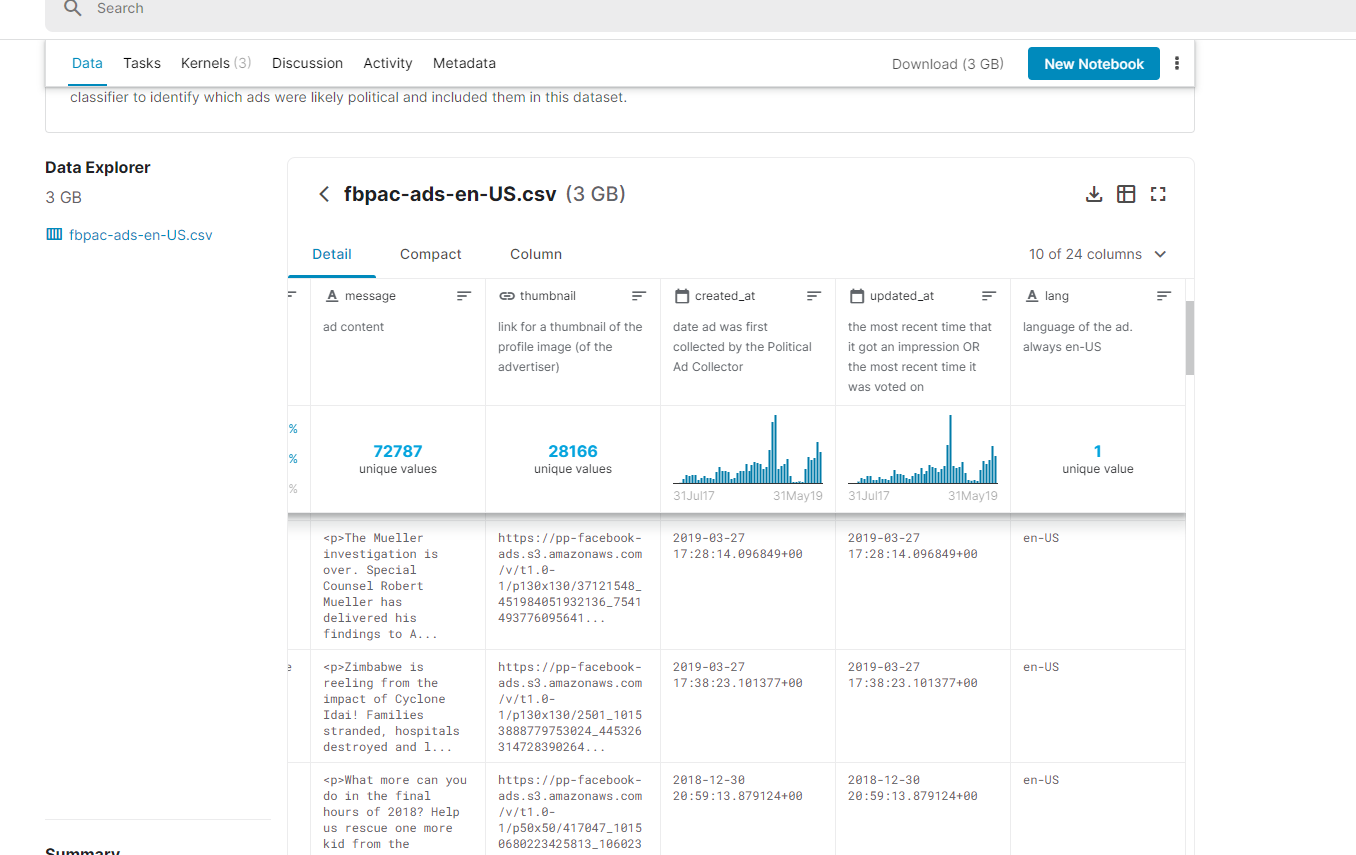


Model training and visualize the loss value.



# Ⅲ. Data Sources (download link):

<https://www.kaggle.com/mrmorj/political-advertisements-from-facebook>



# Ⅳ. Results and Analysis:

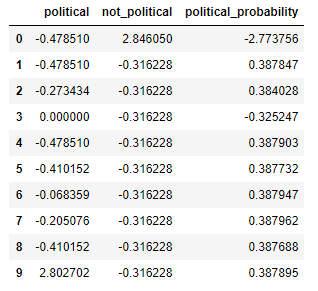
In order to make use of the data, data standardization is necessary to make it easier for us to analysis. It can be seen that there are both number data and text data in our dataset. The features “political” and “not\_political” are first considered since they are both number data and it seems easier to be used in linear regression with the label “political\_probability”. However, the result of the first plan is not ideal. Then the features “title”, “message” and “paid\_for\_by” are considered to be the features and the label “political\_probability” is also turned into “0” or “1” to fit the classifier model. We first used wordcloud to visualize the data and then use tensorflow, naïve bayes, k-fold and neural network to train the module. In this part, these plans will be shown in four parts.

## Plan I: Linear regression under the Tensorflow framework

In plan I, the features “political” and “not\_political” are considered as the input of the system and the label “political\_probability” is considered as the output of the system.

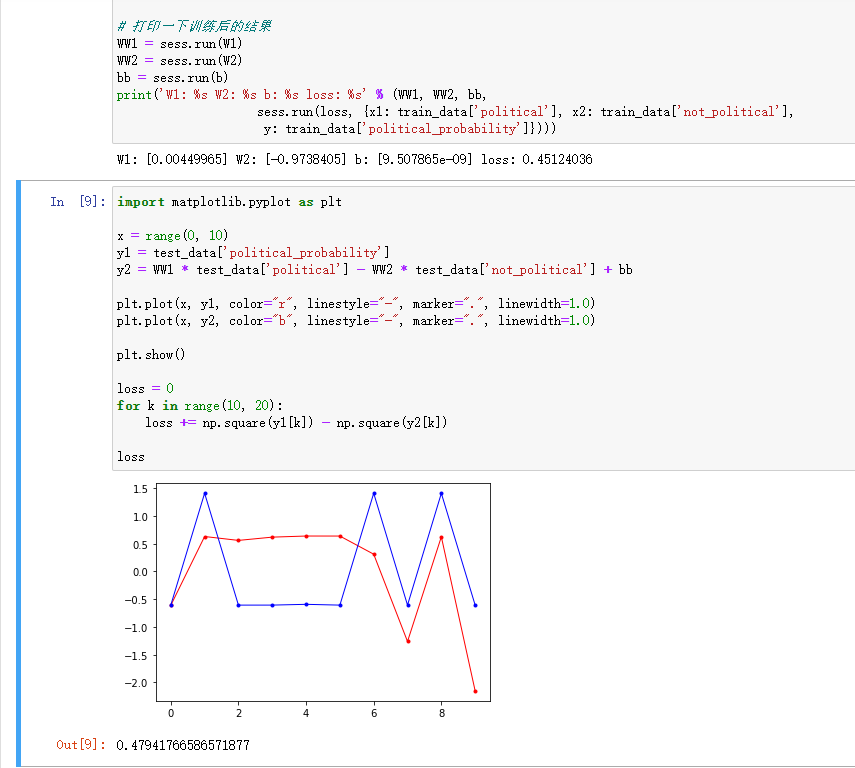
### ⅰ. Data Processing:

First, separate features “political” and “not\_political” from the dataframe which refers to the number of users that vote for the ads. Then, the data are normalized to zero mean and unit variance using (x-mean(x) / std(x)):



### ⅱ. Linear Regression:

Tensorflow is implemented. The linear module is y = W1 \* x1 + W2 \* x2 + b. x1 refers to the feature “political”, x2 refers to the feature “not\_political” and b is the bias. Learning rate is 0.001. The change of loss is written to the tensorboard but for some unknown reason, it only showed up once and can’t be open again, so there is no screenshot about that. 10000 training times is used for the first time and the loss graph that is shown in the tensorflow shows that the loss converges within 1000 iterations. So 1000 times of training is finally used:



As we can see in the screenshot, the loss of the training set is 0.45124036 and that in testing set is 0.47941766586571877. They are close to each other which means the training is effective. However, the plot shows that although the prediction and the labels are similar to each other, they still have some obvious difference. However, when the quantity of sample is set to a bigger one such as 10000, the gradient vanishing or exploding problem became inevitable and it still remains a problem when this report is written. So, plan I is not finished and still needs more discussion. The code of the plan I is attached at the end of the report.

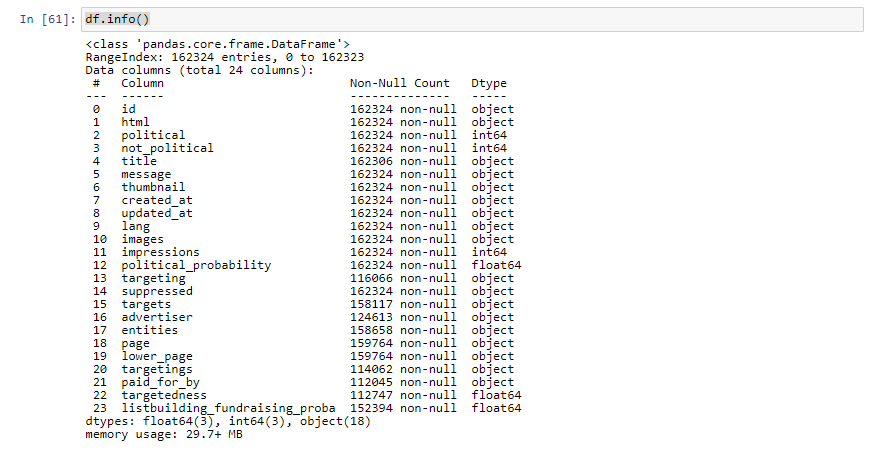
## Plan II: Visualization & Comprehensive Methods

In plan II, the features “title”, “message” and “paid\_for\_by” are considered as input and the label “political\_probability” is considered as output. The data is first visualized to see the word frequency of every words in the features. The classifier packages of the module “sklearn”, including KNN, Decision Tree, Random Forest, Logistic Regression, SGD, Naïve Bayes and SVM Linear, are used to get a quick version of the results.

### ⅰ. Visualization:

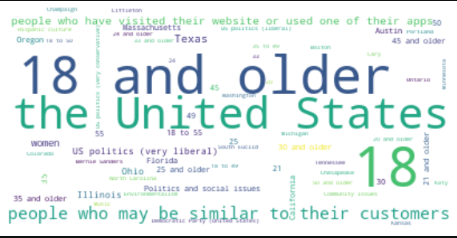
Data information:

Every feature almost has 150000 data, and we will operate based on these data.



Word Cloud:

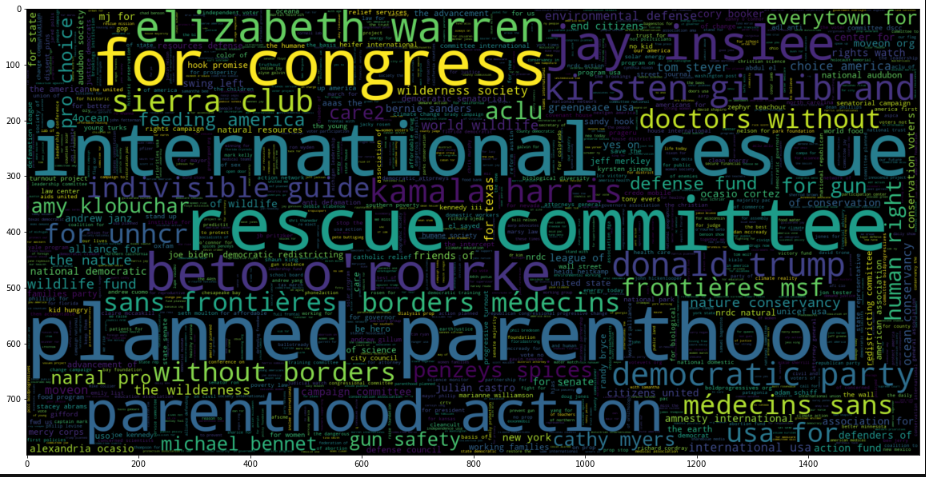
Targetted\_age: We use word cloud to research the targeted age in order to make the targetedness clearer and more pointed.



As we can see the most common age in the targetedness is 18 and older.

Title political word:

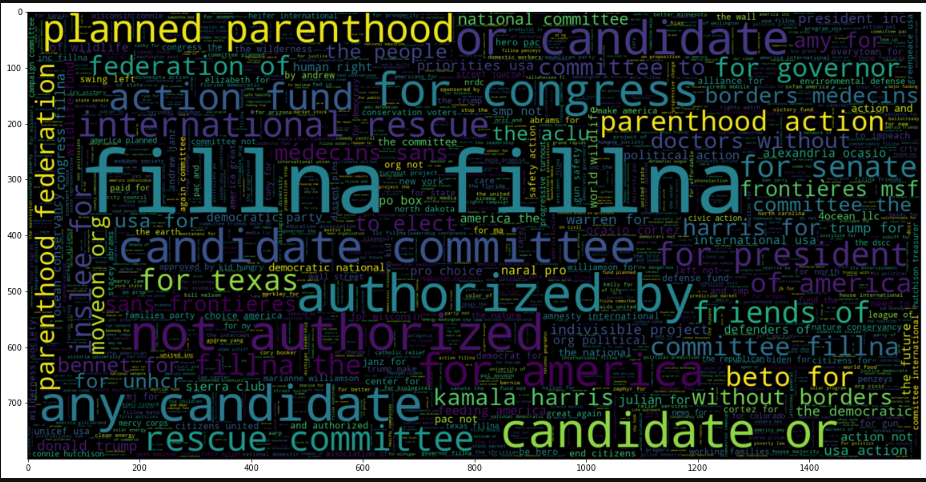
We also use the word cloud to find the most frequent political words in the title.

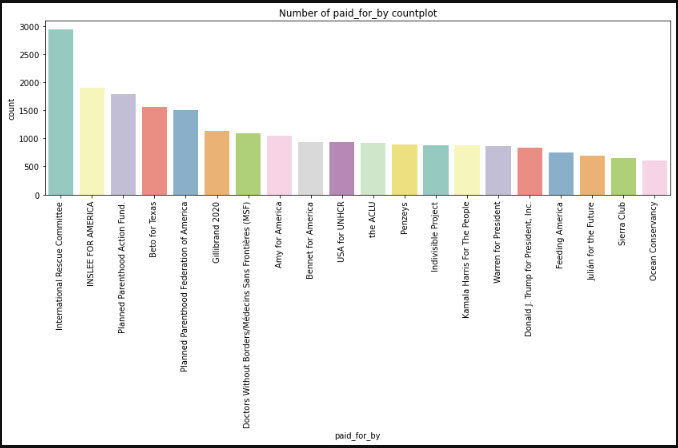


We can see the words, rescue and committee, are very common in the title.

Paid for by:

In this image, we can estimate what is the most advertising payer.

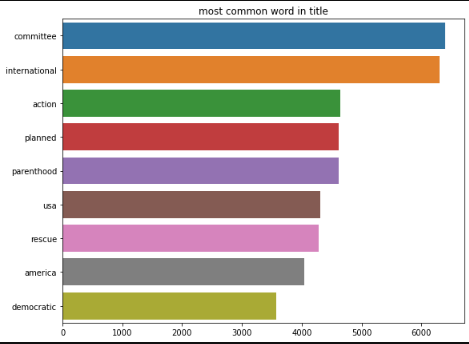




These two images show our investigation and data processing of paid advertisers in different forms.

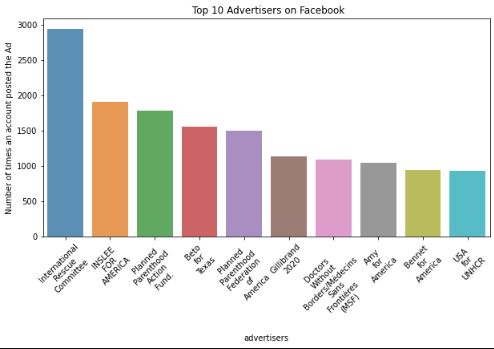
Bar Graph:

Common word in title: In this graph we can see that the most common word and how many times these words appeared in the ads.



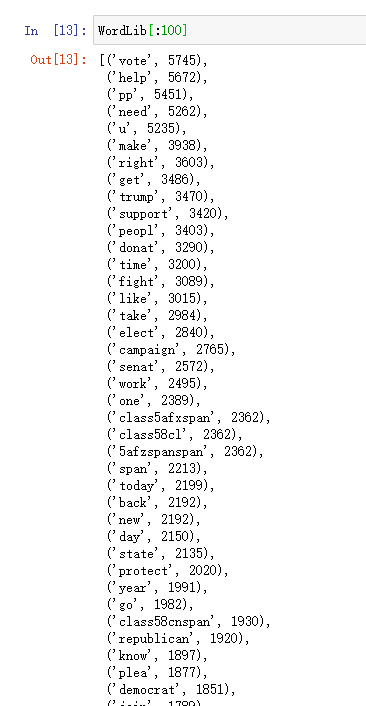
Top 10 advertisers on Facebook:

In this graph we can see which one is the rank of the advertisers.

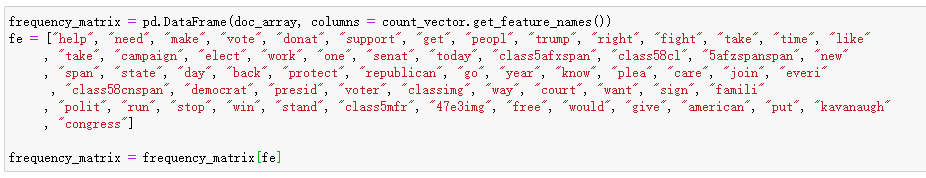


### ⅱ. Data Processing:

While training, the feature “message” is the one that is used because this feature seems to be more relevant to the results than the other two. The most common words in the feature “message” is already figured out in the previous part:



Since too much words may cause “memory error” problem, 62 most common words are used to finish the training:





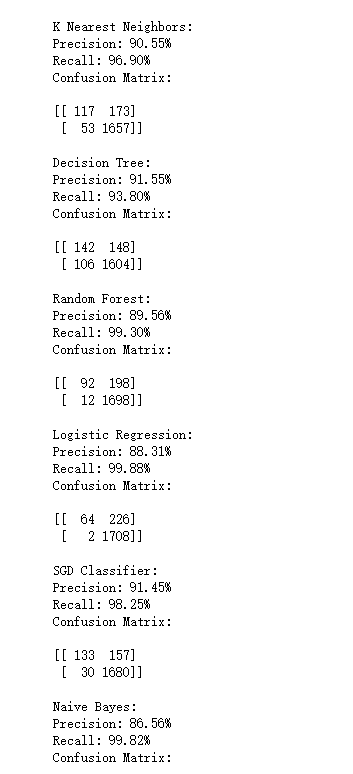
The label “political\_probability” is then set to “0” and “1” referring to not political and political so that we can use a classifier to train the data. Since most of the probability of the label is high, 90% or higher probabilities are set to “1” to ensure there are enough “0” in the data set:

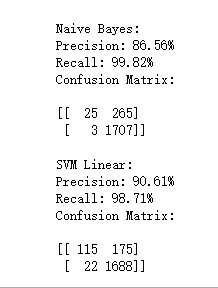


### ⅲ. Modules:

KNN, decision tree, random forest, logistic regression, SGD classifier, Naïve Bayes, SVM Linear are used to train the module and the results are listed:

By using line regression ways, we learn a lot about mathematical principles, we try seven





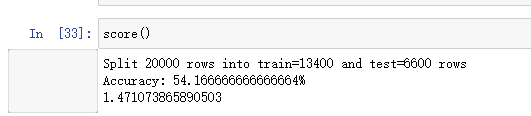
|  |  |  |
| --- | --- | --- |
| Methods | Precision | Recall |
| KNN | 90.55% | 96.90% |
| Decision Tree | 91.55% | 93.80% |
| Random Forest | 89.56% | 99.30% |
| Logistic Regression | 88.31% | 99.88% |
| SGD Classifier | 91.45% | 98.25% |
| Naïve Bayes | 86.56% | 99.82% |
| SVM Linear | 90.61% | 98.71% |

## Plan III: Naive Bayes and K-fold (Cross-validation) [CPU]

The data processing is exactly the same as that in plan II. Then the naïve bayes is accomplished without the sklearn module. The k-fold crossValidation is also used to train the module. The data parallelism of CPU is also implemented to the k-fold crossValidation.

### i. Naïve Bayes:

The training set contains 67% of the dataframe and testing set contains the rest of it. Since the dataset couldn’t be too large or “memory error” will occur, only 20000 samples are used which leads to a low accuracy of 54%:

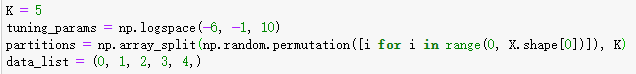


In this case, k-fold crossValidation is used to make some progress.

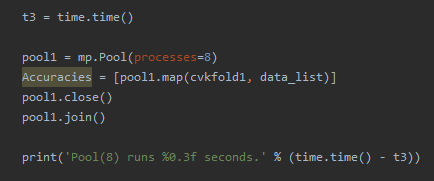
### ii. K-fold (Cross-validation) & data parallelism:

K-fold cross-validation, the initial sample is divided into K sub-samples, a single sub-sample is retained as the data for the verification model, and the other K-1 samples are trained. The cross-validation is repeated K times, and each sub-sample is verified once. The advantage of this method is that it uses randomly generated sub-samples for training and verification at the same time, and each result is verified once. 10-fold cross-validation is the most commonly used.

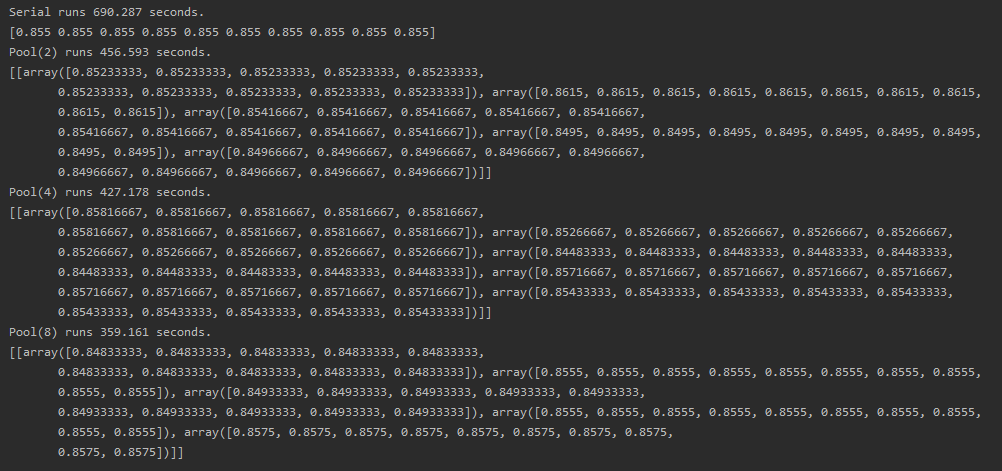
The parameters are set as follow:

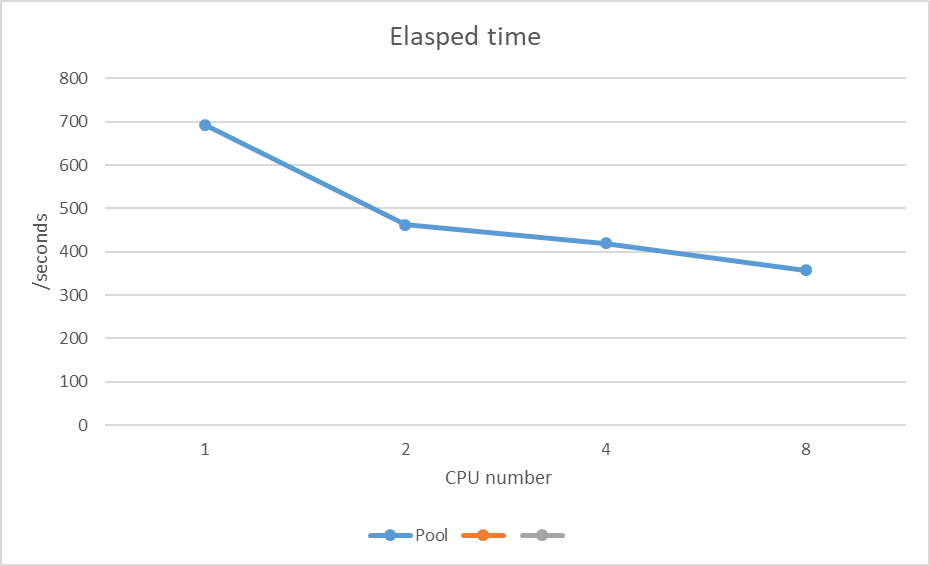


Multiprocessing is also implemented in this part:



The result turns out to be:



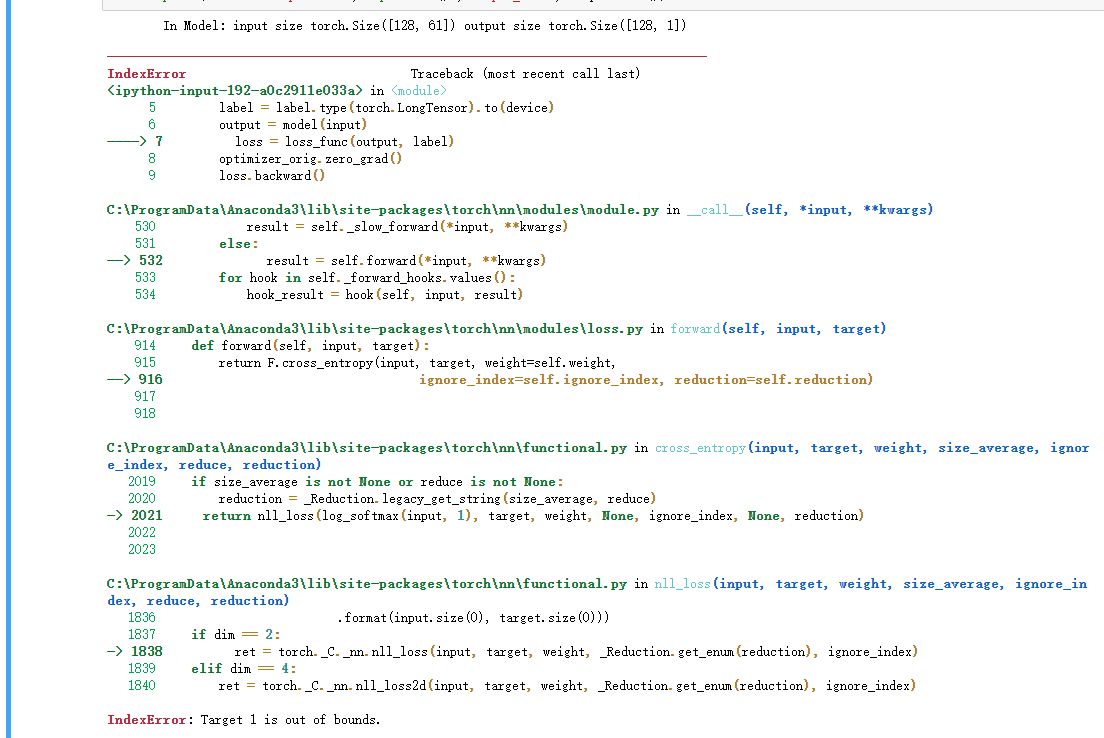


As can be seen in the plot, the more CPUs are used, the faster the operation is, and there is no remarkable difference between the accuracies when using different quantities of CPUs.

## Plan Ⅳ: Neural Network [GPU]

The data processing is exactly the same as that in plan II. Then the naïve bayes is accomplished without the sklearn module.

Unfortunately, we have gone through countless data searches and attempts, but there still remains problems while editing this report. For the FloatTensor data format, our data format cannot be satisfied, so the code cannot run, and the GPU parallel operation is not successfully used:



# Ⅴ. Conclusion:

## ⅰ. Linear regression under the Tensorflow framework:

The loss of the training set is **0.45124036** and that in testing set is **0.47941766586571877**. They are close to each other which means the training is effective. However, the plot shows that although the prediction and the labels are similar to each other, they still have some obvious difference. However, when the quantity of sample is set to a bigger one such as 10000, the gradient vanishing or exploding problem became inevitable and it still remains a problem when this report is written.

## ⅱ. Visualization & Comprehensive Methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Precision | Recall | Average |
| KNN | 90.55% | 96.90% | 93.725% |
| Decision Tree | 91.55% | 93.80% | 92.675% |
| Random Forest | 89.56% | 99.30% | 94.43% |
| Logistic Regression | 88.31% | 99.88% | 94.095% |
| SGD Classifier | 91.45% | 98.25% | 94.85% |
| Naïve Bayes | 86.56% | 99.82% | 93.19% |
| SVM Linear | 90.61% | 98.71% | 94.66% |

It’s a quick version of the training results using the data set. The accuracy is acceptable according to this result which means there is no remarkable problem with our data set.

## ⅲ. Naive Bayes and K-fold (Cross-validation) [CPU]:

Time:

Pool (serial): 690.287s

Pool (2): 456.593s

Pool (4): 427.178s

Pool (8): 359.161s

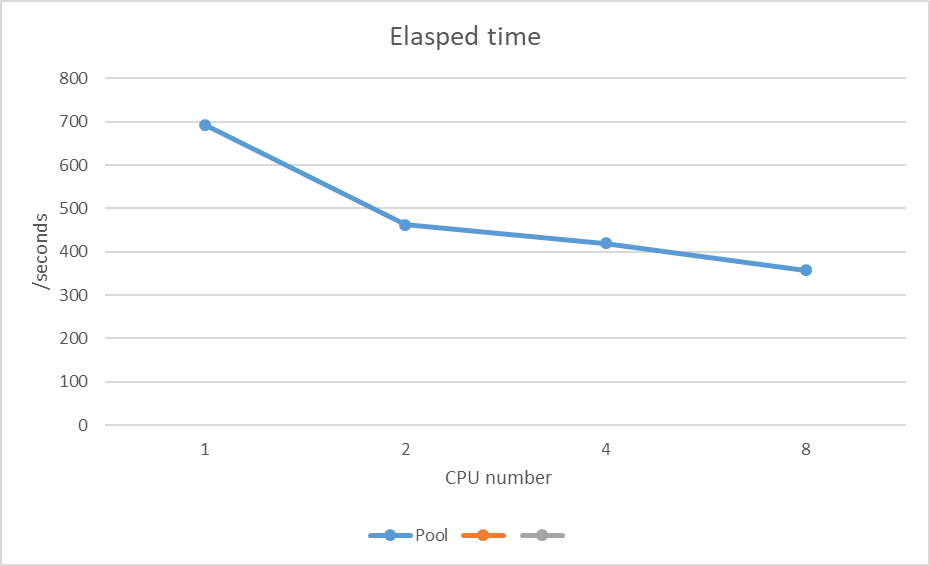
Accuracy:

Pool (serial): 0.855

Pool (2): 0.852

Pool (4): 0.858

Pool (8): 0.848



As the result shows, there is an obvious acceleration in operation, and there is no remarkable difference in the accuracy while using different number of CPUs to do parallelism.

## ⅳ. Neural Network [GPU]:

Because the data types do not match, the GPU parallel operation results cannot be displayed.

## ⅴ. Gain and experience:

Through this project, we learnt a lot about the machine learning and data parallelism. In this project, we tried multiple methods and modules to get to the goal of machine learning and multiprocessing, including Naïve Bayes, K-fold CrossValidation, Tensorflow, Pytorch and so on. Although we’ve learnt a lot theoretically in class this semester, we didn’t practice much in fact, so we faced a lot of difficulties while coding. For example, the Statistics of word frequency is the first one we met which is solved through ntlk and wordcloud. Still, there are many problems we need to figure out, such as the gradient explosion and gradient disappearance while using tensorflow and linear regression to train the data. The time of the project is limited, but the time of learning is not. We’ll keep on learning and figuring out these problems in our future learning.

# Ⅵ. Reference:

[1] Altman, N. S. An introduction to kernel and nearest-neighbor nonparametric

regression. The American Statistician. 1992, 46 (3): 175–185. doi:10.1080/00031305.1992.10475879.

[2] Docs » Installing Software

https://rc-docs.northeastern.edu/en/latest/software/software.html.

[3] OS module related instructions

https://wiki.jikexueyuan.com/project/explore-python/File-Directory/os.html

[4] Python implements naive Bayes classifier from no foundation

https://www.jianshu.com/p/d2745c85bbd4

[5] PackagesNotFoundError: The following packages are not available from current

channels: https://stackoverflow.com/questions/48493505/packagesnotfounderror-

the-following-packages-are-not-available-from-current-cha

[6] Corpora/stopwords not found when import nltk library

https://stackoverflow.com/questions/41610543/corpora-stopwords-not-found-when-import-nltk-library

[7] Chapter 25 How Pytorch uses GPU efficiently

http://www.feiguyunai.com/index.php/2019/04/30/python-ml-25-pytorch-gpu/

[8] Working with GPUs

https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html