**Final Project – Report**

**Abstract**

Customer Relationship Management (CRM) is a key element of modern marketing strategies. The KDD Cup 2009 offered the opportunity to work on large marketing databases from Orange, the French Telecom company. The challenge offered a dataset about a generic problem (classification) which is relevant to the industry, but also presenting a number of scientific and technical challenges of practical interest including: a large number of training examples (50,000) with a large number of missing values (about 60%) and a large number of features (15,000), unbalanced class proportions (fewer than 10% of the examples of the positive class), noisy data, presence of categorical variables with many different values. Besides the work done by the original participants, the subject is still interesting to explore and analyze especially with recent development in Machine Learning and advancement in computation power. Our goal in this project is to try to improve the overall accuracy of customer churn rate prediction based on the dataset with our selection of methodologies in data processing and machine learning.

**Dataset Analysis and Data Preprocessing**

**Dataset Analysis:**

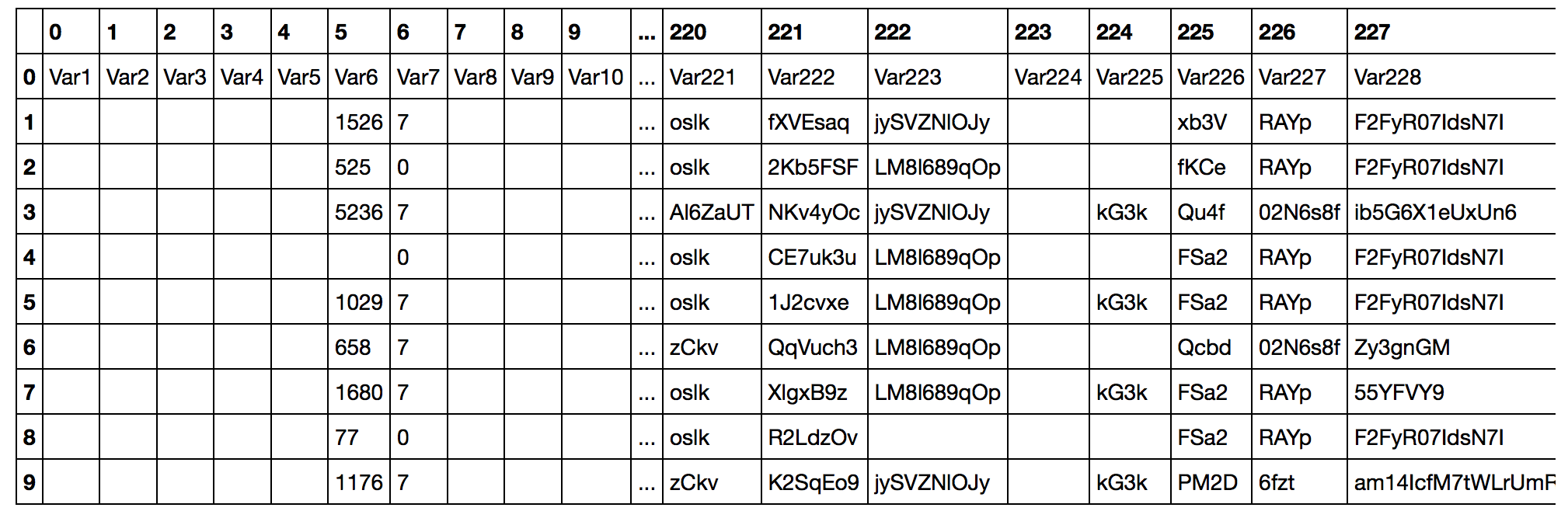
* A large set of data had been made available by Orange Telecom. A small and a large dataset are provided, each of them has 50,000 instances. For each instance, the small dataset has 230 features and the large has 15,000. The dataset offered a variety of difficulties:
* Heterogeneous data (numerical and categorical variables). The small dataset has 40 categorical features (out of 230), while the large dataset has 260 categorical features (out of 15000).
* Noisy data
* Unbalanced distributions of predictive variables, sparse target values (only 1 to 7 percent of the examples belong to the positive class)
* Lots of missing values.

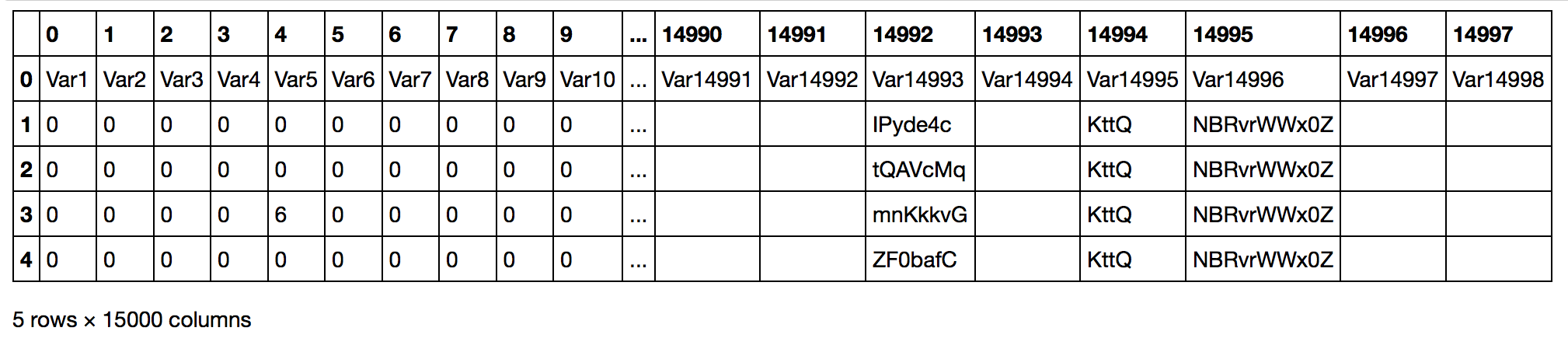
To protect the privacy of the customers whose records were used, the data were anonymized by replacing actual text or labels by meaningless codes and not revealing the meaning of the variables.

**Data Preprocessing**

**Step1: Collect Small Dataset and Label Dataset**

The first step is to split and collect data using **spark**. And then use **pandas.DataFrame** to display the head line of the dataset. Figure 1-1 illustrates the raw datasets for both two versions, for each instance, the small dataset contains 230 features and the large has 15,000. the small dataset contains 230 features and the large has 15,000 There are also Lots of missing values in the raw dataset. Therefore, the next step is to deal with the missing value.

****Figure1-1-small dataset

****Figure1-1-large dataset

**Step2: Processing Missing Data**

From above collected raw data we can find out that there are lots of missing values (about 60%). Besides, both small and large datasets have numerical and categorical variables, For the large dataset, the first **14,740 variables are numerical** and the last **260 are categorical**. For the small dataset, the first **190 variables are numerical** and the last **40 are categorical**. Therefore, we need to deal with the data separately. (We applied the same method to deal with two version datasets, for the large dataset screen shots are in file named **large dataset screenshot**)

* **2.1 dealing with String Empty Data**

****Figure1-2- small dataset

The above figure 1-2 display the data that fill all the string empty to ‘NaN’ for both numerical data and categorical data.

* **2.2 dealing with Numerical Missing Data**

After dealing with the String empty values, we’ll handle the numerical missing data which are the first 190 columns of the raw dataset. The first step is to drop the columns that all consists of NaNs. And then, using the **sklearn.preprocessing.Imputer** method for competing missing values by replacing missing values using the mean along the axis. The figure 1-3 below displays the outcomes, and totally 173 columns.

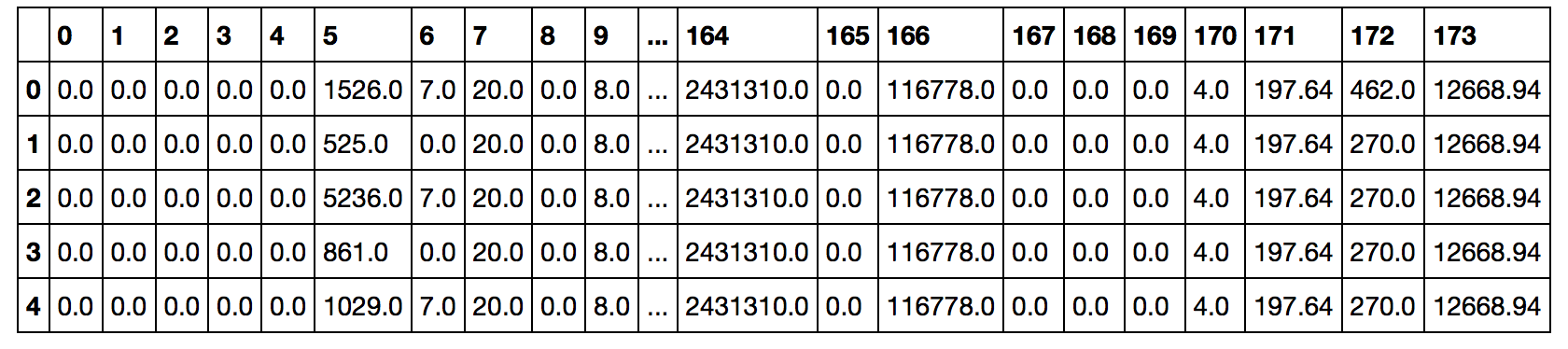
****

Figure1-3- small dataset

* **2.3 Encoding categorical data**

Similarly, for the categorical data, after dropping the NaN columns, there are 37 columns left. For the missing value, we convert text into integer to encode the categorical data. And for the NaN in column, the code is -1. Figure1-4 shows the output of the encoding categorical data.

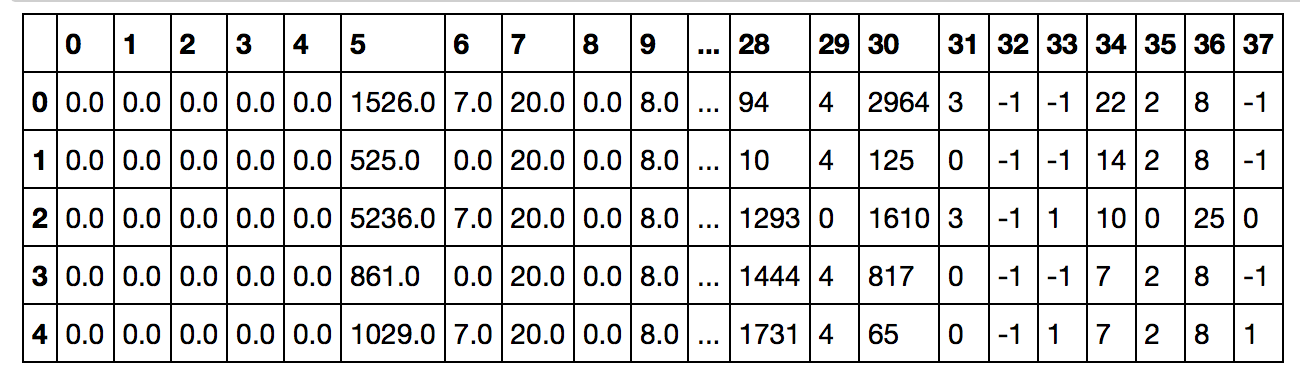
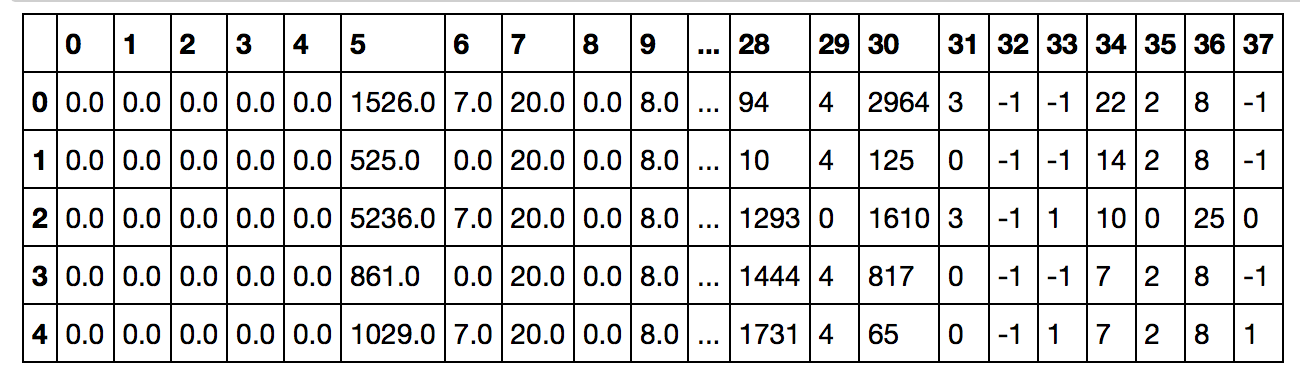
****

Figure1-4- small dataset

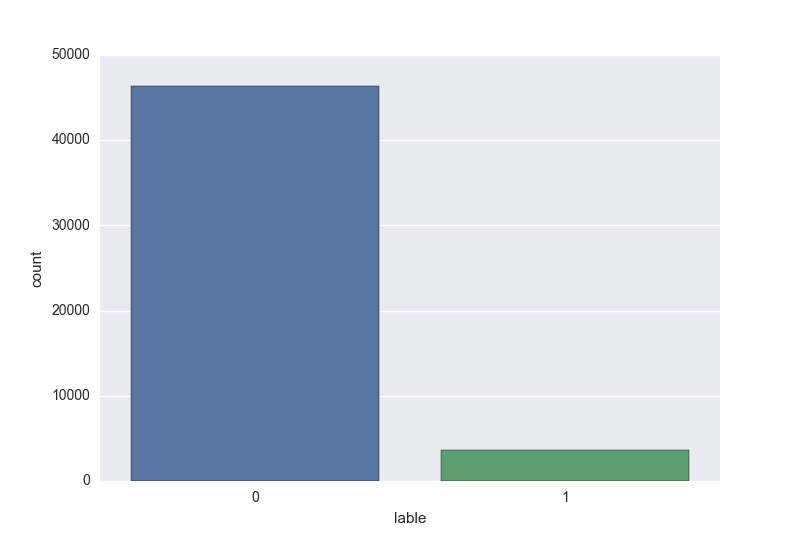
* **2.4 Combine the numerical data and categorical data.**

The following Figure 1-5 displays the fully dataset with both the numerical columns and categorical columns after dealing with the missing data.

****Figure1-5- small dataset

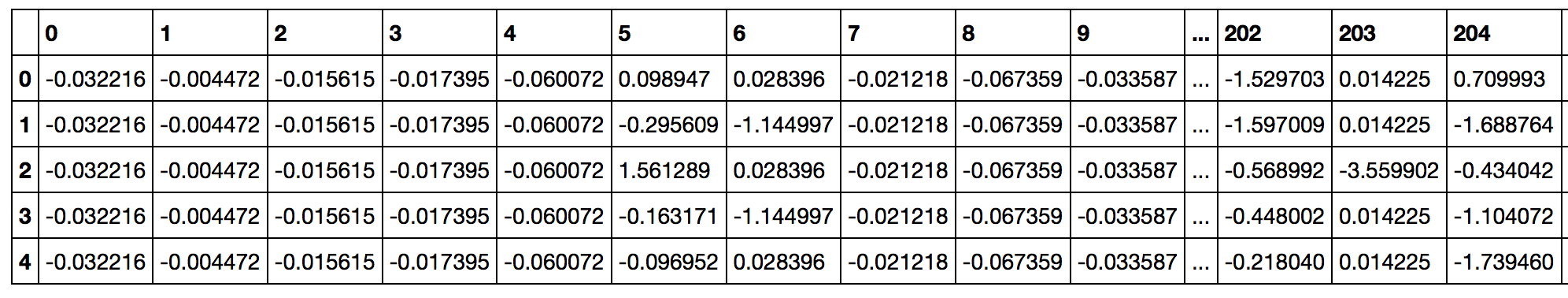
* **2.5** **Label Encoding**

The label data also need to be encoded. We use **LabelEncoder** to encode the label from -1, and 1 to 0 and 1. The challenge of the label data is the unbalanced distributions, sparse target values (only 1 to 7 percent of the examples belong to the positive class). The following figure 1-6 shows the unbalanced predictive variables. (46328: 3672)

**** Figure1-6-small-dataset

### Step3: Feature Scaling- Standardizing

From previous results, we can find out that the scales of the original features are measured on different scales. It makes sense to standardize the data using **StandardScaler( )** function, which is a requirement for the optimal performance of many machine learning algorithms. The following figure demonstrate the dataset after standardizing.

****Figure1-7- small dataset

**Step4: Principal Component Analysis (PCA)**

The main goal of a PCA analysis is to identify patterns in data; PCA aims to detect the correlation between variables. If a strong correlation between variables exists, the attempt to reduce the dimensionality only makes sense. In a word, this is what PCA is all about: Finding the directions of maximum variance in high-dimensional data and project it onto a smaller dimensional subspace while retaining most of the information.

Our two version datasets are high dimensional data ("small" with 230 variables, and "large" with 15,000 variables). It is necessary try to use PCA to reduce the dimensionality. After computing Eigenvectors and Eigenvalues, selecting principal components steps, we generate the following figures to demonstrate how much information(variance) can be attributed to each of the principal components.

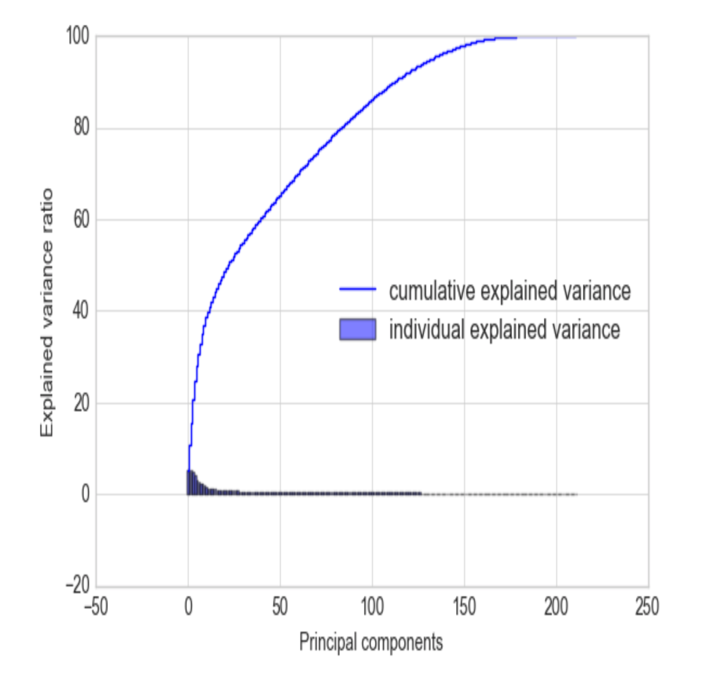
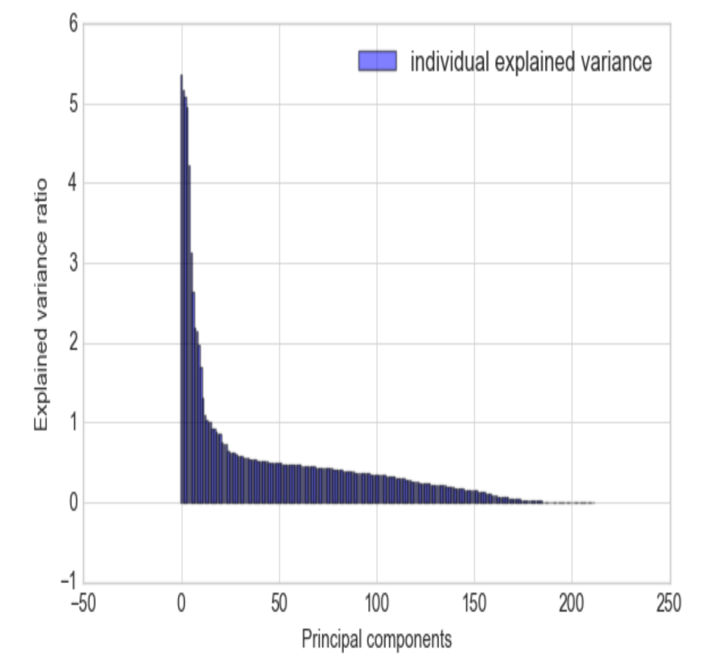
****

Figure1-8 **-** small datasetFigure1-9 - small dataset

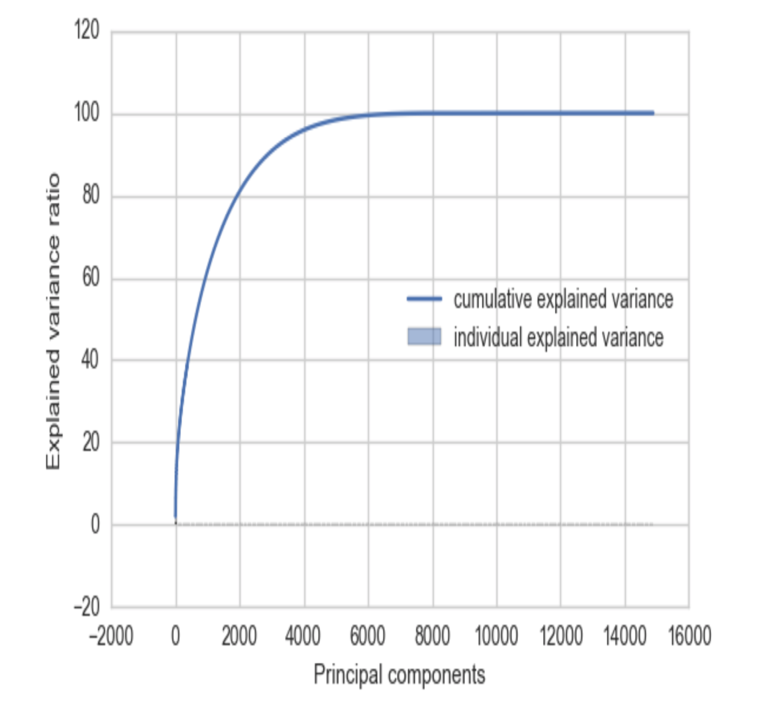
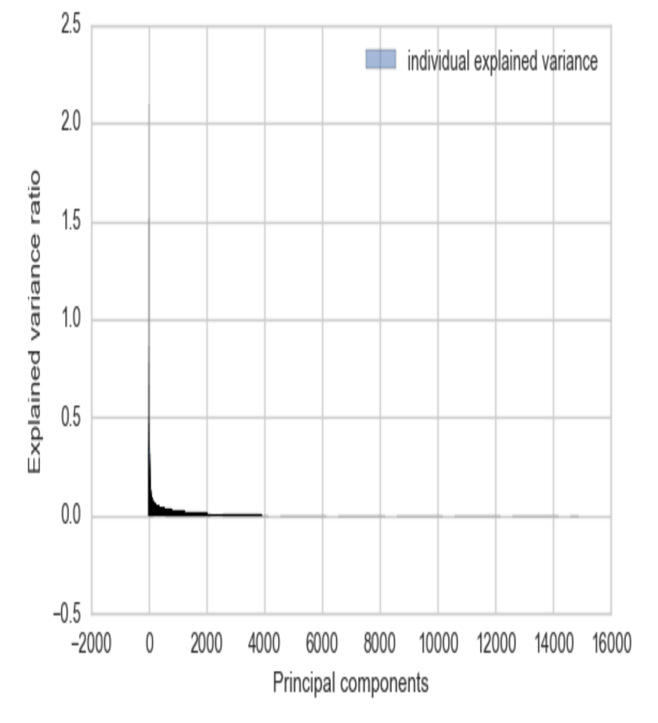
****

Figure 1-10 - large dataset  Figure 1-11- large dataset

For the small dataset, the plot 1-8 and 1-9 above clearly shows that most of the variance can be explained by the first 150 columns. For the large dataset, the plot 1-10 and 1-11 above clearly shows that most of the variance can be explained by the first 6000 columns (almost 100%).

After selecting the principal components, we can projection onto the new feature space. For the large dataset, we will take the 6000 columns and we will directly use **PCA library** to take out the columns. The process and final train data for the large dataset is shown as following:

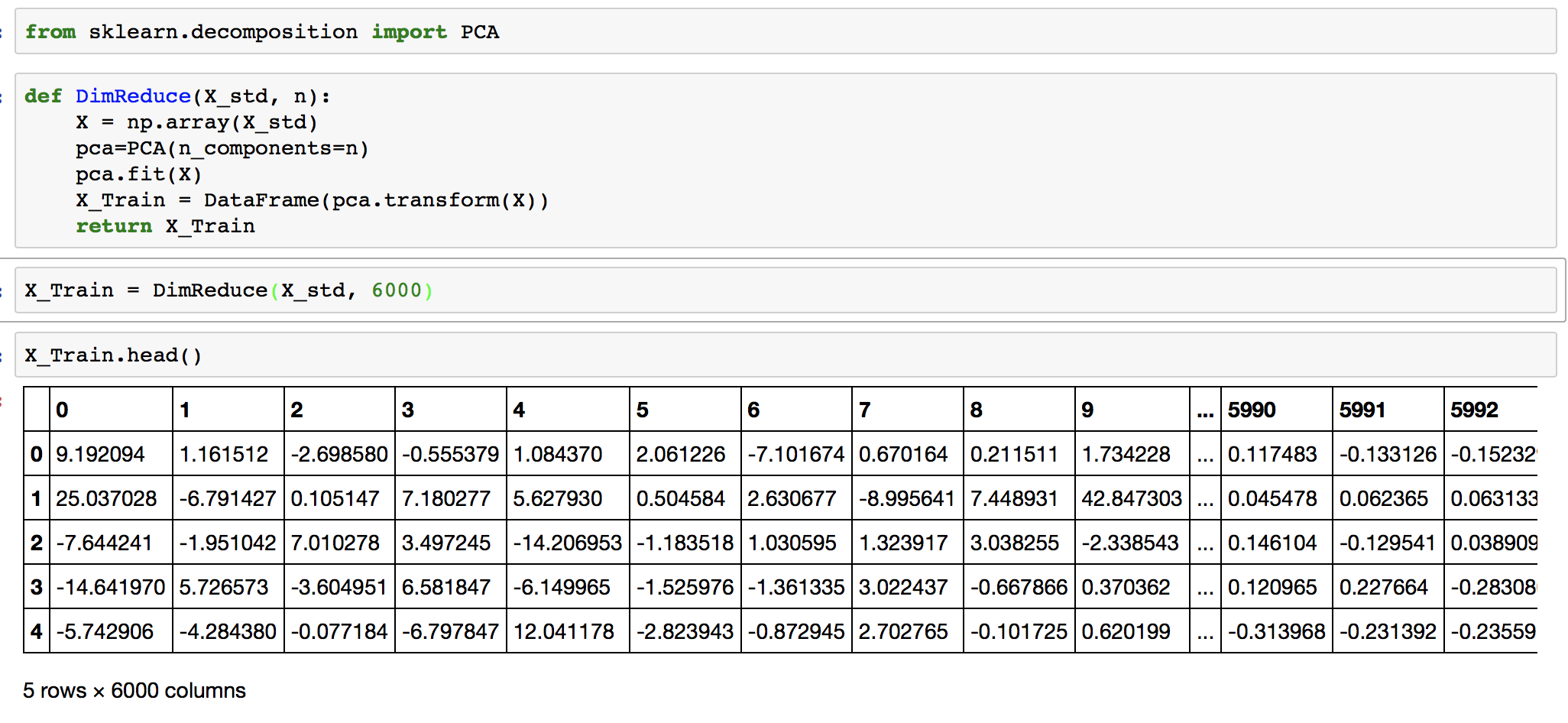


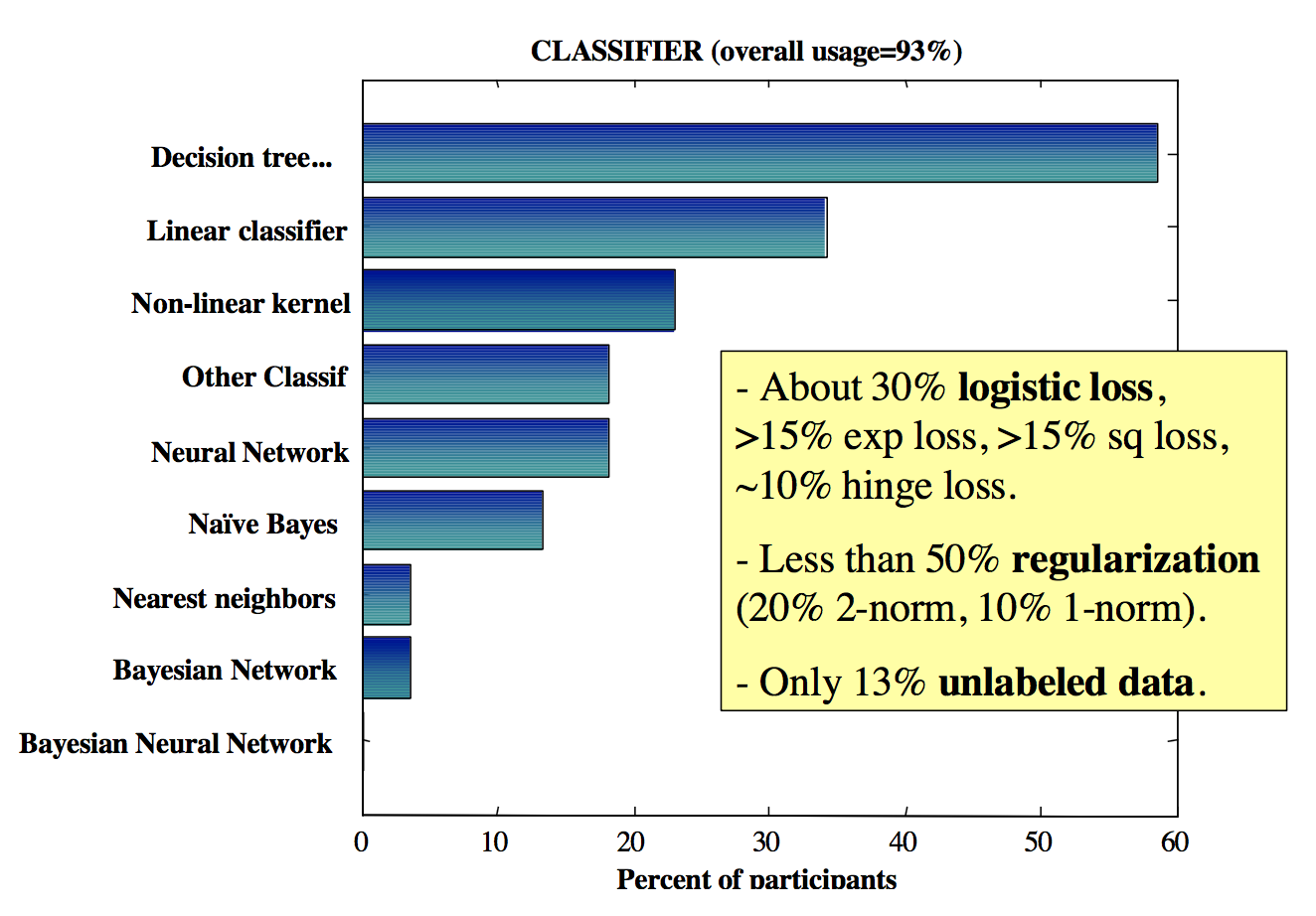
Figure 1-12- large dataset

\

**Machine Learning (Artificial Neural Network)**

**Analysis:**

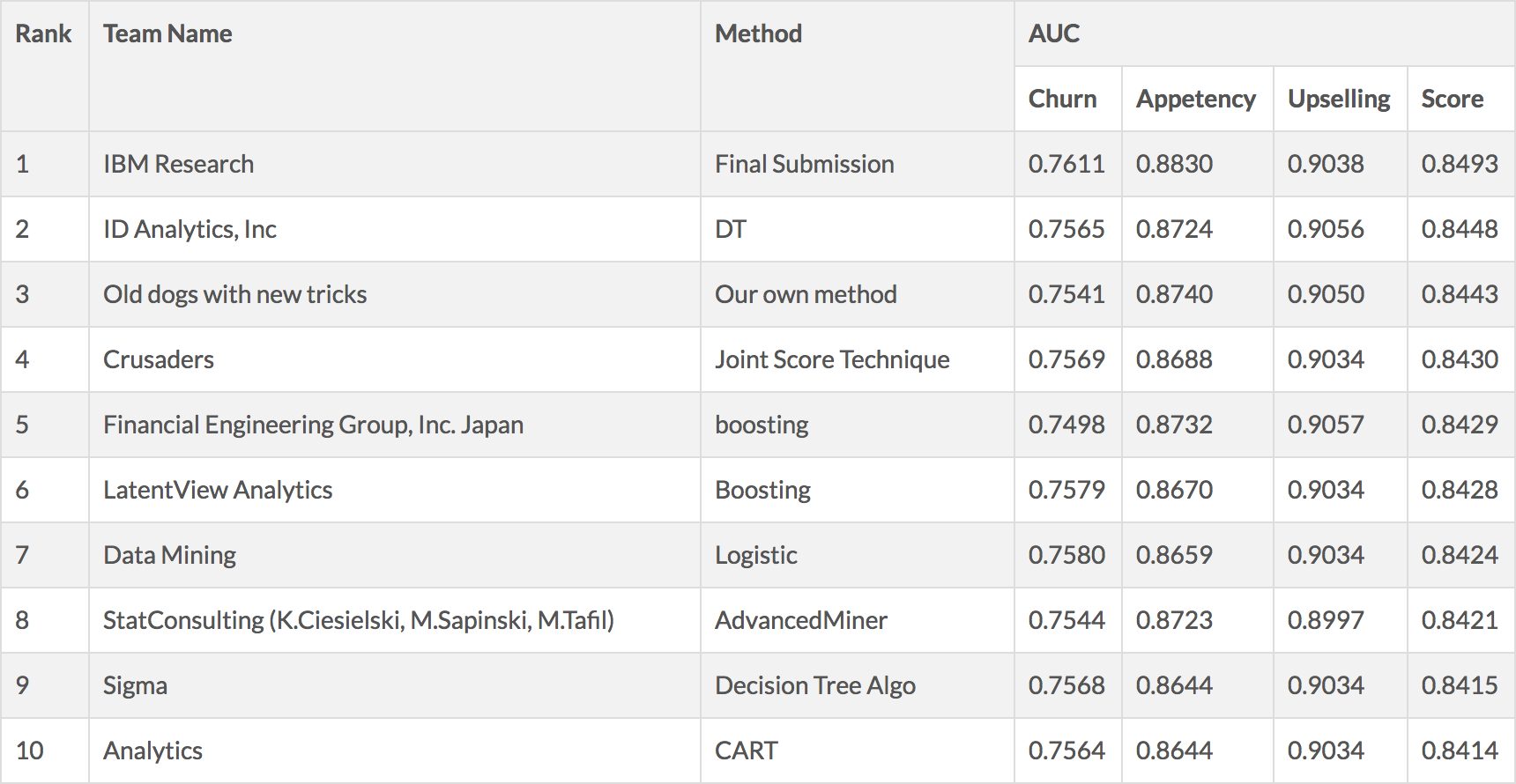
As the dataset has a large number of examples, mixed variable types, and lots of missing values, the most popular choice of classification algorithm among the original participants of the challenge is Ensembles of Decision Trees, followed by linear classifiers, more particularly logistic regression, as depicted in the following chart:



Choice of Classification Algorithms of original participants

Source: Analysis of the KDD Cup 2009: Fast Scoring on a Large Orange Customer Database

The best result of the customer churn rate prediction of the original competition was about 76% (top 10 depicted in the chart below), achieved by IBM with a mix of variety of classifiers following Caruana and Niculescu-Mizil’s research and their algorithms.

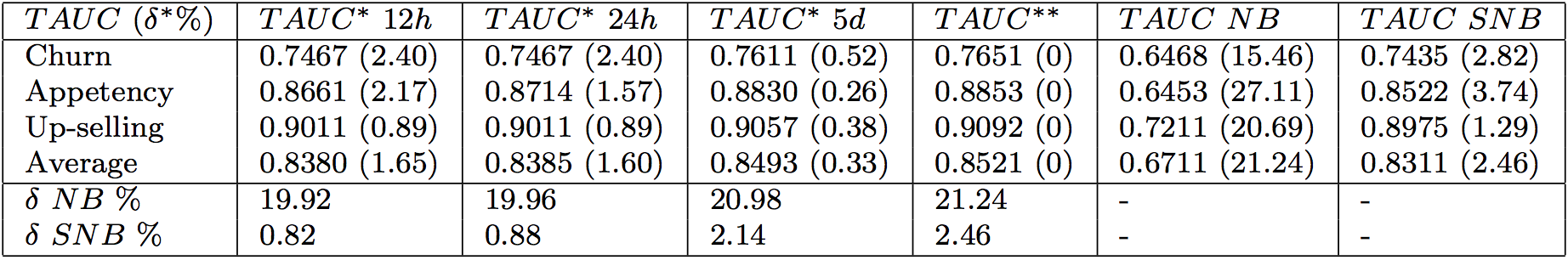


Top 10 results of original challenge

Source: <http://www.kdd.org/kdd-cup/view/kdd-cup-2009/Results>

Artificial Neural Network was used, but only by around 20% percent of the original participants, while none of whom made it to the top 10.

The base line Orange had provided for Churn is 0.7435, which is the result of its in-house Orange system implemented with an extension of the Naive Bayes classifier, called “Selective Naïve Bayes classifier” (Boulle, 2007), shown below (TAUC SNB):



Best results and baselines

Source: Analysis of the KDD Cup 2009: Fast Scoring on a Large Orange Customer Database

Which shows that even the top participant couldn’t beat the variation of Naïve Bayes by a significant margin. We think this shows general Ensembles Decision Trees algorithms’ incapability in modeling the classification boundary for this dataset.

Compared with neural networks, decision tree learning is more efficient with large numbers of labeled examples. Decision tree learning is less efficient with large numbers of features or high dimensionality, but still build or learn a model much quicker than the typical neural network. While for highly non-linear boundaries between classes, neural networks are more likely to find appropriate boundaries because decision trees will have to approximate a non-linear boundary with a series of axis parallel splits.

Another possible cause for Neural Network’s poor performance in the original challenge is hardware, according to KDD Cup 2009, While some teams used heavy computational apparatus, including multiple processors and lots of memory, the majority (including the winners of the slow track) used only laptops with less than 2 GB of memory, sometimes running in parallel several models on different machines.

ANN seems promising on improving customer churn prediction.

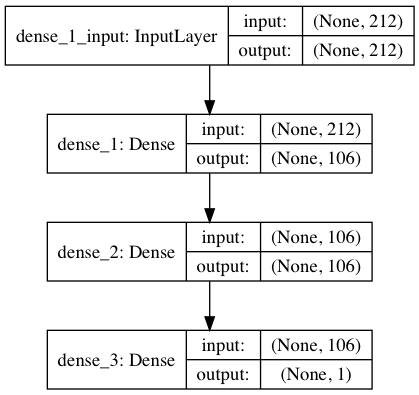
**ANN:**

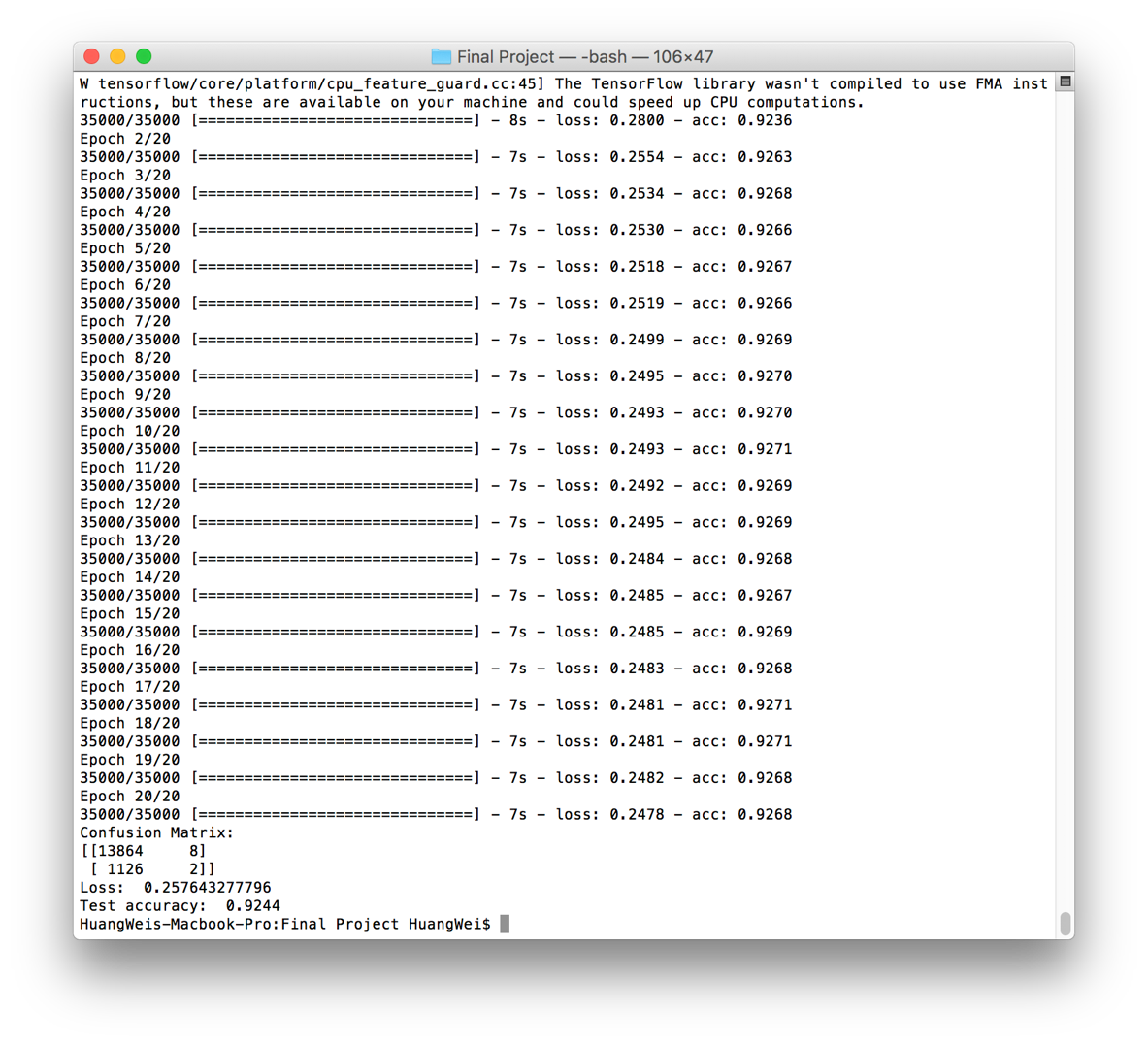
We successfully completed building ANNs on the small dataset. We achieved ANN on the large dataset with 10,000 rows.

For the small dataset, after data preprocessing we reduced the column count to 212 (delete columns that are all empty). For the large dataset, we reduced the column count to 6000 with PCA while retaining 98% of variance. Running results for different parameter combinations:

**1. Combination1 on small dataset (212 columns, 50,000 rows)**

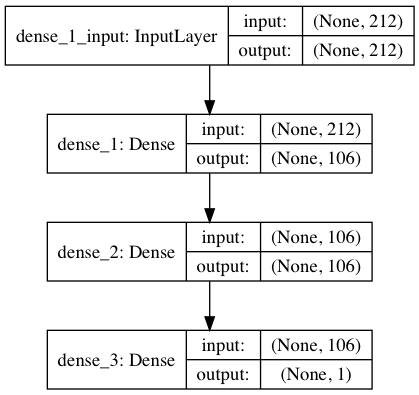
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No. of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 2 | 106 | linear | sigmoid | Adam | 10 | 20 |

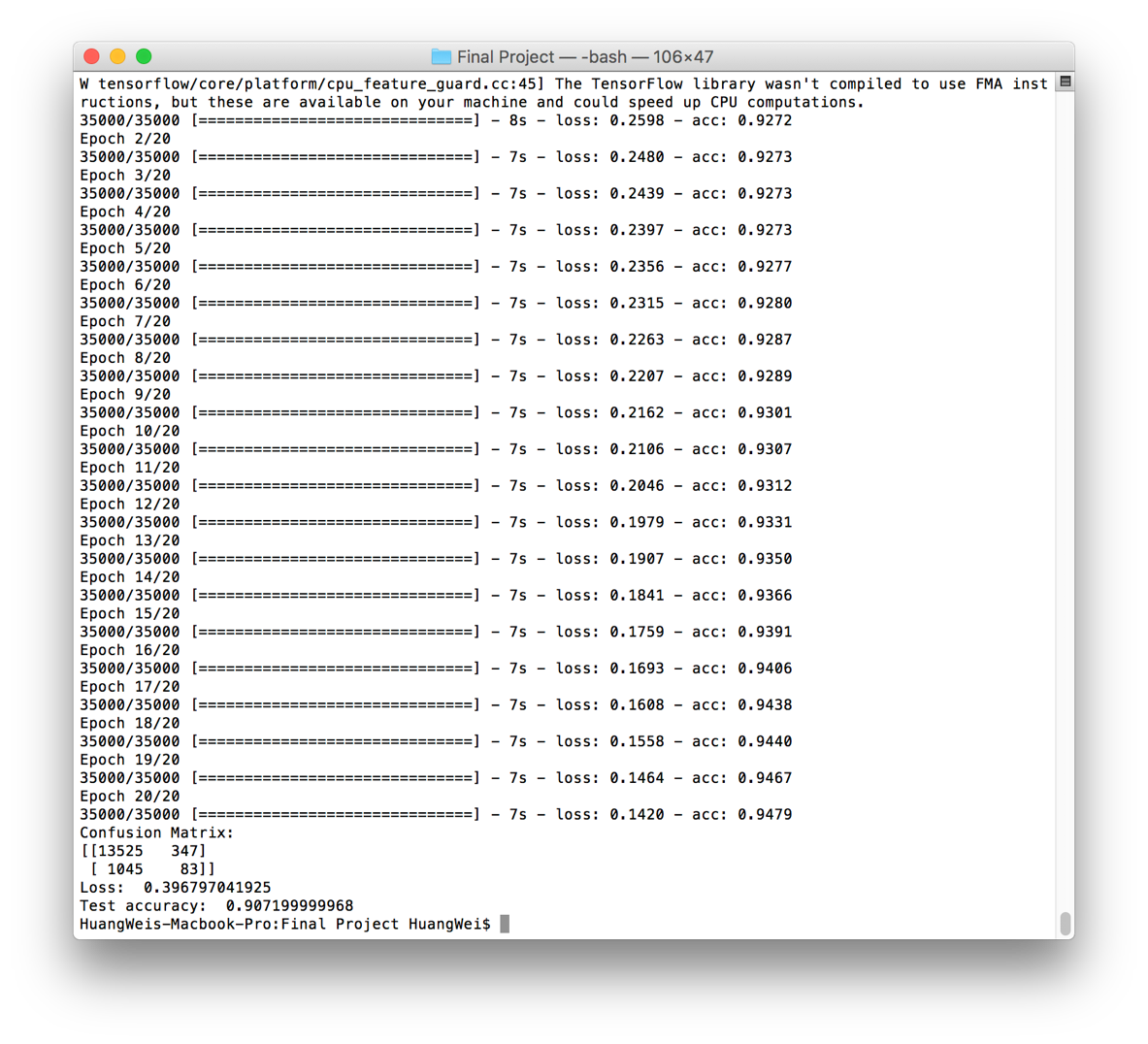




**2. Combination2 on small dataset (212 columns, 50,000 rows)**

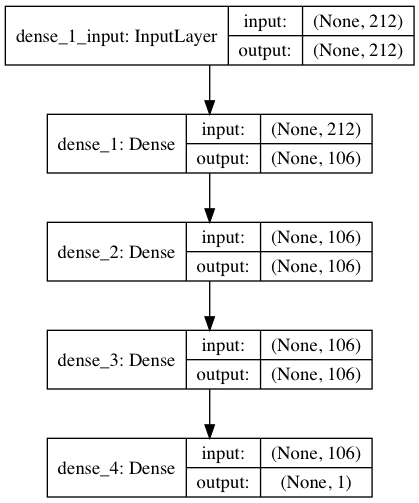
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No.of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 2 | 106 | relu | sigmoid | Adam | 10 | 20 |

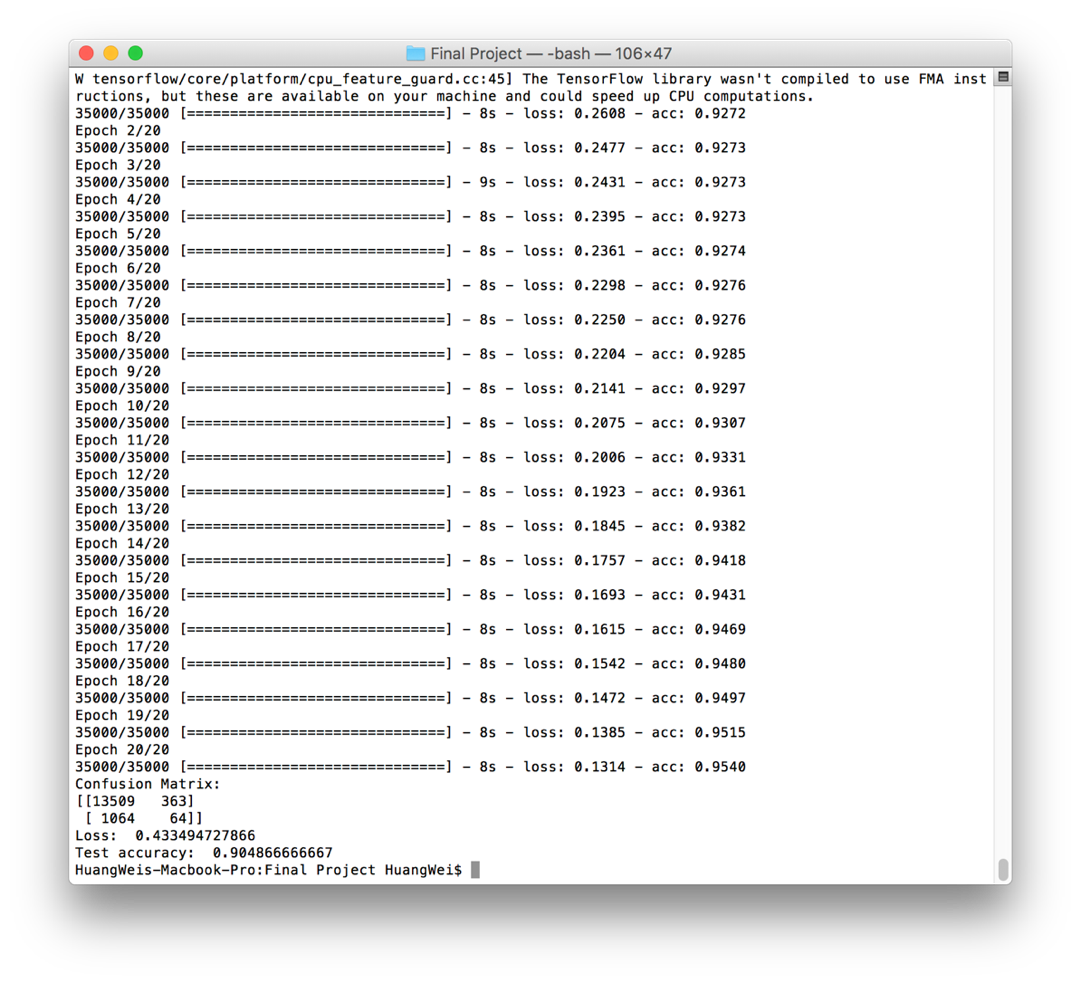




**3. Combination3 on small dataset (212 columns, 50,000 rows)**

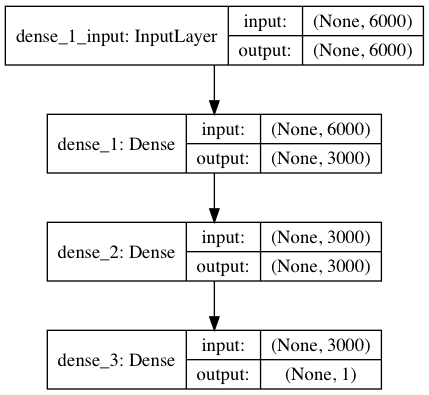
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No. of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 3 | 106 | relu | sigmoid | Adam | 10 | 20 |

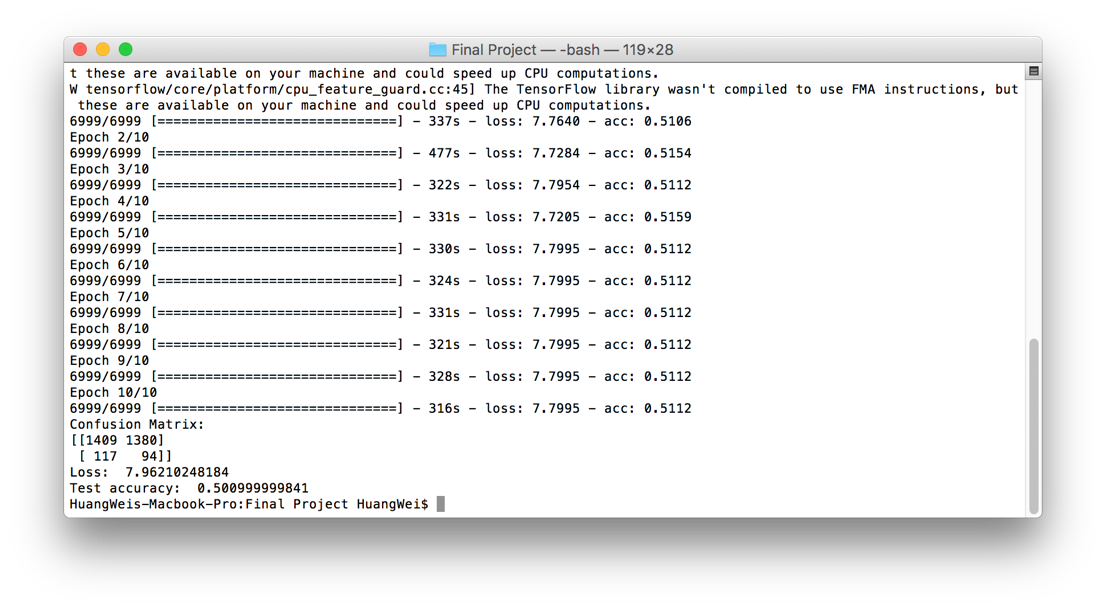




**4. Combination1 on large dataset (6,000 columns, 10,000 rows)**

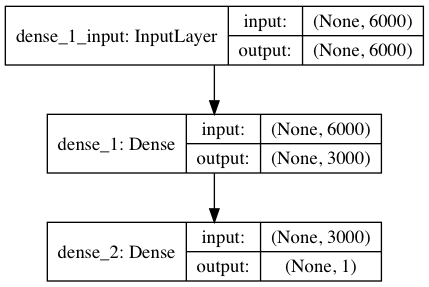
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No. of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 2 | 3000 | linear | sigmoid | Adam | 10 | 10 |

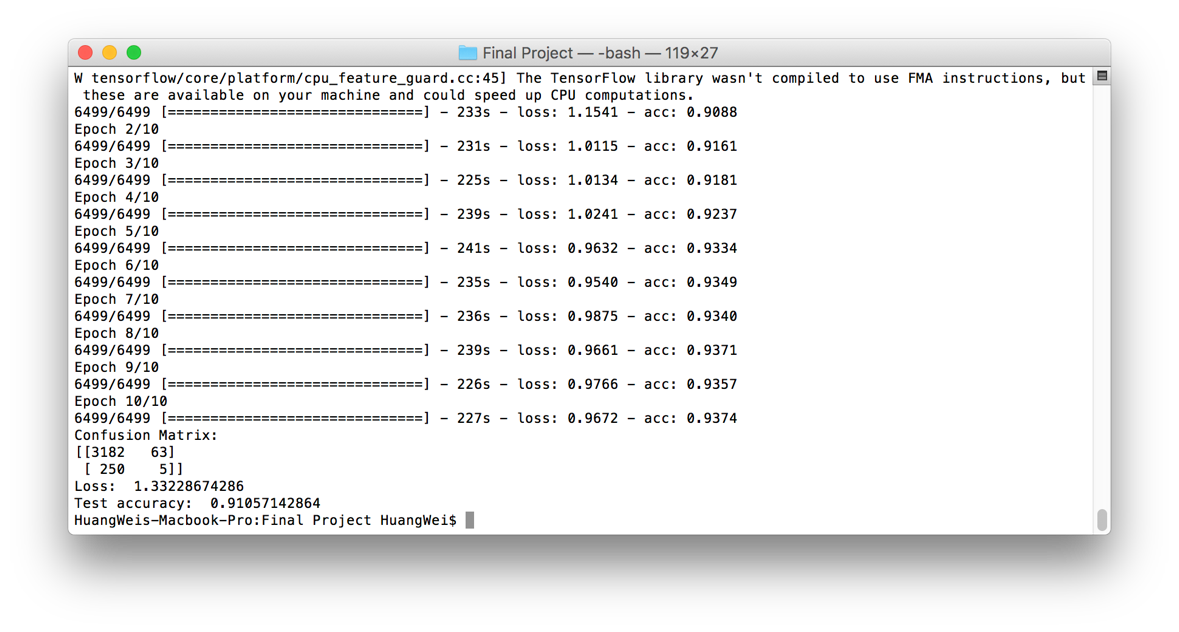




**5. Combination2 on large dataset (6,000 columns, 10,000 rows)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No. of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 1 | 3000 | relu | sigmoid | Adam | 10 | 10 |





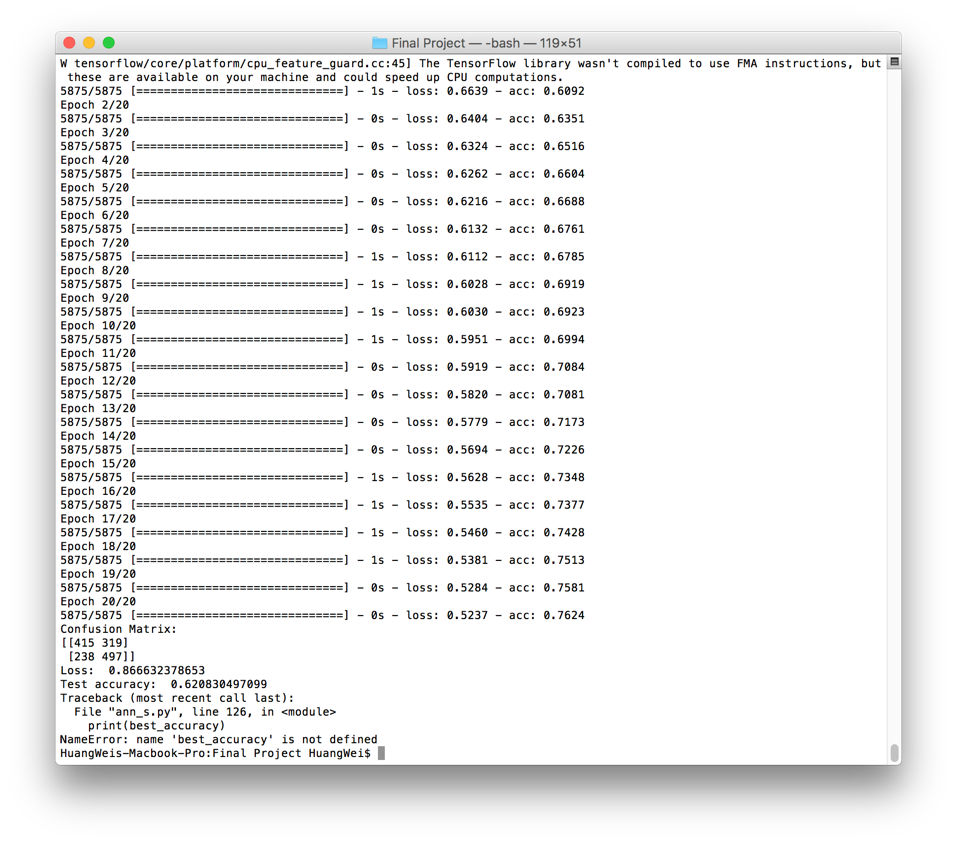
**ANN Parameter tuning**

As shown above, the ANN performed bad on predicting positive examples. The probable reason for this is that the scarcity of positive example in the dataset (which is also one of the challenges of this dataset). And the ANN on large dataset is over-fitted.

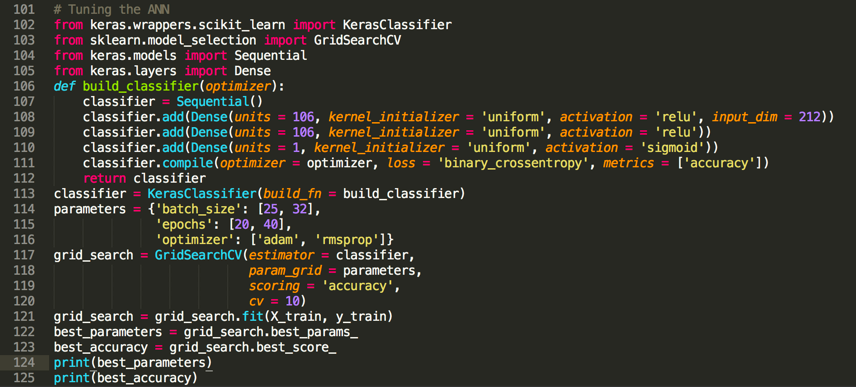
So, we rearranged the small dataset and generated a subset of 7,344 rows containing half positive and half negative examples. The following shows the running result of this subset.

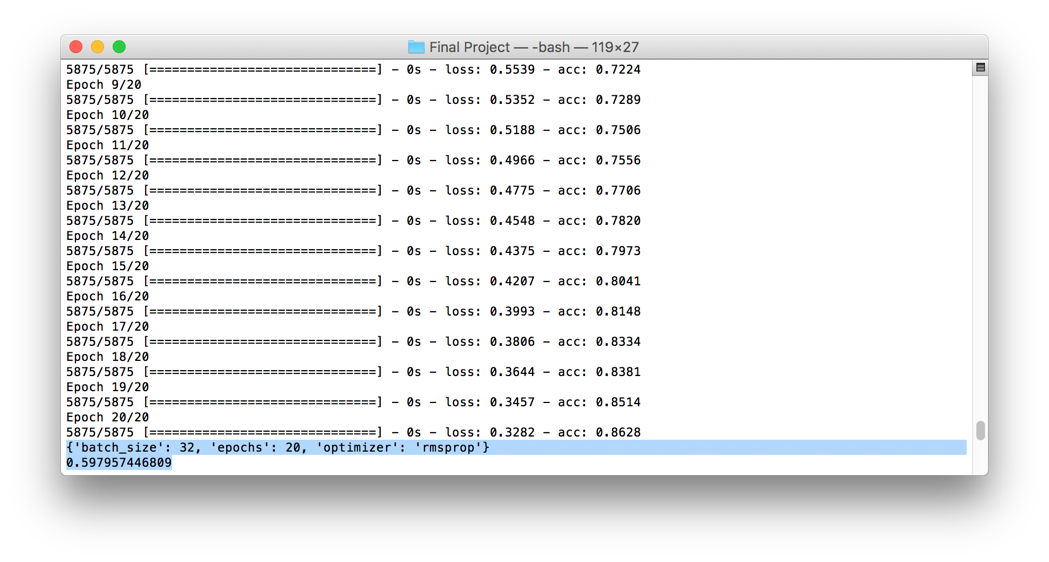
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test size | No. of hidden layer | No. of neurons in hidden layer | Activation function | Activation function of output layer | Optimizer | Batch size | Epochs |
| 30% | 2 | 106 | relu | sigmoid | Adam | 10 | 20 |

As shown, training accuracy is much higher than testing. It’s over-fitted.

****

We ran the following tuning code on small dataset:

****



But there’s not much improvement

**Conclusion and future work**

As a first step, we were successful on the small dataset. That is, from dataset preprocessing to ANN building and got a better prediction accuracy than the original KDD competition while the ability to predict positive examples is poor. We were successful in adopting big-data methodologies in data preprocessing even though the small dataset was successfully processed on our local machine.

While for the large dataset, the big-data methodology come into play. We were not able to completely process the dataset on our local machine (though we did finish the process for one chunk, which is 1/5 of the whole dataset). Due to the tight time frame of this project we were unable to finish our work on a real cluster. But from the result we got, our methodologies in both data processing and machine learning are working.

One point worth noticing is that the confusion matrix shows the accuracy for positive example prediction is generally bad for both the small and large dataset. Our theory is that it’s because that the positive example in the dataset is scarce thus it’s under fitted.

To complete and improve our work, future work would include:

* Run data preprocessing on a real cluster.
* Try to improve ANN training process with big data methodologies.
* Add model selection for ANN and fine tune its parameters.
* More in-depth analysis on positive example prediction accuracy.

**GitHub link of shared team repository**

<https://github.com/wadehuangwei/INFO7250-Final>

**References:**

Analysis of the KDD Cup 2009: Fast Scoring on a Large Orange Customer Database, *Proceedings of KDD-Cup 2009 Competition, PMLR 7:1-22, 2009.*

Naïve Bayes vs. Decision Trees vs. Neural Networks in the Classification of Training Web Pages, *IJCSI International Journal of Computer Science Issues, Vol. 4, No. 1, 2009*

Why are Neural Networks Sometimes Much More Accurate than Decision Trees: An Analysis on a Bio-Informatics Problem, *IEEE International Conference on Systems, Man & Cybernetics, Washington, D.C., pp. 2851-2856, October 5–8, 2003*

Websites:

<http://www.kdd.org/kdd-cup/view/kdd-cup-2009/Intro>

<http://www.312analytics.com/decision-trees-vs-neural-networks/>