2018

Insurance Risk Prediction

FINAL REPORT

CS 412: INRODUCTION TO DATA MINING

Kaggle ID :	**Xiruo Li**	
Group Members	NetID	Work Distribution
Egemen Okte	eokte2	1/3
Qingwen Zhou	qingwen2	1/3
Xiruo Li	xirupli2	1/3

Table of Contents

Executive Summary

For prudential life insurance project on Kaggle, after preprocessing steps, 3 different algorithms were implemented. Naïve Bayes, ADABoost and Linear Discriminant Analysis. From those 3, ADABoost used sklearn packages and therefore was not used for submission but it was only used for comparison. Kaggle scores of the three methods are 0.41, 0.49, 0.51 respectively the best result corresponding to LDA. Therefore LDA was used to classify the dataset.

Project Introduction

In real world, insurance companies are prudent in conducting risk analysis on applications before issuing the insurance. It is obvious that the applications with lower risk are more likely to successfully gain full coverage of insurance for goods or applicants. Thus, the insurance companies need to establish evaluation models on how to get the highest possible profits among all the applications based on their risk levels and at the same time, the clients can have ideas on how to build up the possibilities of being covered based on the evaluation criteria.

However, it becomes a tough process when many factors are taken into consideration. And not all of the factors can be represented as meaningful numeric numbers that can be interpreted in the evaluation model. For example, car brand, car model, car year, car accident history and so on, are required when applying car insurance, but it is difficult to represent car model by a meaningful value and different car brand has different car models, which makes it even harder to fill in this attribute. Besides, the real world dataset can have incomplete data, noisy data, inconsistent data and intentional data due to faulty instruments, human or computer error, transmission error, etc.

In this project, the main task is to learn to predict the risk level of existing clients based on 20000 training client dataset and test the prediction in 10000 client dataset. Each record is provided with 127 different attributes that can possibly contribute to risk level. This prediction is important for the insurance company to filtrate those applicants with lower risk level so that they can have higher promising profits. But the challenges occur that there are 127 attributes that can impact the risk level results, which is a lot. And many data are missing in the dataset which makes the dataset incomplete. What's more, this is a multi-classification problem, which means it is more complicated to build the classifiers than binary classification problem. And in the large training dataset, it is easy to observe the highly in-equal size of different classes, which will make it very hard to predict results. Without using existing code package, it will also take longer time to code the different algorithms to train and predict the dataset.

Analysis

1. To solve challenge mentioned above, for continuous attributes, mean values are used to fill the missing data and for the other attributes, modes values are used.
2. Since the 10000 testing dataset records does not have the risk level information, our group firstly randomly take 85% of 20000 training dataset as training set to gain the prediction model using several different algorithms: Linear Discriminate Analysis(LDA) and Naive Bayes (NB). And then the rest 15% of 20000 dataset are used as testing set to roughly evaluate the accuracy. It can briefly provide an intuitive evaluation on those algorithms.
3. Finally, our group apply these algorithms on all the 10000 testing dataset and get scores on Kaggle. The highest score 0.51 we get is using LDA (See LDA.py) and 0.41 is using NB (See trainingNB.py and testingNB.py).
4. Since ensemble method can help increase the accuracy of base classifiers, Logistic Regression (LR) as base classifier was coded but failed to do Adaboost since it has accuracy lower than 0.5 (See Adaboost_LR_failed.py). But if using the sklearn logistic regression package to do Adaboost, the code can run three rounds to get a score of 0.496 in Kaggle, but it cannot run more rounds since the accuracy can hardly exceed 0.5 from 4th round (See Adaboost_LR_sklearn.py)

Preprocessing

Missing Data

The data that we were presented have 8 classes of health insurance based on risk. There are 126 attributes with missing data. The methods that we have used for prediction are Naive Bayes (NB), AdaBoost, and Linear Discriminant Analysis (LDA). Each of these methods have different ways of dealing with missing data.

Naive Bayes does not need to deal with missing data in the training phase since the missing data does not affect the probability distribution of the selected attribute. However, during testing, if an entry is missing,

For AdaBoost and LDA, for training the missing data is replaced with the mean of the attributes from the training data if the attribute is continuous. If it is not, it is replaced with the most common entry. For testing, same is done with again the training data means

Please note that some entries are too sparse and may skew the data. But those are concerns of attribute selection and will be discussed later when the methods are implemented.

Product_Info_2

In the data, product info 2 seems to have very useful information, but it cannot be used as given. Therefore, the entries are converted into numbers. Ordering was made as A1=1, A2=1,...A8=8,B1=9,B2=10 etc. Replacing the values on product info 2 improved algorithm performance significantly.

Data Imbalance

There is a big imbalance in number of classes. Please see the histogram below.

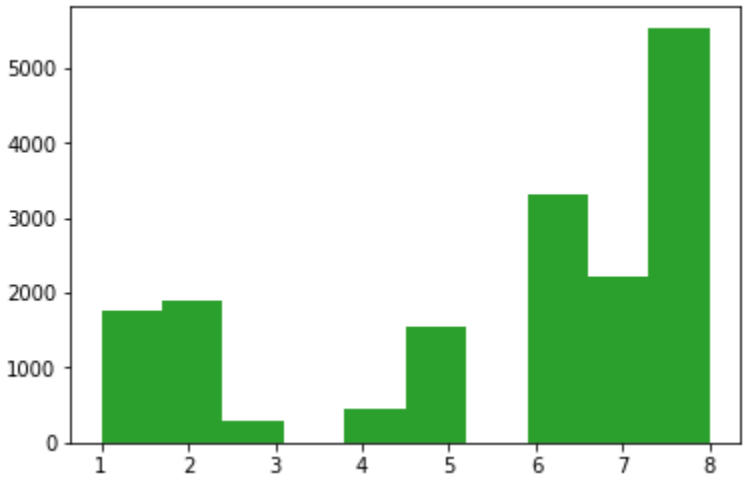

Figure 1: Class Imbalance

It is obvious that there is a very big imbalance between different classes. Class 8, highest risk have the highest rate of occurrence where rate 3 is lowest.

Many methods were tried to equalize the imbalance. First method was to decrease the amount of class 8 and sample 2000 or less from each class. this method resulted in very poor performance and was abandoned.

Second method was to again decrease the number of occurrences but then inside the prediction algorithm multiply the probability of the class with prior probability of the class so that the prior probability would be captured. However, this method also resulted in poor accuracy.

Best results were obtained by the raw data. As counter intuitive as that sounds, since probably the real data is also skewed, learning unbiased classifiers were not realistic. Therefore, data was used as given with no class imbalance correction.

Naive Bayes

Dealing with mix variable types

First method that was tried to create a baseline was Naive Bayes. Naive bayes assumes no correlation between the attributes and it works by calculation the probability of each attribute class and comparing it to response classes.

$$p(c/x) = \prod_{i \epsilon x} p(x/c) * p(c)$$

In this algorithm, calculating $p(c)$ is rather easy since it is just the prior probability of class. However $\prod_{i \epsilon x} p(x/c)$will differ based on the type of variable (John, Langley 1995)

If the variable is discrete or nominal, it is possible to just use the probability of occurrence by

$$p(x/c) = count(x = X, c = C)/count(c = C)$$

This will always result in a number between 0 and 1. However, if the variable is continuous, this approach will not work. In that case, the following assumption can be used

$$p(x/c) = g(x, mean, std) \ = \ normpdf(x, mean, std)$$

Please note that above equation is not strictly correct since the probability should be found with an integral of pdf. However, the pdf value is used as a proxy of the probability rather than the actual probability which is the adopted naive bayes described by John and Langley in 1995.

Selecting the attributes

Once the training methodology is understood, next step is to understand the attribute relations with the response since not all attributes will provide information about the response. Therefore, using all of the attributes may result in overfitting. Therefore, a method is needed to find out which attributes correlate the most with the response.
For this, Mutual Information was used. Mutual information, based on entropy, is a measure of how well one random variable tell us about another. We would like to find out how our attributes tell us about the response

Mutual information can be formulated as

$$I(X,Y) = H(X) - H(X/Y)$$

$$= -\sum_{x\epsilon X} p(x)logp(x) - (-\sum_{x\epsilon X}\sum_{y\epsilon Y} p(x,y)logp(x/y))$$

Please pay attention that if Y is replaced with X, Mutual information results in entropy of that variable. Again, it is important to distinguish between continuous and discrete variables. As explained before, discrete variable probabilities could be found by simply counting the number of occurrences for each class and plugging in to the equation above. However, for continuous variables, one would need to integrate within the problem space, which is between minus infinity to infinity in this problem to find the mutual information for each class. If we perform that operation for our attributes with respect to our response we get the following results

Table 1 Mutual Information for first 20 attributes

Attributes	**Mutual Information**
Product_Info_2	6.0094
BMI	0.7915
Wt	0.2711
Medical_History_15	0.2421
Product_Info_4	0.1877
Employment_Info_1	0.1845
Insurance_History_5	0.1707
Medical_History_2	0.1262
Employment_Info_6	0.1176
Medical_History_4	0.1068
Medical_History_23	0.1036
Employment_Info_4	0.0820
Ins_Age	0.0725
Medical_History_1	0.0555

Medical_History_24	0.0525
Family_Hist_5	0.0465
Family_Hist_3	0.0447
Family_Hist_2	0.0436
Family_Hist_4	0.0414

We can see that some variables provide very high information, whereas others provide little. Therefore using all attributes might result in overfitting. Using this list, the number of attributes were tried to see the optimum number

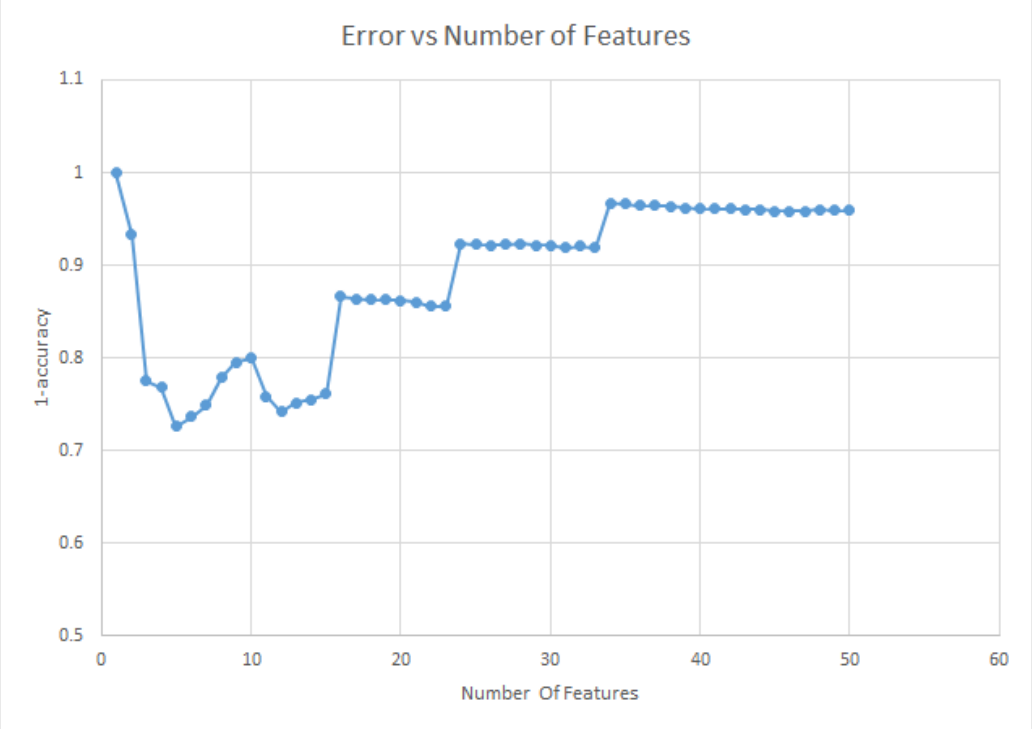

Figure 1 Model complexity vs test accuracy

From this graph, we see two things. One is that mutual information was not the best method for feature selection since there is a fluctuation of accuracy. We would expect a more smooth graph. We can actually achieve that graph if we train by choosing one feature at a time incrementally. But for a proof of concept, we see that as we increase the number of features, the error gets better up to a point, but then since we start overfitting, it starts increasing again.

After our tests, we found that for naive bayes, the best number of features were 12 which gave the highest score on Kaggle which was **0.41.**

Here is the confusion matrix for our training data after using 85% as test data.

Table 2 Confusion Matrix for Naive Bayes Training

		Predicted								TOTAL
Actual		1	2	3	4	5	6	7	8	36
	1	1	40	35	78	113	50	14	80	411
	2	4	41	37	48	169	60	15	66	440
	3	1	0	30	20	8	3	0	1	63
	4	2	1	10	72	2	3	2	3	95
	5	3	11	30	17	242	33	12	38	386
	6	10	20	49	170	211	148	19	146	773
	7	4	3	19	63	148	103	45	143	528
	8	13	2	7	121	95	102	19	947	1306
TOTAL		39	120	220	593	993	508	133	1432	4038

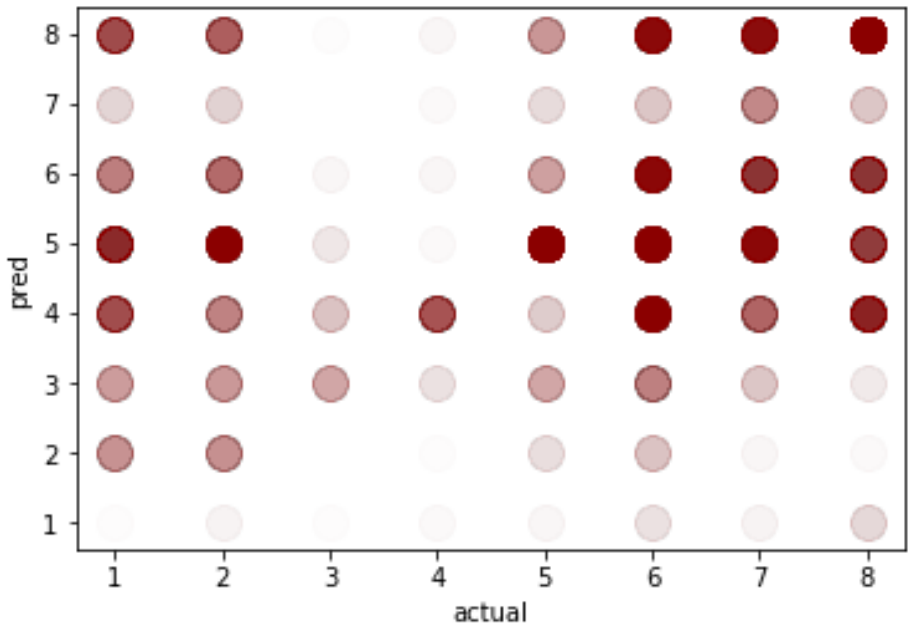

Figure 2 Visual Confusion Matrix

AdaBoost

Methodology

Boosting is an ensemble method to assign weight to each training tuple, mistakenly classified tuples' weight will increase so that learners can pay more attention to misclassified tuples in next training round. Adaboost (Adaptive Boosting) is a popular boosting method that as soon as a new classifier is built, the weights will be updated based on the accuracy of that classifier. This ensemble method tends to be more accurate than its base classifiers since it makes mistake when half of the base classifiers make mistakes.

Suppose a dataset D is with d class-labeled tuples, $(X_1, y_1), (X_2, y_2), \ldots, (X_d, y_d)$, where y_1 is the class label of tuple X_1. Initial Adaboost assigns each training tuple an equal weight of 1/d. Assuming there will be k classifiers ensemble together to predict the test set, thus, k rounds of classifiers are required. In round i, training dataset D_i is constructed by sampling with replacement from dataset D based on the weights of each tuple in D that induced from round i-1. Then build classifier M_i on $D_i$and calculate its error rate.

$$error\ (M_i) = \sum_{j=1}^{d} w_j \times err(X_j)$$

where $err(X_j)$is the misclassification error of tuple X_j: if the tuple was misclassified, then $err(X_j)$ is 1; otherwise, it is 0. If the performance of classifier M_i is so poor that its error rate exceeds 0.5, then abandon it and try again using new generated D_i in round i.

In each round, the $error\ (M_i)$ is used to update those tuples who are correctly classified by timing its weight by $error\ (M_i)$/(1-$error\ (M_i)$). As soon as each weight of tuples in round i is updated, they should be normalized so that their sum remains 1. After go through k rounds, there will be k classifiers and each classifier has an $error\ (M_i)$. The way to predict the testing dataset is to run each tuple through k classifiers, so each tuple will has k results. As we know that, each classifier (or each result) has an $error\ (M_i)$, then $log(error\ (M_i)$/(1-$error\ (M_i)$)) is used to assign the weight of each result for a tuple,. So for each tuple, group by the same result by summing their weights and select the result with the highest total weight as the tuple's class.

Implementation

In this project, we use multiclass logistic regression as the base classifier method and then do boosting. In our trial, we build multiclass logistic regression using one-vs-all algorithms. It is because the basic logistic regression method is used to solve binary classification, but when it comes to multinomial classification, we can do binary for each classifier. So for example, there are 8 classes in our project, we need 8 classifier to differentiate class i or non class i results and it become the binary classification problems again. However, using our own code, this method has very low accuracy when training the

whole 20000 tuples with 126 attributes, the accuracy of each round can hardly exceed 0.5, which limit the code keeping training in the first round and it cannot update the weight of each tuple, thus, our manually written logistic regression code failed to do Adaboost (See Adaboost_LR_failed.py).

To check the possible reasons, we also use the sklearn Logistic Regression package to learn if logistic regression can be used in Adaboost (See Adaboost_LR_sklearn.py). It shows that the adaboost can run at least 3 rounds on multinomial logistic regression and get a prediction of 0.496 on Kaggle. But the same problems happen that it cannot go through the 4th round since the accuracy rate is almost lower than 0.5. So the problem is probably due to the unsuitable use of logistic regression in this project since the training data has a lot attributes, and some of them may has very low correlation with the target class. Besides, the data has very high inequality in the data size of each class. All these factors can make it has high error when using logistic regression method.

Linear Discriminant Analysis

Methodology

From Bayes Rules, the condition probability can be formulated as:

$$\mathrm{p(Y = k|X = x)} = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Where $f_k(x)$ is the conditional density function of X|Y=k, and $\pi_k = P(Y = k$ is the prior probability.

The best prediction is picking the one that maximizing the posterior:

$$\arg\max_k \ \pi_k f_k(x)$$

LDA model $f_k(x)$ as a normal distribution. Suppose we model each class density as multivariate Gaussian $N(u_k, \sum_k)$, and assume that the covariance matrices are the same across all k, i.e., $\sum_k = \sum$

Then, the probability function for class k is:

$$f_k(x) = \frac{1}{(2\pi)^{\frac{p}{2}} \sum^{\frac{1}{2}}} \exp[-\frac{1}{2}(x - u_k)^T \sum^{-1} (x - u_k)]$$

The log-likelihood function for the conditional distribution is:

$$logf_k(x) = -\log((2\pi)^{\frac{p}{2}}\sum^{\frac{1}{2}}) - \frac{1}{2}(x-u_k)^T\sum^{-1}(x-u_k)]$$

$$= -\tfrac{1}{2}(x-u_k)^T\sum^{-1}(x-u_k) + constant$$

Hence, we just need to select the category that attains the highest posterior density:

$$Y_{pred} = \arg\max_k log(\pi_k f_k(x))$$

$$= \arg\max_k -\tfrac{1}{2}(x-u_k)^T\sum^{-1}(x-u_k) + log(\pi_k)$$

Noticing that quadratic term can be simplified to:

$$-\frac{1}{2}(x-u_k)^T\sum^{-1}(x-u_k)$$

$$= \mathrm{x}^T\sum^{-1}u_k - \tfrac{1}{2}{u_k}^T\sum^{-1}u_k + irrelevant\ things$$

Then the discriminant function is defined as:

$$\delta_k(x) = \mathrm{x}^T\sum^{-1}u_k - \frac{1}{2}{u_k}^T\sum^{-1}u_k + log(\pi_k)$$

$$= \mathrm{w}_k^T x + b_k$$

We can just calculate w_k and b_k for each class k.

Feature Selection

An important factor as with many other machine learning models is feature selection. It is possible to over fit the model and get poor performance with increasing number of features. On the contrary too little features may result in under fit.

To determine the optimum number of features, the following methodology was implemented:

1) Select the best attribute that gives the best testing accuracy by trying each attribute one by one
2) Remove that attribute from the search list and add it to model
3) Search the remaining attributes to add to the existing model.
4) Repeat Step 2 until accuracy decreases.

Implementing this strategy, we can plot the following graph

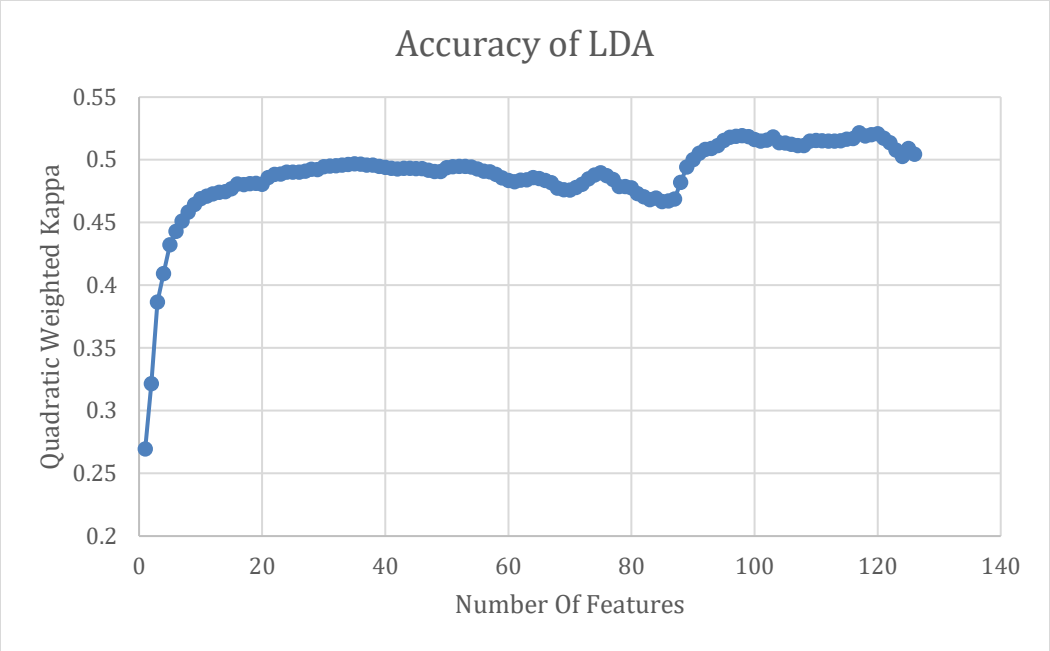

Figure 4 Number of Features vs Performance in LDA

As the plot shows, testing accuracy keeps increasing with increasing number of features which is counter intuitive. This may show that this method of feature selection may not be the best and the features are not independent of each other. However, due to time restrictions, this methodology was adopted. This graph illustrates that the best testing accuracy is between 116 and 126 features. During testing, best Kaggle score was obtained by using all features.

Accuracy

Preprocessing is required as it mentioned before. In LDA.py, all the 126 attributes are taken into consideration and it only cost 0.16 seconds. All the prediction to testing dataset is saved as variable “pred_kaggle” and it got 0.51 score on Kaggle.

Here is accuracy plot (by splitting training dataset into training and testing):

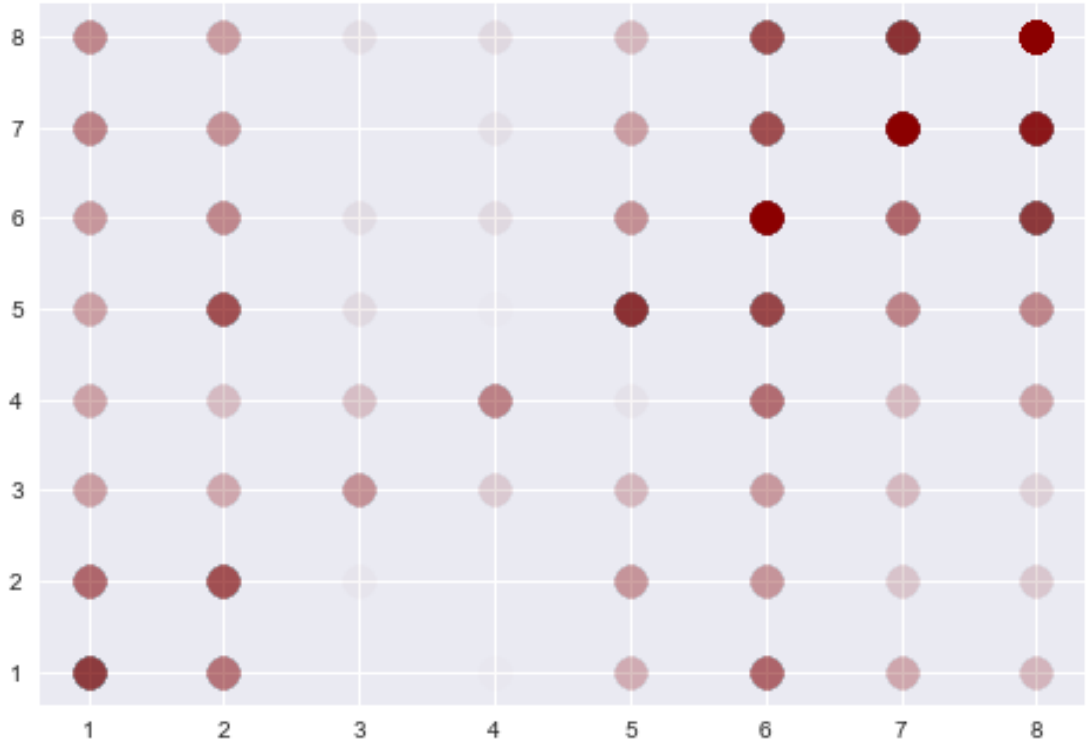

Figure 5 Visual Confusion Matrix for LDA

Conclusion and Discussion

For this project, three different algorithms were implemented. LDA, Naive Bayes and AdaBoost with logistic regression. Their respective Kaggle scores were 0.41 and 0.51 for Naive Bayes and LDA, respectively. Since our AdaBoost used sklearn logistic regression packages, the results will not be used even though it gets 0.49 score in Kaggle. It was just used to compare our results to each other.

Looking at the data, it is clear that the data is skewed and there is no clear distinction between risk classes and attributes that is straightforward. Therefore, the accuracies are rather low, lower than what we obtained with packages for the preliminary result. However, given that the implemented codes are more simplified than the sklearn packages, a score of 0.51 by LDA is pretty good given that random chance could only predict the right class with an accuracy of 0.142.

Naive Bayes approach was used to create a baseline. Despite its assumptions and simplicity, it is still a very powerful tool to get baseline result. Later, LDA method was used to improve on Naive Bayes and the accuracy were increased significantly, about 10%. LDA uses a multivariate Gaussian distribution for

the class conditional density and Naive Bayes treats the conditional density as a product of univariate Gaussian distributions. It seems that for this dataset, multivariate gaussian distribution was a better fit than independent univariate variables.

Adaboost method seems to not work well when the base classifier is logistic regression. It is probably because that the Adaboost can help increase the accuracy especially the base classifier is a weak learner. But when the base classifier has relatively good learner behavior, Adaboost cannot improve the results a lot.

LDA gava the best score with 0.51 on Kaggle. Even though some methods were implemented for feature selection, at the end, best score was obtained by using all of the attributes. Therefore LDA was used to classify the dataset.

Reference

Bentley, J.L. (1975). “Multidimensional binary search trees used for associative searching”, Communications of the ACM.

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, *48*(1-3), 253-285.

G. John, and P. Langley, (1995). “Estimating Continuous Distributions in Bayesian Classifiers”.

Hsiang-Fu Yu, Fang-Lan Huang, Chih-Jen Lin (2011). Dual coordinate descent methods for logistic regression and maximum entropy models. Machine Learning 85(1-2):41-75.

H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

Kingma, Diederik, and Jimmy Ba (2014). “Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980.

L.Breiman (2001). “Random Forests”, Machine Learning, 45(1), 5-32.

Y. Freund, and R. Schapire, (1997). “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”.