GROUP 3: FINAL REPORT

PROJECT 1: FRAUD DETECTION OF INSURANCE CLAIMS

Oct 28 , 2018

LI Weihao 1155077142
LI Jinzhao 1155077016
Wang Yiqun 1155062115
Peng Zhichao 1155062015

Contents

1 Data pre-processing

In this project, we employed several statistical methods to solve the fraud case detection problem. The data were cleaned before the models were fitted. Since the data mainly consisted of time variables, categorical variables and interval variables, certain modifications had to be made to the data so that our statistical models could be applied.

1.1 Time gap

The dataset has the following time variables: **Year, Month, WeekOfMonth, DayOfWeek, DayOfWeek-Claimed, MonthClaimed, WeekOfMonthClaimed.**Out of common sense, the time when the accident happens and the claim is made is not directly related to the fraud. Hence we would like to calculate the time gap between the claim and the accident to make better use of them.

Assumption 1 Considering we don't know exactly in which year the claim was made, we assume that the time gap is smaller than one year, because in real life it is quite rare that the claim is made such a long time after the accident. We noticed a few records whose claim was recorded earlier than accident. These records generated abnormal time gap values far from values of the rest of data, which had a huge influence on the result of modeling. Considering the sufficiency of data in this dataset, we decided to remove all 1655 abnormal records, 10.7% of the dataset.

Assumption 2 Assume the number of week of the month starts from the first day of the very month, instead of the first line of the calendar. We would like to explain with an example. January 1st in 1994 is a Saturday, then the first week of January of 1994 starts from January 1st and ends on January 7th, instead of consisting of only one day, January 1st, which is the only day on the first line of calendar of January, 1994. Similarly, the second week starts from the January 8th to January 14th, instead of starting from January 2nd to January 8th, the second line of the calendar in January, 1994.

Method

$$Date(MonthClaimed, DayOfWeekClaimed, DayOfWeek)$$
$$-Date(Year, Month, WeekOfMonth, DayOfWeek)$$

1.2 Mapping to numeric values

Some variables are interval variables, and this will cause ambiguity and inconvenience as we expect more numeric variables in most of the statistical models. We want to convert the interval variables to some numeric values so that they can be fitted into models better.

Assumption Assume that for a certain interval variable, the values are similar if falling into the same interval. E.g. for **NumberOfCars** indicating the number of cars owned by the policy holder, "3 cars"and "4 cars" are similar as they fall into "3 to 4" interval.

Method $Assign\ the\ midpoint\ of\ the\ interval\ as\ the\ corresponding\ value\ of\ this\ interval.$

1.3 Change to dummy variable

We have 11 categorical variables, accounting %33.3 of the entire 33 variables. For each categorical variable, we created some dummy variables to replace them. For the binary variables, only one dummy variable is used. For categorical variables with c levels, $(c-1)$ dummy variables are used to replace the original one categorical variable

1.4 Create new features using PCA

We want to improve the performance of the classifier and reduce the dimensionality of the data by introducing PCA analysis to the data. Suppose we have n records with p variables, and we plan to reduce the number of variables from p to q. PCA achieves this task by constructing principal components $a_1^t x, a_2^t x, ..., a_q^t x$. The first component tries to contain maximum variance for the data, and for the next (q-1) components, the next one should be uncorrelated with the previous one and explain the maximum variance for the data successively.[1]

[1]Jolliffe I. (2011) Principal Component Analysis. In: Lovric M. (eds) International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg

2 Selection Criteria

- Precision is equal to the proportion of correctly raised alarms, as follows:

$$Pr = \frac{TP}{TP+FP}$$

- Recall is equal to the proportion of deviant signatures, which are correctly identified as such:

$$Re = \frac{TP}{TP+FN}$$

		Classified as	
Actual		Fraud	No fraud
	Fraud	TP-true positive	FN-false negative
	No fraud	FP-false positive	TN-true negative

- *F-measure* is a measure that calculates a harmonic mean between precision and recall, as follows:

$$F\text{-}measure = \frac{2 * Pr * Re}{Pr + Re}$$

- Use Recall and *F-measure* as final criterion and not use accuracy rate

Precision is the proportion of real fraud cases in cases classified as fraud.
Recall is the proportion that fraud cases are classified correctly.
F_score conveys the balance between these two rates.

Explanation: Usually, there are two steps in fraud claims detection, first people try to pick out all potential fraud cases, second, they further classify each selected case artificially. Recall measures the performance of step one that how many real fraud cases are selected while precision measures the workload of step two. Since we want both high recall and high precision, F_score is used to evaluate their integrative performance. Because our focus is on the first step, we eventually pick Recall and F-score as our selection criterion. The reason why accuracy rate is not adopted is that the imbalanced dataset is dominated by not fraud part so that Given low fraud detection rate, the accuracy rate is still high. One obvious extreme cases is that even there is no fraud detection, it gives an accuracy rate of 94 percent.

3 Method

3.1 Processing Unbalanced Data

To deal with unbalanced data, there are several commonly used methods, namely random under-sampling, random over-sampling and Synthetic Minority Over-Sampling Technique. The following section will illustrate these three methods. It's worth mentioning that the proportion of majority class and minority class after random under-sampling are both 50%.

3.1.1 Random Under-sampling

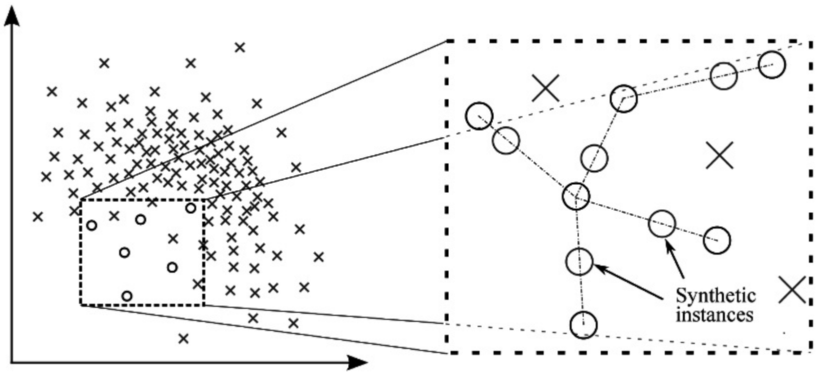
Random under-sampling will randomly sample data from majority class to match with the number of the minority class in order to make the data balanced. The training sample obtained in this way will be relatively small. For huge training data set, this method will help improve the run time and storage problem. On the other hand, useful information may lost because of under-sampling and small training may turn out to be a biased sample.

3.1.2 Random Over-sampling

Opposite to random under-sampling, random over-sampling will randomly over sample the data from minority class to deal with the unbalanced problem. In this way, all information from the majority class will be preserved and there is no information loss in this case. However, the replication of minority class may increase the risk of over fitting.

3.1.3 Synthetic Minority Over-Sampling Technique (SMOTE)

SMOTE is one of the modified over-sampling method. In stead of directly copying the minority class a couples of times, SMOTE generate the minority class by performing linear interpolation of between some k-nearest neighbour. The following plot will illustrate how SMOTE work.



3.2 Logistic Regression & K-Nearest Neighbors

For logistics regression, we noticed that there exists problem when applied the trained model to the testing set. There are nine levels in 'PolicyType', but for level 'Sport – Liability',only one person In the whole dataset, if this individual is not in the training set but in the testing data, error will come up since no coefficient for that level exist. It is also a motivation for us to change the categories to dummy and numerical.

First, we use the training set without PCA to train the logistics model. When put all independent variables into the logistics regression, some variables' coefficients are NA reported by R, we realized these variables may be collinear to other variables. We delete them from independent variables of training set and the testing set, and rerun the logistics regression.

Secondly, we apply the PCA on the training set and change the variables in the testing set according to same PCA linear combination. This time there is no NA in the model coefficients. In each case, we selected the 0.5 as the threshold to determine whether the individual is fraud or not, if the probability predicted is bigger than 0.5, we classified it as fraud case.

When comes to the KNN, KNN is a simple method which does not assume the data distribution, it contains all data point from the training set and use them in the testing procedure. By voting from the neighbor training point, the class of testing point can be determined. Due to this property, imbalanced dataset will make individual be classified as the majority class more easily when K is large. So we think that if the testing data is imbalanced, then the result of this method is undesirable.[3]

The selection of the K is based on our experiment, we run the KNN with k=1 to k=20 and select the one with best F score.

3.3 Support Vector Machines

Support Vector Machines (SVM) has been a successful classifier in the past fewer years and performed well in our project. We would like to briefly introduce this method.

3.3.1 Intuition of the methodology

To kick off, consider a set of linear separable data in 2-dimension, and there is more than one line to separate these data. Instead of finding just one separating line, we want to select the best among all separating lines possible. A notion of *margin of a separating line* is introduced as following: the *margin* is the distance between the line and the nearest data point. We claim that the line with larger margin separates the data better, intuitively meaning that the best line passes the midpoint of the area between two data groups. An informal justification is that the chance that new data point falls in the wrong side of the separating line decreases as the line moves away from the center of data group.

3.3.2 Maximize margins

In the light of the intuition that fatter margin gives better separating lines, we try to maximize the margin. Say in the d-dimensional space, we have N data points and plane $\boldsymbol{w^t x} = 0$. Note w multiplying two different constants leads to $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ respectively, but $\boldsymbol{w}$, $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ indicate the same plane. So we normalize $\boldsymbol{w}$ by setting $|\boldsymbol{w^t x_n}| = 1$, where $\boldsymbol{x_n}$ is the nearest point to the plane. We also pull out w_0, one coefficient in $\boldsymbol{w}$ corresponding to the constant element in $\boldsymbol{x}$, and the plane is expressed as $\boldsymbol{w^t x} + b = 0$.

Like the margin of a line, the *margin of a plane* is defined as the distance between the nearest point and the plane. We set $\boldsymbol{x_n}$ is the nearest point and choose any point $\boldsymbol{x}$ on the plane $\boldsymbol{w^t x} + b = 0$. Note $\boldsymbol{w}$ is the normal vector with direction vector $\boldsymbol{\hat{w}}$. The margin of this plane is $||\boldsymbol{\hat{w}^t (x_n - x)}|| = ||\boldsymbol{w^t x_n} - \boldsymbol{w^t x}||/||\boldsymbol{w}|| = 1/||\boldsymbol{w}||$. Now, we have

$$Maximize\ \frac{1}{||\boldsymbol{w}||}, \quad subject\ to \min_{n=1,2,\dots,N} |\boldsymbol{w^t x_n} + b| = 1$$

Notice that $y_n = 1$ or -1 and $y_n = sign(\boldsymbol{w^t x_n} + b)$, and thus $|\boldsymbol{w^t x_n} + b| = y_n(\boldsymbol{w^t x_n} + b)$. So the problem above is equivalent to

$$Minimize\ \frac{1}{2}\boldsymbol{w^t w}, \quad subject\ to\ y_n(\boldsymbol{w^t x_n} + b) \geq 1$$

Then we have Lagrange formulation

$$Minimize\ \mathscr{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w^t w} - \sum_{i=1}^{N} \alpha_n (y_n(\boldsymbol{w^t x_n} + b) - 1), \quad where\ each\ \alpha_n \geq 0$$

Note that

$$\nabla_w \mathscr{L} = \boldsymbol{w} - \sum_{i=1}^{N} \alpha_n y_n \boldsymbol{x_n} = \boldsymbol{0} \qquad \frac{\partial \mathscr{L}}{\partial b} = -\sum_{i=1}^{N} \alpha_n y_n = 0$$

We have

$$\mathscr{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \boldsymbol{x_n^t x_m}, \quad where\ each\ \alpha_n \geq 0\ and\ \sum_{n=1}^{N} \alpha_n y_n = 0$$

This will be passed to quadratic programming and the solution $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)$ will maximize $\mathscr{L}$. Since quadratic programming is a well-developed numeric method, it will not be discussed here. Then we can solve $\boldsymbol{w}$, $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_n y_n \boldsymbol{x_n}$.

Next we solve b. Note for $n = 1, 2, \dots, N, \alpha_n(y_n(\boldsymbol{w^t x_n} + b) - 1) = 0$. We would like to reason this claim out of intuition. Remember we are defining the best separating plane, i.e. the line with largest margin. Notice that margin only concerns with the points nearest to the very plane, so those points far away contribute absolutely nothing to defining our plane and that's why the coefficient of the plane concerns none of these points far away from this plane. In other words, if a data point $\boldsymbol{x_n}$ has a positive α, it contributes to defining the best plane. We name these $\boldsymbol{x_n}$ as *support vectors.* Solve b with $y_n(\boldsymbol{w^t x_n} + b) = 1$ for any $\alpha_n \geq 0$.

Then the separating plane is known with $\boldsymbol{w}$ and b. Note the generalization of SVM concerns the number of support vectors and the size of dataset, not the dimension of the data points.

3.3.3 Nonlinear separable case: Kernel function

We assumed that the data is linear separable above. If data is not linear separable, non-linear transformation can be used to convert the data from original space $\mathscr{X}$ into a high dimension space $\mathscr{Z}$ (i.e.

create a new data point $\boldsymbol{z_i}$ from each of data points $\boldsymbol{x_i}$), where the new data is linear separable in $\mathscr{Z}$, and our linear separable assumption is valid then.
We can observe that after non-linear transformation the only difference made to SVM formula is that $\boldsymbol{x^t x}$ becomes $\boldsymbol{z^t z}$, and the generated $\boldsymbol{w}$ can be applied in the original space $\mathscr{X}$. Eventually, we will have a curve surface to separate the data points in $\mathscr{X}$, instead of a straight plane in the linear separable case.
We can observe that in SVM, x only appears in the form of dot product with its transpose, $\boldsymbol{x^t x}$, implying that it is not $\boldsymbol{z}$ itself but $\boldsymbol{z^t z}$ is concerned in SVM. If we define $\boldsymbol{z_i^t z_i}$ as the kernel $\mathscr{K}(\boldsymbol{x_i}, \boldsymbol{x_i})$, we can drop $\boldsymbol{z}$ out of the SVM formula by claiming that as long as a function $\mathscr{K}(\boldsymbol{x_i}, \boldsymbol{x_i})$ can be written in terms of $\boldsymbol{z_i^t z_i}$ for an existing $\boldsymbol{z}$ of any space, even if in infinite dimension, $\mathscr{K}$ is a valid kernel and can be put in the place of $\boldsymbol{x^t x}$. Then we can claim our hypothesis as

$$g(\boldsymbol{x}) = sign(\sum_{\alpha_n > 0} \alpha_n y_n \mathscr{K}(\boldsymbol{x_n}, \boldsymbol{x}) + b)$$

$$where\ b = y_m - \sum_{\alpha_n > 0} \alpha_n y_n \mathscr{K}(\boldsymbol{x_n}, \boldsymbol{x_m})\ for\ any\ support\ vector(\alpha_m > 0)$$

The valid kernel can be manually constructed by finding the corresponding $\boldsymbol{z}$. But apart from this tedious method, *Mercer's Condition* can be applied: $\mathscr{K}(\boldsymbol{x}, \boldsymbol{x'})$ is a valid kernel if and only if it is symmetric and the matrix $[\mathscr{K}(\boldsymbol{x_i}, \boldsymbol{x_j})]_{ij}$ is positive semi-definite for any $\boldsymbol{x_i}, ..., \boldsymbol{x_N}$.

3.4 Artificial Neural Network

We also train an Artificial Neural Network to solve for this classification problem. Artificial Neural Network is a sophisticated method and the following paragraph will give a brief introduction of Artificial Neural Network (will called ANN for simplicity in the later section).

3.4.1 Structure of Neural Network

ANN is inspired by the neural network in human brain and how the neuron works to deliver message. A neural network consists of an input layer, an output layer and some hidden layers in between. On each layer there are many nodes and the nodes of different layers are connected by some weights represented by lines (See Appendix 3.4.1). For a true neuron in human brain, it receives some signal from the outside, then evaluate the signal and become activate if the signal is strong enough to pass through some threshold. Similarly, in neural network, each nodes, or "neuron", in the hidden layers will receive some input, which is the linear combination of the output of the preceding neurons and the weights that connects to the underlying neuron and those preceding neurons. A bias term and it corresponding weights are also added in each layers to serve as the thresholds for the activation of the neurons in the next layer [4].

3.4.2 Activation Function

The input of each neuron will pass through an activation function in order to generate the output of that neuron. In biologically inspired neural network, activation function has only binary output to mimic the "active"(1) and "inactive"(0) state of the neuron while in ANN, we replace it with some smooth functions that map real number to a range between 0 and 1 or -1 and 1. The advantage of smooth function is that beside indicating the state of activation, it also reveals the extent of activation and meanwhile makes the activation function differentiable. The differentiability of activation function is crucial when we use backpropagation to update the weights and bias. Typical activation functions includes logistic(sigmoid) function, tanh function and softmax function. Their function, domain and range is shown is appendix 3.4.2.

It's worth mentioning that tanh is used as the activation function of the hidden layer while softmax is adopted as the activation function of the output layer. Tanh is chosen to be the activation function because it has stronger gradient than sigmoid, which will be more efficient in optimization. The range of tanh is $[-1, 1]$ while the range of sigmoid is $[0, 1]$. Plots comparing the range of the derivatives of tanh and sigmoid are given in the appendix 3.4.3. The range of the derivative of tanh is $[0.42, 1]$ while that of sigmoid is $[0.2, 0.25]$ [5]. It's obvious that tanh function provide stronger gradient. Softmax function is implemented to transform the output in the output layer into probability. It has the following formulae:

$$Pr(y = j|o_j) = \frac{e^{o_j}}{\sum_{k=1}^{K} e^{o_k}}$$

These probabilities tells us how likely the model think an observation is belong to a certain class and they are directly used in the computation of the loss of the model, which is our optimization objective [6].

3.4.3 Loss Function and Regularization

The loss function has been chosen to be log loss. Log loss is equivalent to cross entropy when class=2. Their formula are as follow:

$$\text{Log Loss: } L(p, y) = -ylog(p) - (1 - y)log(1 - p)$$
$$\text{Cross Entropy: } L(P, Y) = -\sum_k y_k log(p_k)$$

P_k is the probability predicted by the model, y_k is the true class that the observation belongs to and k indicates the class. P and Y are both encoded in a form of vector, like dummy variable. For example, in our case, which is binary classification, for an observation in class 1, $Y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and $y_1 = 1$, $y_2 = 0$, P may be $\begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$ and $p_1 = 0.6$ and $p_2 = 0.4$. Cross entropy will be large when the $y_k = 1$ and p_k is close to 0, meaning that the model classify the observation wrong.

We also add L1 regularization to the loss function:

$$L = -\frac{1}{n}\sum_{allx}\sum_k y_k log(p_k) + \lambda\sum_l\sum_i\sum_j |w_{ij}^{(l)}|$$

$w_{ij}^{(l)}$ means the weight from the i^{th} node in layer l to the j^{th} node in layer $l+1$. L1 regularization takes the same logic from lasso regression. It has the ability to reduce some of the weights to zero. The logic behind can be easily understood by appendix 3.4.4. By forcing some of the weights to be zero, some of the neurons in our model will "die" and we do this in order to avoid over-fitting our model. Noted that regularization is only applied to weights instead of bias.

3.4.4 Gradient Descent and Backpropagation

With the above loss function, we can now update the weights to minimizing the loss function. The usual algorithm is called gradient descent. The algorithm of update one weight in the model at a time as follow:

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial L}{\partial w_{ij}}$$

η here is called learning rate, which control the step size of the movement along the direction of the gradient. t stands for the tth iteration of the of the update and the each iteration is also known as an "epoch". The gradient, or the partial derivative of the loss function L with respect to the weigh w_{ij} can be found using a method called backpropagation. Backpropagation has essentially two phases. Phase 1 is called propagation, or forward propagation. In phase 1, we propagate the model forward through the network to generate the output and calculate the error(loss) via loss function. Phase 2 is called backpropagation, in which we propagate the predicted output back through the model using the true training target pattern to generate delta of all output and hidden layer [7]. Delta here is the difference between the targeted and actual output value. The gradient of the weight is given by the product of weight's output delta and input activation. An example illustrating how to obtain the gradient with detail derivation will be given in the appendix 3.4.5.

3.4.5 Stochastic Gradient Descent and Mini-batch Gradient Descent

Obtaining the gradient using backpropagation is computationally intensive. In reality, because of the large size of the data set and the complicated structure of the neural network, gradient descent is extremely slow. In order to solve this problem, stochastic gradient descent(SGD) and mini-batch gradient descent are proposed. The difference between traditional gradient descent, or so call batch gradient descent, and SGD is that instead of taking the whole batch of data to calculate the gradient, SGD randomly draws one piece of training example out of the training set at each iteration and performs backpropagation solely based on it. There is a good metaphor illustrating the difference between SGD and batch gradient descent: SGD is like a drunken man walking downhill while batch gradient descent is like a carefully guy calculating each of his step prudentially in order to go downhill in the steepest way at each step. Mini-batch gradient descent is a compromise between batch gradient descent and SGD, which take a small batch of training example to perform the backpropagation update [8].

3.4.6 Model Fitting and Tuning Using R

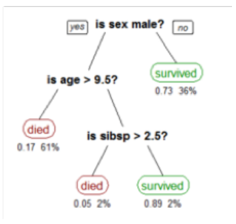
neuralnetwork function in *ANN2* package is used to train our neural network model. ANN2 is capable of performing SGD. 20% of the original train set is randomly selected out to be validation set while the remaining will still be the training set. Common method for parameter selection in neural network might be performing a grid search with all combination of parameters using cross validation. However, we didn't use this method since it's too time consuming and effectiveness. An alternative way is to tune the model using the loss plot of the training set and validation set. The loss plot plot loss of each epoch and it's a commonly used diagnosis plot to detect over fitting. Our final model has the following set of parameters: batchsize=32, learning rate=$1e-4$, L1(λ)=0.03. The structure of the network is 59-5-2. 59 input nodes is result from changing the categorical variables in to dummy variables. Results will be shown in later section and loss plot can be found in the appendix 3.4.7

3.5 Random Forest

Random forests are built by combining the predictions of several trees, each of which is trained in isolation. Unlike in boosting (Schapire & Freund, 2012) where the base models are trained and combined using a sophisticated weighting scheme, typically the trees are trained independently then take a majority vote to determine what class should the observation belongs to.

To understand RF better, we need some knowledge about classification tree and bagging.

Classification tree is a popular method in machine learning when tackling classification problem. We build up the tree from the top to the bottom, based on the formula of information gain like what is shown here. H(T) stands for the entropy of the parent nodes and H(H|a) is the weighted sum of the entropy of the children nodes given we choose a as our split. The variable with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0. A typical classification tree flow is as following:

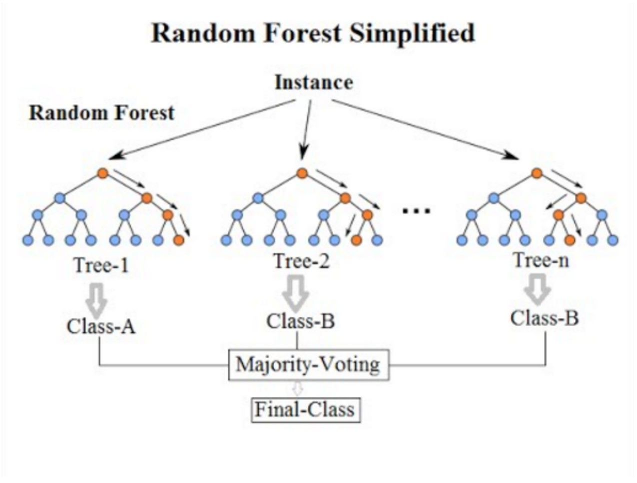


For bagging, or bootstrap aggregating, it is basically an algorithm designed to improve the stability and accuracy of machine learning algorithm for classification and regression.Bagging is usually applied to decision tree methods but in general it can be applied to any types of methods. It is shown that bagging is leads to "improvement of unstable procedure", like ANN, decision tree and subset selection of linear regression while it may mildly degrade the performance of stable methods such as KNN. (A stable learning algorithm is one for which the prediction does not change dramatically when the

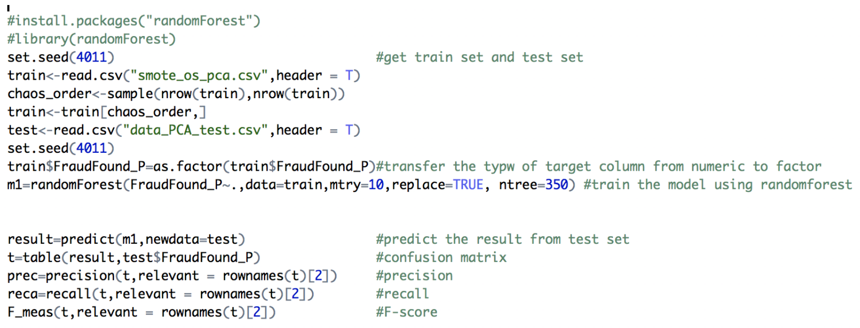
training data is modified slightly.)
In general, the sample size for a random forest acts as a control on the "degree of randomness" involved, and thus as a way of adjusting the bias-variance tradeoff. Increasing the sample size results in a "less random" forest, and so has a tendency to overfit. Decreasing the sample size increases the variation in the individual trees within the forest, preventing overfitting, but usually at the expense of model performance. A useful side-effect is that lower sample sizes reduce the time needed to train the model.

The usual rule of thumb for the best sample size is a "bootstrap sample", a sample equal in size to the original dataset, but selected with replacement, so some rows are not selected, and others are selected more than once. Using a bootstrapped data set, each tree in the forest is trained on slightly different data, which introduces differences between the trees, and further randomness is introduced by identifying the best split feature from a random subset of available features.
Advantages Random forest is an ensemble method in which a classifier is constructed by combining several different Independent base classifiers.

- By combining different classifier predictions together, the net of information is much greater. Furthermore, the more diverse source of information, the more robust the random forest is because it will not be swayed by a single anomalous data source.
- The prediction is less likely to be wrong since it can be shown that an ensemble of independent classifiers, each with an error rate e, when combined significantly reduces the error rate.



randomForest in R The following is the R code of random Forest



```
#install.packages("randomForest")
#library(randomForest)
set.seed(4011)                                     #get train set and test set
train<-read.csv("smote_os_pca.csv",header = T)
chaos_order<-sample(nrow(train),nrow(train))
train<-train[chaos_order,]
test<-read.csv("data_PCA_test.csv",header = T)
set.seed(4011)
train$FraudFound_P=as.factor(train$FraudFound_P)#transfer the typw of target column from numeric to factor
m1=randomForest(FraudFound_P~.,data=train,mtry=10,replace=TRUE, ntree=350) #train the model using randomforest


result=predict(m1,newdata=test)                    #predict the result from test set
t=table(result,test$FraudFound_P)                  #confusion matrix
prec=precision(t,relevant = rownames(t)[2])        #precision
reca=recall(t,relevant = rownames(t)[2])           #recall
F_meas(t,relevant = rownames(t)[2])                #F-score
```

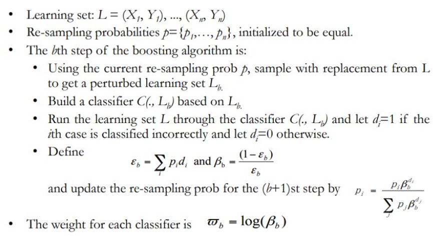
Remarks

- The type of target column must be transferred to factor or there will be error running random-Forest.
- In this pair of train and test dataset, there are 27 variables. 10 variables are selected as decision nodes randomly in each tree.

- There are 350 trees in the forest. If the number of trees is too small, the randomness of the result is too high; if the number of trees is too large, there would be error message that R cannot allocate large size data.
- "replace=true" means using bagging method.

3.6 Boosting

After we use the single method to solve the problem, we find the F score is low and each time there are many misclassifications. So we decide to use the boosting to combine the different classifier to reduce the error. Each time the sample probability is changed according the misclassification you make and calculate the corresponding weight. We used the “-1” to denote the non-fraud and “1” to stand for the fraud case, we computed the weighted sum of the testing result and to make it as fraud case when summation result is bigger than 0.[2]



- Learning set: $L = (X_1, Y_1), ..., (X_n, Y_n)$
- Re-sampling probabilities $p=\{p_1,\dots,p_n\}$, initialized to be equal.
- The bth step of the boosting algorithm is:
  - Using the current re-sampling prob p, sample with replacement from L to get a perturbed learning set L_b.
  - Build a classifier $C(., L_b)$ based on L_b.
  - Run the learning set L through the classifier $C(., L_b)$ and let d_i=1 if the ith case is classified incorrectly and let d_i=0 otherwise.
  - Define $\varepsilon_b = \sum_i p_i d_i$ and $\beta_b = \frac{(1-\varepsilon_b)}{\varepsilon_b}$

    and update the re-sampling prob for the (b+1)st step by $p_i = \frac{p_i \beta_b^{d_i}}{\sum_j p_j \beta_b^{d_j}}$
- The weight for each classifier is $\varpi_b = \log(\beta_b)$

We selected 20 as our iteration times, the classifier used in the boosting are KNN, SVM and classification tree. We used the R to implement the algorithm in above way.

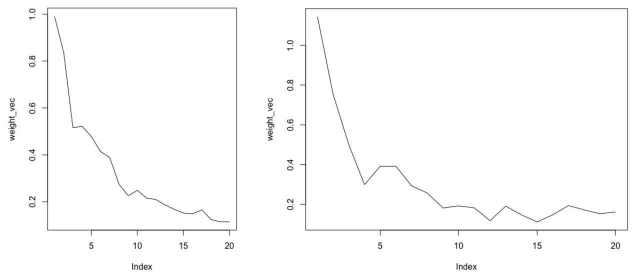
4 Results: tables and graphs

Bagging and Boosting does not perform well according to the F score. Choose under sample and over sample as example.Others can be found in the appendix.

	under sam	over sam		under sam	over sam
Normal-KNN	0.1861	0.1872	Normal-SVM	0.2109	0.2172
Bagging-KNN	0.1831	0.1503	Bagging-SVM	0.211	0.2096
Boost-KNN	0.1663	0.1349	Boost-SVM	0.2085	0.209

Initially, boosting is supposed to correctly classify those mis-classified at previous step, but when we see the weight sequence plot, we find that the error tend to increasing, which means that mistake cannot be corrected by any classifier.We show the boosting with KNN & SVM and B=20 as example.

Figure 1: Boosting weight for KNN and SVM



Results from five methods

Table 1: F score for five methods

	over	under	smote	PCA over	PCA under	PCAsmote
RF	0.16492	0.23145	0.24424	0.19081	0.22138	0.22678
KNN	0.16725	0.18002	0.17701	0.17241	0.18413	0.1699
Logistics	0.21086	0.22244	0.2122	0.20796	0.21668	0.20942
ANN	0.2181	0.2254	0.2107	0.2173	0.2318	0.2106
SVM	0.2237	0.216	0.2201	0.2378	0.22915	0.2177

Table 2: Recall of five methods

	over	under	smote	PCA over	PCA under	PCAsmote
RF	0.13751	0.69276	0.59314	0.556955	0.61373	0.67947
KNN	0.57594	0.57547	0.59485	0.59887	0.58565	0.71662
Logistics	0.86399	0.72811	0.75838	0.87251	0.7316	0.85823
SVM	0.91885	0.9146	0.9188	0.8507	0.82965	0.91355
ANN	0.9365	0.7692	0.8195	0.9106	0.7457	0.8805

- Overall, the KNN perform worse than other method, it may imply that training is still necessary to get some information from the training set.
- No matter what kinds of sampling used, the F score seems no significant difference.
- ANN,SVM & Logistics have higher better ability to classify the fraud correctly, but they are not the best method, which imply that they also categorize many non-fraud to fraud, which make the F score lower.

Below is the ten testing results from our best model, we apply the smote oversampling method to deal with the dataset and use "draws" index to extract ten dataset. It is unreasonable to apply our classifier without re-train the data because some of the testing data point are already in our previous training set, which will cause the over-fitting. So we remove corresponding testing index and make 10 training set for each testing dataset, after that we train the RandomForest on these ten dataset and get corresponding confusion matrix.

RandomForest			recall	F_score	F_mean
smote_OS_1	749	23	0.5741	0.2199	0.24424
	197	31			
smote_OS_2	742	19	0.6842	0.2626	
	201	39			
smote_OS_3	769	22	0.6271	0.2761	
	172	37			
smote_OS_4	746	24	0.5932	0.2422	
	195	35			
smote_OS_5	746	20	0.7015	0.3123	
	187	47			
smote_OS_6	736	27	0.5714	0.24	
	201	36			
smote_OS_7	777	27	0.4706	0.1943	
	172	24			
smote_OS_8	751	25	0.5283	0.2022	
	196	28			
smote_OS_9	759	25	0.5192	0.2015	
	189	27			
smote_OS_10	736	23	0.6618	0.2913	
	196	45			

Con_matrix

TN	FN
FP	TP

	TN	FN	FP	TP
Min	736	19	172	24
Q1	743	22.25	187.5	28.75
Median	747.5	23.5	195.5	35.5
Q3	757	25	196.75	38.5
Max	777	27	201	47
mean	751.1	23.5	190.6	34.9
Sd	13.53555	2.6770631	10.762073	7.5638027

5 Limitations

First limitation is about the time gap we calculate for the claim, since there exists a complete fifth week in a month(which is not understandable), we cannot get accurate time gap in those cases. Secondly, since some methods like SVM, KNN, ANN are supposed to use numerical variables as input, our numeric variables transferred from categorical variables may not express real information, then the methods may not perform well

REFERENCES

[1] Jolliffe I. (2011) Principal Component Analysis. In: Lovric M. (eds) International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg

[2] Boosting Methods
`http://di.ulb.ac.be/map/gbonte/bioinfo/boosting.pdf`

[3] Introduction to k-Nearest Neighbors
`https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/`

[4] Artificial Neural Network
`https://en.wikipedia.org/wiki/Artificial_neural_network`

[5] Tanh activation function vs Sigmoid activation function
`https://stats.stackexchange.com/questions/101560/tanh-activation-function-vs-sigmoid-activation-function`

[6] Why is the softmax function often used as activation function of output layer in classification neural networks?
`https://datascience.stackexchange.com/questions/37357/why-is-the-softmax-function-often-used-as-activation-function-of-output-layer-in`

[7] Backpropagation
`https://en.wikipedia.org/wiki/Backpropagation`

[8] Stochastic gradient descent
`https://en.wikipedia.org/wiki/Stochastic_gradient_descent`

[9] Derivative of Softmax Loss Function
`https://math.stackexchange.com/questions/945871/derivative-of-softmax-loss-function`

[10] A Step By Step Backpropagation Example
`https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`

[11] Bagging explanation
`https://stats.stackexchange.com/questions/24330/is-there-a-formula-or-rule-for-determining-the-correct-sampsize-for-a-randomfore`

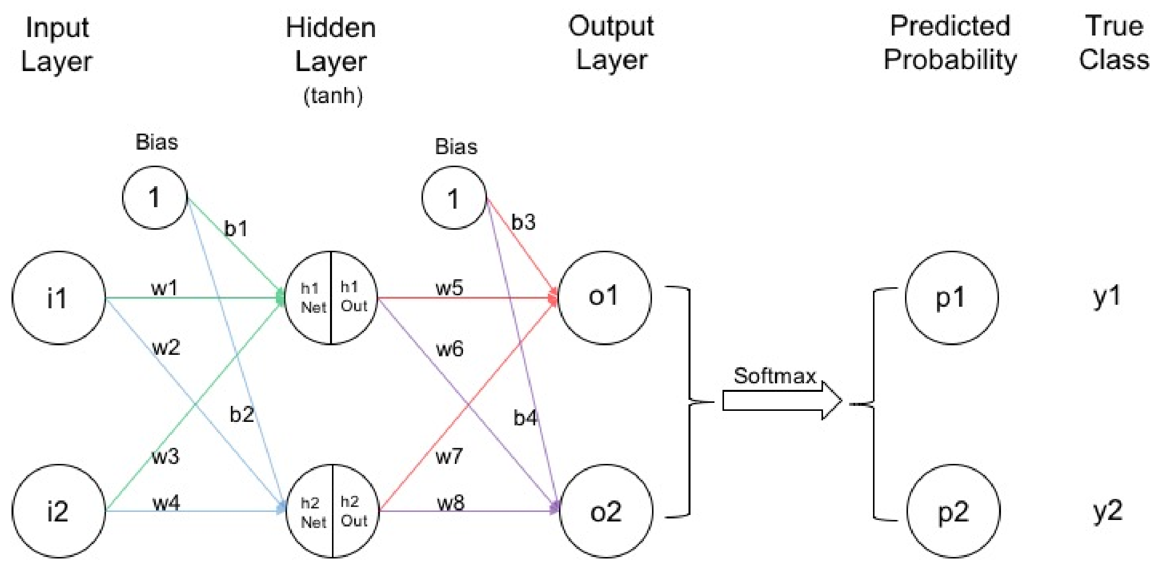
[12] RandomForest
`https://en.wikipedia.org/wiki/Random_forest`

Appendix

For convenience, I put the necessary file and code in the google drive, you can open the following url directly.

`https://drive.google.com/drive/folders/1snhEfhPJI_iMLLKqkTguXGMUKGH4tB0d?usp=sharing`

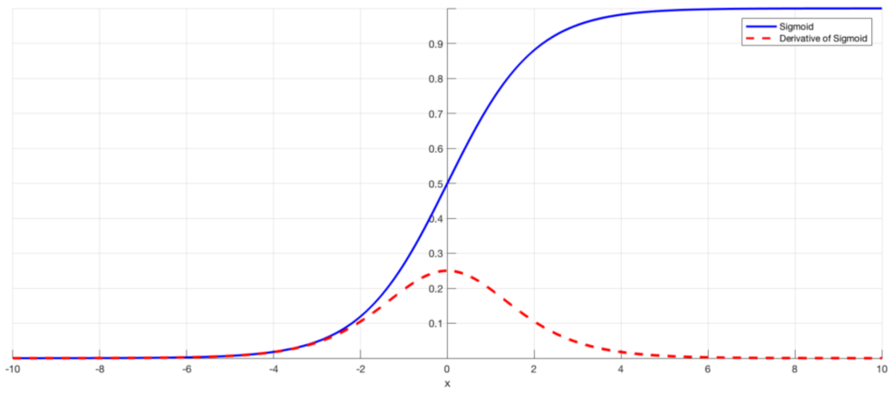
Appendix 3.4.1
Structure of Neural Network

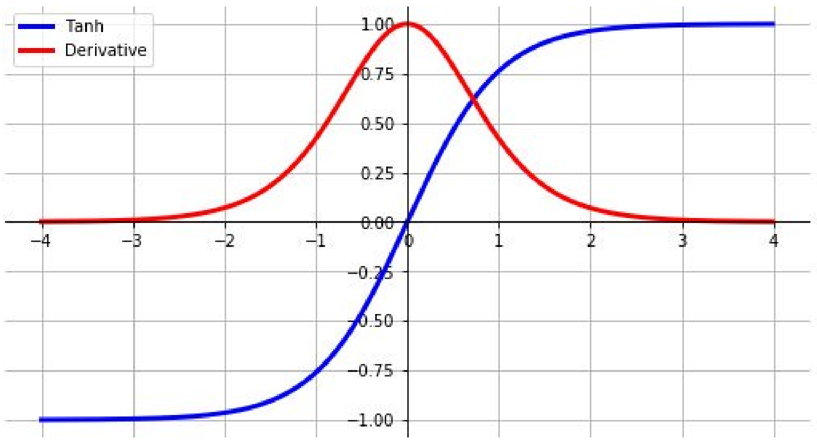


Appendix 3.4.2
Properties of Activation Functions

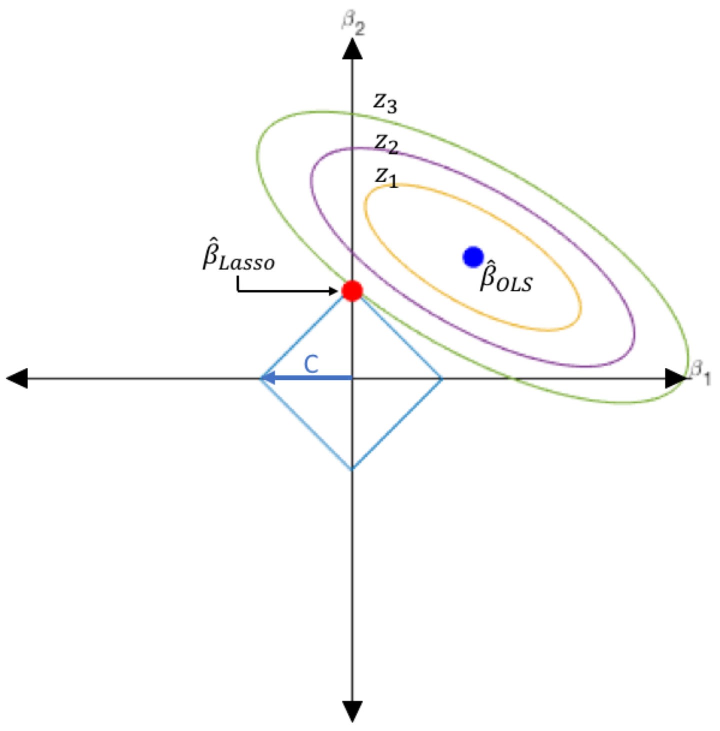
Name	Function	Domain	Range
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$\mathbb{R}$	$(0,1)$
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\mathbb{R}$	$(-1,1)$
Sofmax	$f_i(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$	$\mathbb{R}^K$	$(0,1)$

Appendix 3.4.3
Plots of Sigmoid and Tanh Function and their First Derivatives





Appendix 3.4.4
Illustration of L1 Penalty similar to Lasso



Appendix 3.4.5
Derivation of Partial Derivative of the Loss Function (Cross Entropy) with Softmax Function as Transformation Function [9]

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}}$$
$$L = -\sum_k y_k log(p_k)$$

We encode $Y = (y_k)$ as dummy variable. Since y_k can only be 0 or 1, we will have $\sum_k y_k = 1$.

1. $j = i$

$$\begin{aligned}\frac{\partial p_j}{\partial o_i} &= \frac{e^{o_i}\Sigma - e^{o_i}e^{o_i}}{\Sigma} \\ &= \frac{e^{o_j}}{\Sigma}\frac{\Sigma - e^{o_i}}{\Sigma} \\ &= p_i(1-p_i)\end{aligned}$$

Note: Here we use Σ to represent $\sum_k e^{o_k}$.

2. $j \neq i$

$$\begin{aligned}\frac{\partial p_j}{\partial o_i} &= \frac{0 - e^{o_j}e^{o_i}}{\Sigma^2} \\ &= -p_i p_j\end{aligned}$$

The partial derivatives of loss function(Cross Entropy) with softmax function as transformation function can be derived as follow:

$$\begin{aligned}\frac{\partial L}{\partial o_i} &= \frac{\partial}{\partial o_i}(-\sum_k y_k log(p_k)) \\ &= -\sum_k y_k \frac{\partial}{\partial o_i} log(p_k) \\ &= -\sum_k y_k \frac{\partial}{\partial p_k} log(p_k) \frac{\partial}{\partial o_i} p_k \\ &= -\sum_k y_k \frac{1}{p_k}(\frac{\partial}{\partial o_i} p_k) \\ &= -y_i \frac{1}{p_i} p_i(1-p_i) - \sum_{k\neq i} y_k \frac{1}{p_k}(-p_i p_k) \\ &= -y_i(1-p_i) + \sum_{k\neq i} y_k p_i \\ &= -y_i + \sum_k y_k p_i \\ &= p_i(\sum_k y_k) - y_i \\ &= p_i - y_i\end{aligned}$$

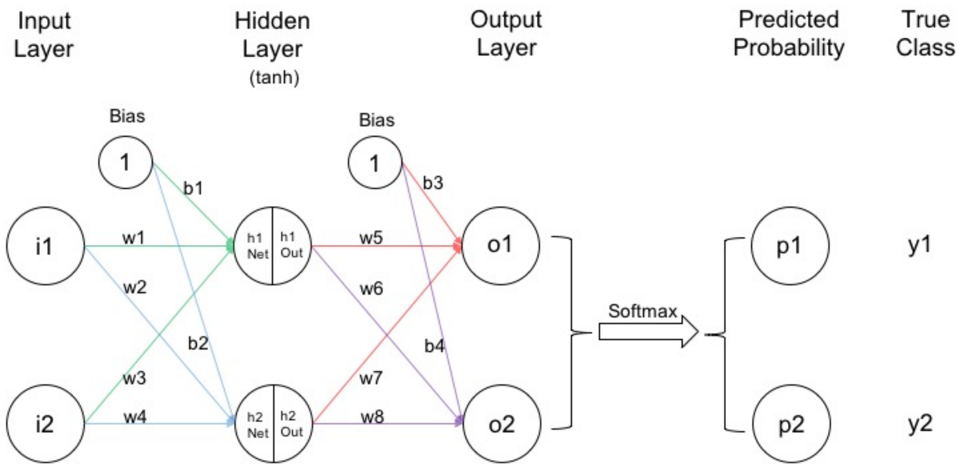
Appendix 3.4.6
Derivation of Gradients using Backpropagation with Simple Example [10]

As illustrated in the article, weights are updated using the following formula:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial L}{\partial w_{ij}}$$

Therefore we need to compute the gradient $\frac{\partial L}{\partial w_{ij}}$ using backpropagration.

For demonstration, we consider a simple neural network having only one hidden layer with two nodes, two input nodes, and two output nodes. h_{1Net} is the input of the activation function $tanh$ while h_{1Out} is the corresponding output after applying the activation function. The whole setting of this simple neural network is similar to our ANN model except for the structure.



The goal is to obtain the gradient with respect to weights between layers. w_1 and w_5 will be derived as examples in the following section.

$$h_{1Net} = b_1 + w_1 i_1 + w_3 i_2 \tag{1}$$

$$h_{1Out} = tanh(h_{1Net}) = \frac{e^{h_{1Net}} - e^{h_{1Net}}}{e^{h_{1Net}} + e^{h_{1Net}}} \tag{2}$$

$$o_1 = b_3 + w_5 h_{1Out} + w_7 h_{2Out} \tag{3}$$

$$o_2 = b_4 + w_6 h_{1Out} + w_8 h_{2Out} \tag{4}$$

$$Pr(y=1|obs) = p_1 = softmax(o_1) = \frac{e^{o_1}}{\sum_{k=1}^{2} e^{o_k}} \tag{5}$$

$$Pr(y=2|obs) = p_2 = softmax(o_1) = \frac{e^{o_2}}{\sum_{k=1}^{2} e^{o_k}} \tag{6}$$

$$L = -\sum_k y_k log(p_k) = -y_1 log(p_1) - y_2 log(p_2) \tag{7}$$

- $\frac{\partial L}{\partial w_5}$

  1. Apply chain rule, we can expand the gradient as: $\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial o_1}\frac{\partial o_1}{\partial w_5} = \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial w_5}$
  2. $\frac{\partial L}{\partial o_1} = (p_1 - y_1)$. (Derived in appendix 3.4)
  3. From (3), $\frac{\partial o_1}{\partial w_5} = \frac{\partial}{\partial w_5}(b_3 + w_5 h_{1Out} + w_7 h_{2Out}) = h_{1Out}$.
  4. $\frac{\partial L}{\partial w_5} = (p_1 - y_1) h_{1Out}$

- $\frac{\partial L}{\partial w_1}$
  1. Apply chain rule again. Noticed that now w_1 will affect o_1 and o_2, therefore we have $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial h_{1Out}}\frac{\partial h_{1Out}}{\partial h_{1Net}}\frac{\partial h_{1Net}}{\partial w_1} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial h_{1Out}}\frac{\partial h_{1Out}}{\partial h_{1Net}}\frac{\partial h_{1Net}}{\partial w_1}$
  2. $\frac{\partial L}{\partial o_1} = (p_1 - y_1)$; $\frac{\partial L}{\partial o_2} = (p_2 - y_2)$.
  3. From (3), $\frac{\partial o_1}{\partial h_{1Out}} = \frac{\partial}{\partial h_{1Out}}(b_3 + w_5 h_{1Out} + w_7 h_{2Out}) = w_5$. Similarly, from (4), $\frac{\partial o_2}{\partial h_{1Out}} = \frac{\partial}{\partial h_{1Out}}(b_4 + w_6 h_{1Out} + w_8 h_{2Out}) = w_6$.
  4. Note that $\frac{d}{dx}(tanh(x)) = 1 - tanh^2(x)$. Combined with (2) we have $\frac{\partial h_{1Out}}{\partial h_{1Net}} = 1 - tanh^2(h_{1Net})$.
  5. From (1), $\frac{\partial h_{1Net}}{\partial w_1} = \frac{\partial}{\partial w_1}(b_1 + w_1 i_1 + w_3 i_2) = i_1$.
  6. Assemble all the components and we will have

$$\begin{aligned}\frac{\partial L}{\partial w_1} &= (p_1 - y_1)w_5[1 - tanh^2(h_{1Net})]i_1 + (p_2 - y_2)w_6[1 - tanh^2(h_{1Net})]i_1 \\ &= [(p_1 - y_1)w_5 + (p_2 - y_2)w_6][1 - tanh^2(h_{1Net})]i_1\end{aligned}$$

We further inspect the gradients and rewrite their form as follows:

$$\frac{\partial L}{\partial w_5} = (p_1 - y_1)h_{1Out} = \delta_5 \times h_{1Out} \tag{8}$$

$$\frac{\partial L}{\partial w_1} = [(p_1 - y_1)w_5 + (p_2 - y_2)w_6][1 - tanh^2(h_{1Net})]i_1 = \delta_1 \times i_1 \tag{9}$$

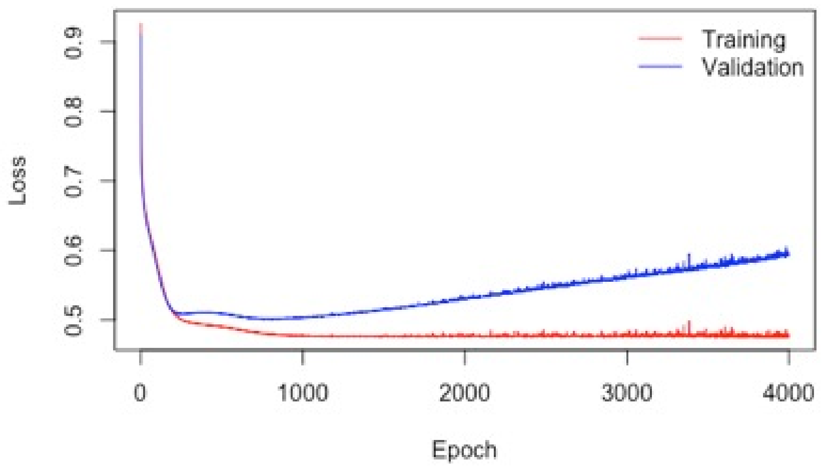
where $\delta_5 = (p_1 - y_1)$ and $\delta_1 = [(p_1 - y_1)w_5 + (p_2 - y_2)w_6][1 - tanh^2(h_{1Net})]$.

It's obvious that the gradients do follow the form of delta times the input. From the hidden layer to the output layer, the difference between predicted and actual value is essentially $(p_i - y_i)$. The delta corresponding to w_1 can be understood as the cumulative difference related to w_1 times before performing transformation via activation function.

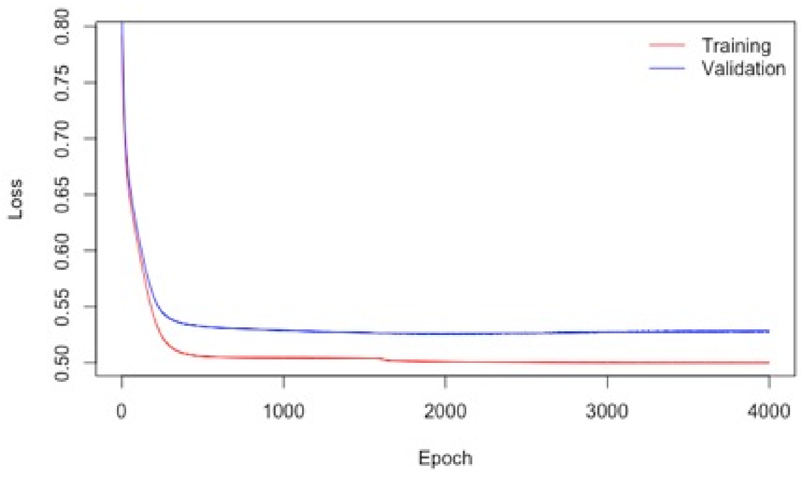
Meanwhile, noticed that when deriving the gradient $\frac{\partial L}{\partial w_1}$, the formula contains $(p_1 - y_1)$ and $(p_2 - y_2)$. These two components have been calculated when we deriving the gradient $\frac{\partial L}{\partial w_5}$ and $\frac{\partial L}{\partial w_6}$. We are essentially reusing the results that we obtain before. In practice, we usually work out the gradients near the output layer at the back first, and then work backward to obtain the gradients at the front reusing the former results. The logic behind is that the delta at the front is affected by the delta at the back, therefore we can obtain the delta at the front by "back-propagate" the effect of deltas at the back. This is how backpropagation got its name.

Appendix 3.4.7
Loss Plot Examples

Loss plot indicating over-fitting

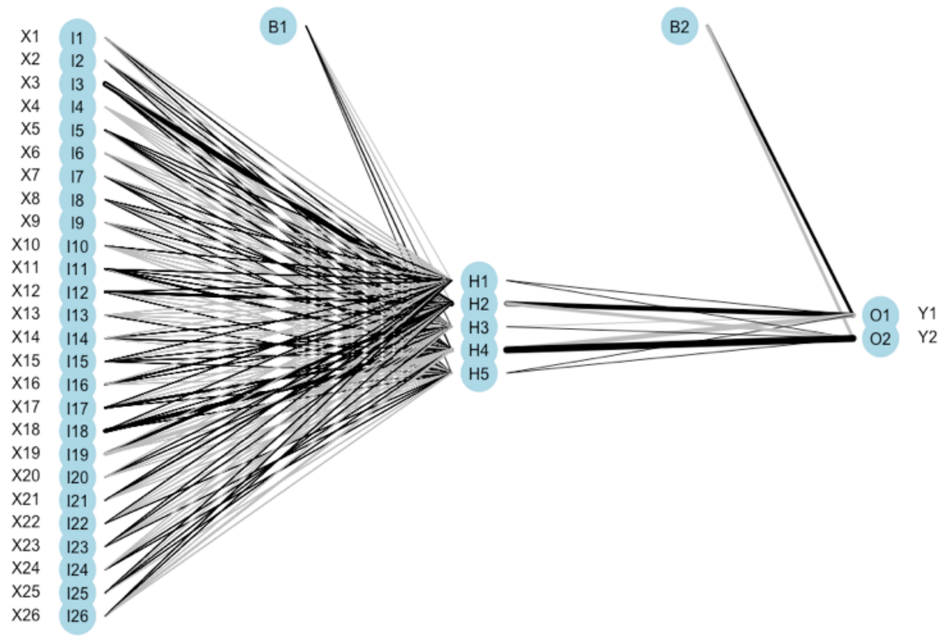


Normal loss plot

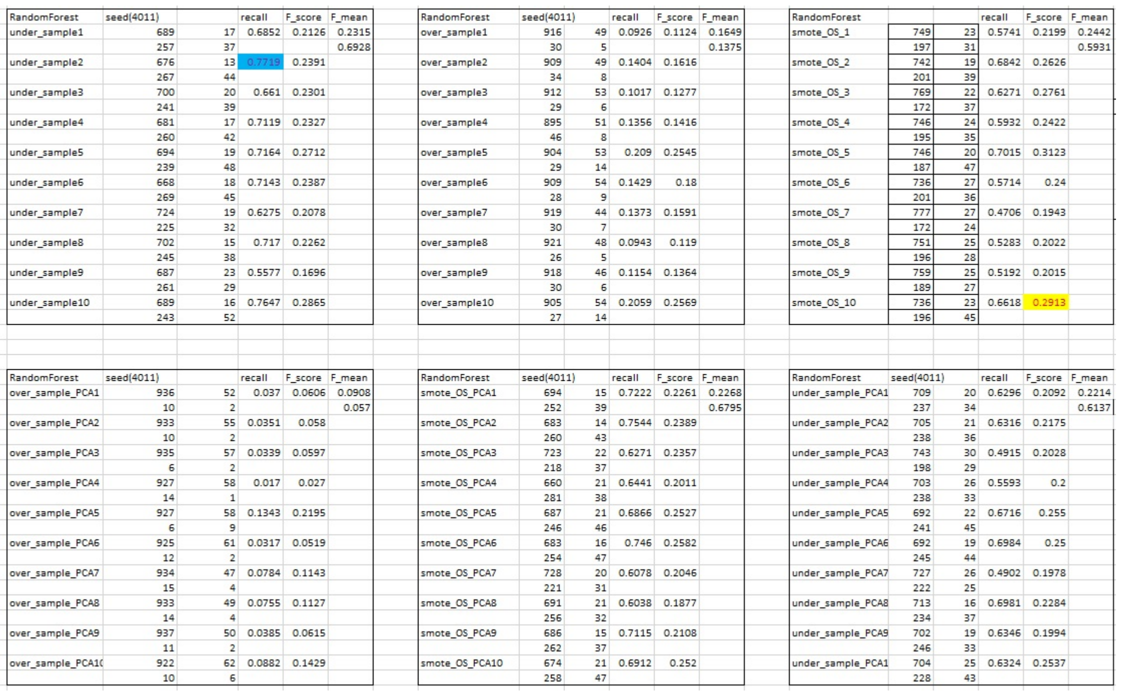


Appendix 3.4.8
Final ANN Model Structure (PCA Processed data)

Black lines represent positive weights while gray lines represent negative weights. The width of the line is determined by the absolute value of the weight.



Appendix 3.4.8 Result for running



RandomForest	seed(4011)		recall	F_score	F_mean
under_sample1	689	17	0.6852	0.2126	0.2315
	257	37			0.6928
under_sample2	676	13	0.7719	0.2391	
	267	44			
under_sample3	700	20	0.661	0.2301	
	241	39			
under_sample4	681	17	0.7119	0.2327	
	260	42			
under_sample5	694	19	0.7164	0.2712	
	239	48			
under_sample6	668	18	0.7143	0.2387	
	269	45			
under_sample7	724	19	0.6275	0.2078	
	225	32			
under_sample8	702	15	0.717	0.2262	
	245	38			
under_sample9	687	23	0.5577	0.1696	
	261	29			
under_sample10	689	16	0.7647	0.2865	
	243	52			

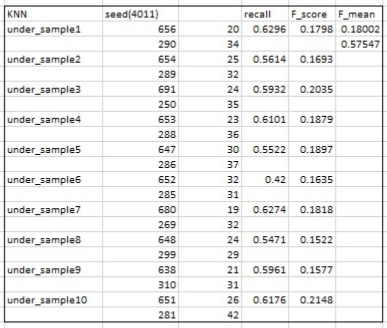
RandomForest	seed(4011)		recall	F_score	F_mean
over_sample1	916	49	0.0926	0.1124	0.1649
	30	5			0.1375
over_sample2	909	49	0.1404	0.1616	
	34	8			
over_sample3	912	53	0.1017	0.1277	
	29	6			
over_sample4	895	51	0.1356	0.1416	
	46	8			
over_sample5	904	53	0.209	0.2545	
	29	14			
over_sample6	909	54	0.1429	0.18	
	28	9			
over_sample7	919	44	0.1373	0.1591	
	30	7			
over_sample8	921	48	0.0943	0.119	
	26	5			
over_sample9	918	46	0.1154	0.1364	
	30	6			
over_sample10	905	54	0.2059	0.2569	
	27	14			

RandomForest			recall	F_score	F_mean
smote_OS_1	749	23	0.5741	0.2199	0.2442
	197	31			0.5931
smote_OS_2	742	19	0.6842	0.2626	
	201	39			
smote_OS_3	769	22	0.6271	0.2761	
	172	37			
smote_OS_4	746	24	0.5932	0.2422	
	195	35			
smote_OS_5	746	20	0.7015	0.3123	
	187	47			
smote_OS_6	736	27	0.5714	0.24	
	201	36			
smote_OS_7	777	27	0.4706	0.1943	
	172	24			
smote_OS_8	751	25	0.5283	0.2022	
	196	28			
smote_OS_9	759	25	0.5192	0.2015	
	189	27			
smote_OS_10	736	23	0.6618	0.2913	
	196	45			

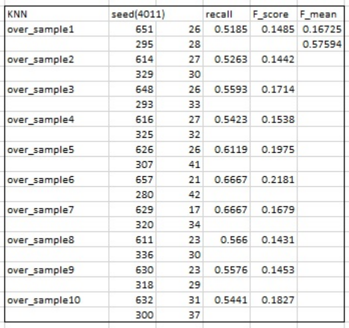
RandomForest	seed(4011)		recall	F_score	F_mean
over_sample_PCA1	936	52	0.037	0.0606	0.0908
	10	2			0.057
over_sample_PCA2	933	55	0.0351	0.058	
	10	2			
over_sample_PCA3	935	57	0.0339	0.0597	
	6	2			
over_sample_PCA4	927	58	0.017	0.027	
	14	1			
over_sample_PCA5	927	58	0.1343	0.2195	
	6	9			
over_sample_PCA6	925	61	0.0317	0.0519	
	12	2			
over_sample_PCA7	934	47	0.0784	0.1143	
	15	4			
over_sample_PCA8	933	49	0.0755	0.1127	
	14	4			
over_sample_PCA9	937	50	0.0385	0.0615	
	11	2			
over_sample_PCA1(	922	62	0.0882	0.1429	
	10	6			

RandomForest	seed(4011)		recall	F_score	F_mean
smote_OS_PCA1	694	15	0.7222	0.2261	0.2268
	252	39			0.6795
smote_OS_PCA2	683	14	0.7544	0.2389	
	260	43			
smote_OS_PCA3	723	22	0.6271	0.2357	
	218	37			
smote_OS_PCA4	660	21	0.6441	0.2011	
	281	38			
smote_OS_PCA5	687	21	0.6866	0.2527	
	246	46			
smote_OS_PCA6	683	16	0.746	0.2582	
	254	47			
smote_OS_PCA7	728	20	0.6078	0.2046	
	221	31			
smote_OS_PCA8	691	21	0.6038	0.1877	
	256	32			
smote_OS_PCA9	686	15	0.7115	0.2108	
	262	37			
smote_OS_PCA10	674	21	0.6912	0.252	
	258	47			

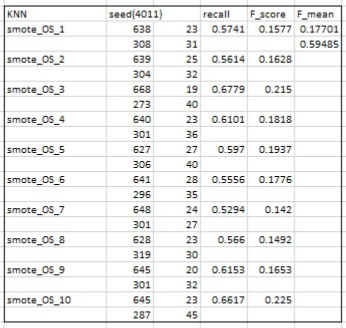
RandomForest	seed(4011)		recall	F_score	F_mean
under_sample_PCA1	709	20	0.6296	0.2092	0.2214
	237	34			0.6137
under_sample_PCA2	705	21	0.6316	0.2175	
	238	36			
under_sample_PCA3	743	30	0.4915	0.2028	
	198	29			
under_sample_PCA4	703	26	0.5593	0.2	
	238	33			
under_sample_PCA5	692	22	0.6716	0.255	
	241	45			
under_sample_PCA6	692	19	0.6984	0.25	
	245	44			
under_sample_PCA7	727	26	0.4902	0.1978	
	222	25			
under_sample_PCA8	713	16	0.6981	0.2284	
	234	37			
under_sample_PCA9	702	19	0.6346	0.1994	
	246	33			
under_sample_PCA1	704	25	0.6324	0.2537	
	228	43			



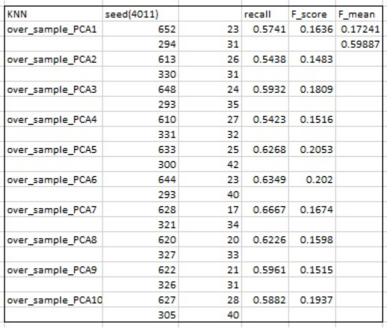
KNN	seed(4011)		recall	F_score	F_mean
under_sample1	656	20	0.6296	0.1798	0.18002
	290	34			0.57547
under_sample2	654	25	0.5614	0.1693	
	289	32			
under_sample3	691	24	0.5932	0.2035	
	250	35			
under_sample4	653	23	0.6101	0.1879	
	288	36			
under_sample5	647	30	0.5522	0.1897	
	286	37			
under_sample6	652	32	0.42	0.1635	
	285	31			
under_sample7	680	19	0.6274	0.1818	
	269	32			
under_sample8	648	24	0.5471	0.1522	
	299	29			
under_sample9	638	21	0.5961	0.1577	
	310	31			
under_sample10	651	26	0.6176	0.2148	
	281	42			



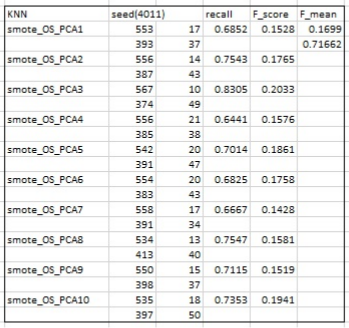
KNN	seed(4011)		recall	F_score	F_mean
over_sample1	651	26	0.5185	0.1485	0.16725
	295	28			0.57594
over_sample2	614	27	0.5263	0.1442	
	329	30			
over_sample3	648	26	0.5593	0.1714	
	293	33			
over_sample4	616	27	0.5423	0.1538	
	325	32			
over_sample5	626	26	0.6119	0.1975	
	307	41			
over_sample6	657	21	0.6667	0.2181	
	280	42			
over_sample7	629	17	0.6667	0.1679	
	320	34			
over_sample8	611	23	0.566	0.1431	
	336	30			
over_sample9	630	23	0.5576	0.1453	
	318	29			
over_sample10	632	31	0.5441	0.1827	
	300	37			



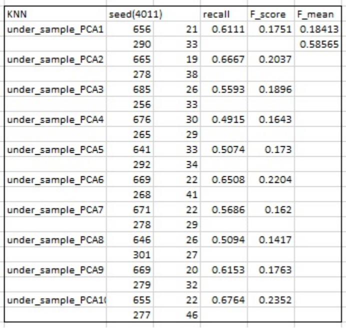
KNN	seed(4011)		recall	F_score	F_mean
smote_OS_1	638	23	0.5741	0.1577	0.17701
	308	31			0.59485
smote_OS_2	639	25	0.5614	0.1628	
	304	32			
smote_OS_3	668	19	0.6779	0.215	
	273	40			
smote_OS_4	640	23	0.6101	0.1818	
	301	36			
smote_OS_5	627	27	0.597	0.1937	
	306	40			
smote_OS_6	641	28	0.5556	0.1776	
	296	35			
smote_OS_7	648	24	0.5294	0.142	
	301	27			
smote_OS_8	628	23	0.566	0.1492	
	319	30			
smote_OS_9	645	20	0.6153	0.1653	
	301	32			
smote_OS_10	645	23	0.6617	0.225	
	287	45			



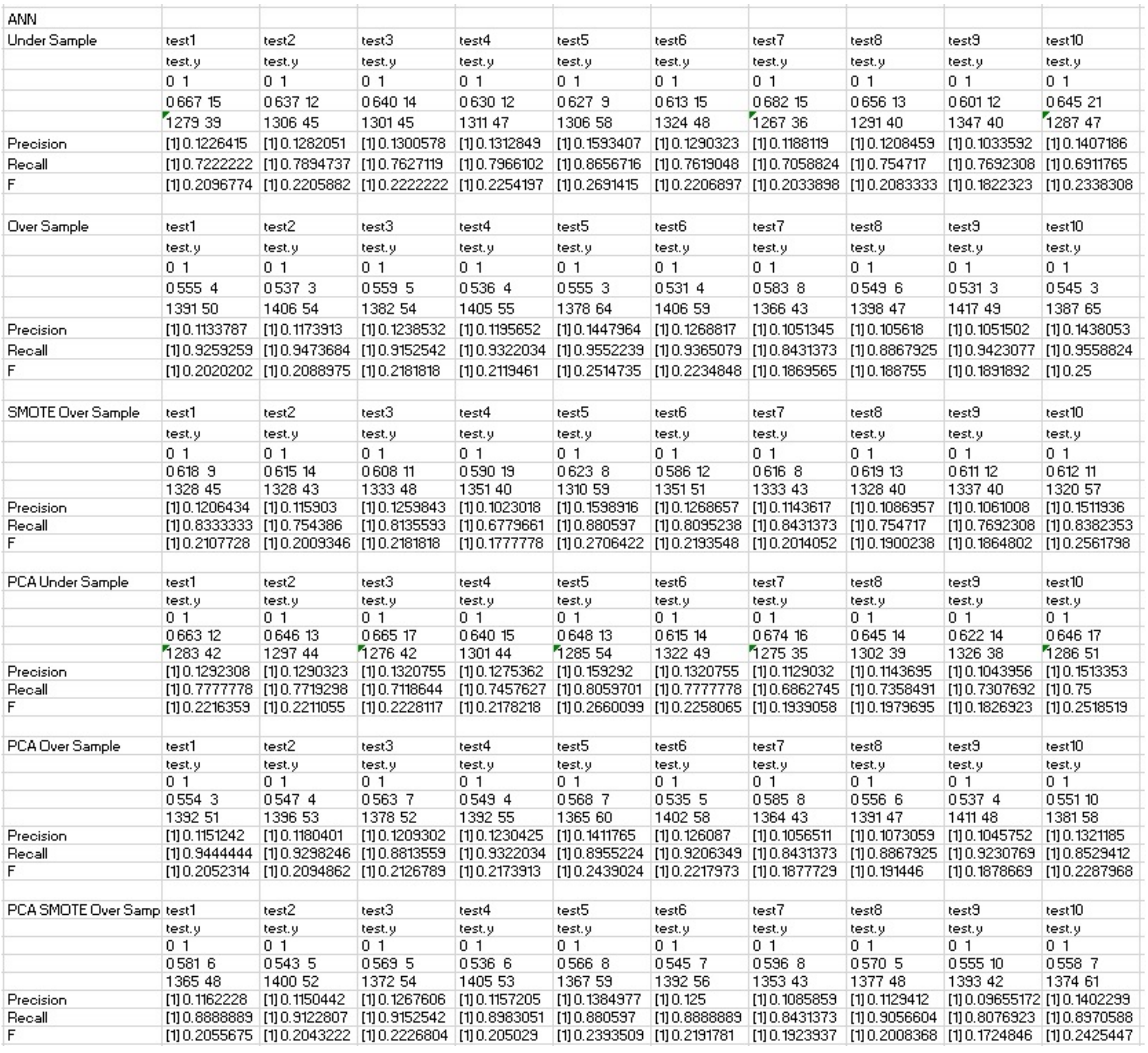
KNN	seed(4011)		recall	F_score	F_mean
over_sample_PCA1	652	23	0.5741	0.1636	0.17241
	294	31			0.59887
over_sample_PCA2	613	26	0.5438	0.1483	
	330	31			
over_sample_PCA3	648	24	0.5932	0.1809	
	293	35			
over_sample_PCA4	610	27	0.5423	0.1516	
	331	32			
over_sample_PCA5	633	25	0.6268	0.2053	
	300	42			
over_sample_PCA6	644	23	0.6349	0.202	
	293	40			
over_sample_PCA7	628	17	0.6667	0.1674	
	321	34			
over_sample_PCA8	620	20	0.6226	0.1598	
	327	33			
over_sample_PCA9	622	21	0.5961	0.1515	
	326	31			
over_sample_PCA10	627	28	0.5882	0.1937	
	305	40			



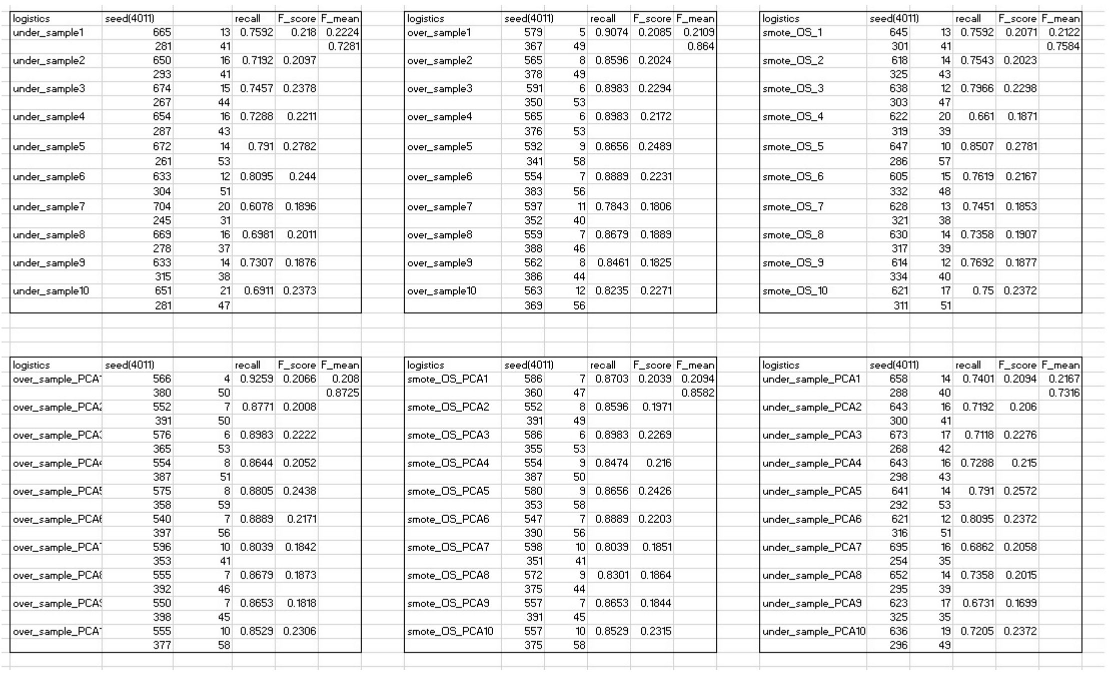
KNN	seed(4011)		recall	F_score	F_mean
smote_OS_PCA1	553	17	0.6852	0.1528	0.1699
	393	37			0.71662
smote_OS_PCA2	556	14	0.7543	0.1765	
	387	43			
smote_OS_PCA3	567	10	0.8305	0.2033	
	374	49			
smote_OS_PCA4	556	21	0.6441	0.1576	
	385	38			
smote_OS_PCA5	542	20	0.7014	0.1861	
	391	47			
smote_OS_PCA6	554	20	0.6825	0.1758	
	383	43			
smote_OS_PCA7	558	17	0.6667	0.1428	
	391	34			
smote_OS_PCA8	534	13	0.7547	0.1581	
	413	40			
smote_OS_PCA9	550	15	0.7115	0.1519	
	398	37			
smote_OS_PCA10	535	18	0.7353	0.1941	
	397	50			



KNN	seed(4011)		recall	F_score	F_mean
under_sample_PCA1	656	21	0.6111	0.1751	0.18413
	290	33			0.58565
under_sample_PCA2	665	19	0.6667	0.2037	
	278	38			
under_sample_PCA3	685	26	0.5593	0.1896	
	256	33			
under_sample_PCA4	676	30	0.4915	0.1643	
	265	29			
under_sample_PCA5	641	33	0.5074	0.173	
	292	34			
under_sample_PCA6	669	22	0.6508	0.2204	
	268	41			
under_sample_PCA7	671	22	0.5686	0.162	
	278	29			
under_sample_PCA8	646	26	0.5094	0.1417	
	301	27			
under_sample_PCA9	669	20	0.6153	0.1763	
	279	32			
under_sample_PCA1(	655	22	0.6764	0.2352	
	277	46			



ANN										
Under Sample	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 667 15	0 637 12	0 640 14	0 630 12	0 627 9	0 613 15	0 682 15	0 656 13	0 601 12	0 645 21
	1279 39	1306 45	1301 45	1311 47	1306 58	1324 48	1267 36	1291 40	1347 40	1287 47
Precision	[1] 0.1226415	[1] 0.1282051	[1] 0.1300578	[1] 0.1312849	[1] 0.1593407	[1] 0.1290323	[1] 0.1188119	[1] 0.1208459	[1] 0.1033592	[1] 0.1407186
Recall	[1] 0.7222222	[1] 0.7894737	[1] 0.7627119	[1] 0.7966102	[1] 0.8656716	[1] 0.7619048	[1] 0.7058824	[1] 0.754717	[1] 0.7692308	[1] 0.6911765
F	[1] 0.2096774	[1] 0.2205882	[1] 0.2222222	[1] 0.2254197	[1] 0.2691415	[1] 0.2206897	[1] 0.2033898	[1] 0.2083333	[1] 0.1822323	[1] 0.2338308
Over Sample	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 555 4	0 537 3	0 559 5	0 536 4	0 555 3	0 531 4	0 583 8	0 549 6	0 531 3	0 545 3
	1391 50	1406 54	1382 54	1405 55	1378 64	1406 59	1366 43	1398 47	1417 49	1387 65
Precision	[1] 0.1133787	[1] 0.1173913	[1] 0.1238532	[1] 0.1195652	[1] 0.1447964	[1] 0.1268817	[1] 0.1051345	[1] 0.105618	[1] 0.1051502	[1] 0.1438053
Recall	[1] 0.9259259	[1] 0.9473684	[1] 0.9152542	[1] 0.9322034	[1] 0.9552239	[1] 0.9365079	[1] 0.8431373	[1] 0.8867925	[1] 0.9423077	[1] 0.9558824
F	[1] 0.2020202	[1] 0.2088975	[1] 0.2181818	[1] 0.2119461	[1] 0.2514735	[1] 0.2234848	[1] 0.1869565	[1] 0.188755	[1] 0.1891892	[1] 0.25
SMOTE Over Sample	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 618 9	0 615 14	0 608 11	0 590 19	0 623 8	0 586 12	0 616 8	0 619 13	0 611 12	0 612 11
	1328 45	1328 43	1333 48	1351 40	1310 59	1351 51	1333 43	1328 40	1337 40	1320 57
Precision	[1] 0.1206434	[1] 0.115903	[1] 0.1259843	[1] 0.1023018	[1] 0.1598916	[1] 0.1268657	[1] 0.1143617	[1] 0.1086957	[1] 0.1061008	[1] 0.1511936
Recall	[1] 0.8333333	[1] 0.754386	[1] 0.8135593	[1] 0.6779661	[1] 0.880597	[1] 0.8095238	[1] 0.8431373	[1] 0.754717	[1] 0.7692308	[1] 0.8382353
F	[1] 0.2107728	[1] 0.2009346	[1] 0.2181818	[1] 0.1777778	[1] 0.2706422	[1] 0.2193548	[1] 0.2014052	[1] 0.1900238	[1] 0.1864802	[1] 0.2561798
PCA Under Sample	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 663 12	0 646 13	0 665 17	0 640 15	0 648 13	0 615 14	0 674 16	0 645 14	0 622 14	0 646 17
	1283 42	1297 44	1276 42	1301 44	1285 54	1322 49	1275 35	1302 39	1326 38	1286 51
Precision	[1] 0.1292308	[1] 0.1290323	[1] 0.1320755	[1] 0.1275362	[1] 0.159292	[1] 0.1320755	[1] 0.1129032	[1] 0.1143695	[1] 0.1043956	[1] 0.1513353
Recall	[1] 0.7777778	[1] 0.7719298	[1] 0.7118644	[1] 0.7457627	[1] 0.8059701	[1] 0.7777778	[1] 0.6862745	[1] 0.7358491	[1] 0.7307692	[1] 0.75
F	[1] 0.2216359	[1] 0.2211055	[1] 0.2228117	[1] 0.2178218	[1] 0.2660099	[1] 0.2258065	[1] 0.1939058	[1] 0.1979695	[1] 0.1826923	[1] 0.2518519
PCA Over Sample	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 554 3	0 547 4	0 563 7	0 549 4	0 568 7	0 535 5	0 585 8	0 556 6	0 537 4	0 551 10
	1392 51	1396 53	1378 52	1392 55	1365 60	1402 58	1364 43	1391 47	1411 48	1381 58
Precision	[1] 0.1151242	[1] 0.1180401	[1] 0.1209302	[1] 0.1230425	[1] 0.1411765	[1] 0.126087	[1] 0.1056511	[1] 0.1073059	[1] 0.1045752	[1] 0.1321185
Recall	[1] 0.9444444	[1] 0.9298246	[1] 0.8813559	[1] 0.9322034	[1] 0.8955224	[1] 0.9206349	[1] 0.8431373	[1] 0.8867925	[1] 0.9230769	[1] 0.8529412
F	[1] 0.2052314	[1] 0.2094862	[1] 0.2126789	[1] 0.2173913	[1] 0.2439024	[1] 0.2217973	[1] 0.1877729	[1] 0.191446	[1] 0.1878669	[1] 0.2287968
PCA SMOTE Over Samp	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y	test.y
	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
	0 581 6	0 543 5	0 569 5	0 536 6	0 566 8	0 545 7	0 596 8	0 570 5	0 555 10	0 558 7
	1365 48	1400 52	1372 54	1405 53	1367 59	1392 56	1353 43	1377 48	1393 42	1374 61
Precision	[1] 0.1162228	[1] 0.1150442	[1] 0.1267606	[1] 0.1157205	[1] 0.1384977	[1] 0.125	[1] 0.1085859	[1] 0.1129412	[1] 0.09655172	[1] 0.1402299
Recall	[1] 0.8888889	[1] 0.9122807	[1] 0.9152542	[1] 0.8983051	[1] 0.880597	[1] 0.8888889	[1] 0.8431373	[1] 0.9056604	[1] 0.8076923	[1] 0.8970588
F	[1] 0.2055675	[1] 0.2043222	[1] 0.2226804	[1] 0.205029	[1] 0.2393509	[1] 0.2191781	[1] 0.1923937	[1] 0.2008368	[1] 0.1724846	[1] 0.2425447



logistics	seed(4011)		recall	F_score	F_mean
under_sample1	665	13	0.7592	0.218	0.2224
	281	41			0.7281
under_sample2	650	16	0.7192	0.2097	
	293	41			
under_sample3	674	15	0.7457	0.2378	
	267	44			
under_sample4	654	16	0.7288	0.2211	
	287	43			
under_sample5	672	14	0.791	0.2782	
	261	53			
under_sample6	633	12	0.8095	0.244	
	304	51			
under_sample7	704	20	0.6078	0.1896	
	245	31			
under_sample8	669	16	0.6981	0.2011	
	278	37			
under_sample9	633	14	0.7307	0.1876	
	315	38			
under_sample10	651	21	0.6911	0.2373	
	281	47			

logistics	seed(4011)		recall	F_score	F_mean
over_sample1	579	5	0.9074	0.2085	0.2109
	367	49			0.864
over_sample2	565	8	0.8596	0.2024	
	378	49			
over_sample3	591	6	0.8983	0.2294	
	350	53			
over_sample4	565	6	0.8983	0.2172	
	376	53			
over_sample5	592	9	0.8656	0.2489	
	341	58			
over_sample6	554	7	0.8889	0.2231	
	383	56			
over_sample7	597	11	0.7843	0.1806	
	352	40			
over_sample8	559	7	0.8679	0.1889	
	388	46			
over_sample9	562	8	0.8461	0.1825	
	386	44			
over_sample10	563	12	0.8235	0.2271	
	369	56			

logistics	seed(4011)		recall	F_score	F_mean
smote_OS_1	645	13	0.7592	0.2071	0.2122
	301	41			0.7584
smote_OS_2	618	14	0.7543	0.2023	
	325	43			
smote_OS_3	638	12	0.7966	0.2298	
	303	47			
smote_OS_4	622	20	0.661	0.1871	
	319	39			
smote_OS_5	647	10	0.8507	0.2781	
	286	57			
smote_OS_6	605	15	0.7619	0.2167	
	332	48			
smote_OS_7	628	13	0.7451	0.1853	
	321	38			
smote_OS_8	630	14	0.7358	0.1907	
	317	39			
smote_OS_9	614	12	0.7692	0.1877	
	334	40			
smote_OS_10	621	17	0.75	0.2372	
	311	51			

logistics	seed(4011)		recall	F_score	F_mean
over_sample_PCA	566	4	0.9259	0.2066	0.208
	380	50			0.8725
over_sample_PCA	552	7	0.8771	0.2008	
	391	50			
over_sample_PCA	576	6	0.8983	0.2222	
	365	53			
over_sample_PCA	554	8	0.8644	0.2052	
	387	51			
over_sample_PCA	575	8	0.8805	0.2438	
	358	59			
over_sample_PCA	540	7	0.8889	0.2171	
	397	56			
over_sample_PCA	596	10	0.8039	0.1842	
	353	41			
over_sample_PCA	555	7	0.8679	0.1873	
	392	46			
over_sample_PCA	550	7	0.8653	0.1818	
	398	45			
over_sample_PCA	555	10	0.8529	0.2306	
	377	58			

logistics	seed(4011)		recall	F_score	F_mean
smote_OS_PCA1	586	7	0.8703	0.2039	0.2094
	360	47			0.8582
smote_OS_PCA2	552	8	0.8596	0.1971	
	391	49			
smote_OS_PCA3	586	6	0.8983	0.2269	
	355	53			
smote_OS_PCA4	554	9	0.8474	0.216	
	387	50			
smote_OS_PCA5	580	9	0.8656	0.2426	
	353	58			
smote_OS_PCA6	547	7	0.8889	0.2203	
	390	56			
smote_OS_PCA7	598	10	0.8039	0.1851	
	351	41			
smote_OS_PCA8	572	9	0.8301	0.1864	
	375	44			
smote_OS_PCA9	557	7	0.8653	0.1844	
	391	45			
smote_OS_PCA10	557	10	0.8529	0.2315	
	375	58			

logistics	seed(4011)		recall	F_score	F_mean
under_sample_PCA1	658	14	0.7401	0.2094	0.2167
	288	40			0.7316
under_sample_PCA2	643	16	0.7192	0.206	
	300	41			
under_sample_PCA3	673	17	0.7118	0.2276	
	268	42			
under_sample_PCA4	643	16	0.7288	0.215	
	298	43			
under_sample_PCA5	641	14	0.791	0.2572	
	292	53			
under_sample_PCA6	621	12	0.8095	0.2372	
	316	51			
under_sample_PCA7	695	16	0.6862	0.2058	
	254	35			
under_sample_PCA8	652	14	0.7358	0.2015	
	295	39			
under_sample_PCA9	623	17	0.6731	0.1699	
	325	35			
under_sample_PCA10	636	19	0.7205	0.2372	
	296	49			