Comparison of various classification models for making financial decisions

Vaibhav Mohan
Computer Science Department
Johns Hopkins University
Baltimore, MD 21218, USA
vmohan3@jhu.edu

Abstract

Banks are having an important place in the market economies. They decide who could get financial loans and the terms on which it should be given. Individuals and companies require bank loans in order for markets and society to function properly.

Credit scoring algorithms are the tools used by banks in determining whether the loan should be approved for individual/company or not. In this project, I implemented various classification algorithms for predicting whether somebody will experience financial distress in the next two years by looking on various parameters like monthly income, age, number of open credit lines and loans etc. A comparison of accuracy of implemented models is also done.

The model can be used by borrowers to help them in making best financial decisions.

1 Introduction

Individuals/companies apply for loans so as to keep their business going. Banks play crucial role in providing individuals/companies with funds so that the market and society could function properly. But then, not all individuals/company's loan application should be approved. In order to give a decision on a loan application, the bank looks on credit history of an individual/company. The parameters include monthly income, age, number of open credit lines and other approved loans for that individual. These parameters are used in predicting whether a person would be suitable for giving loan or not. This functionality can be achieved by using credit scoring algorithms to predict whether somebody will experience financial distress in the next two years.

In this paper, I have implemented various classifiers for predicting the aforementioned decision. The algorithms implemented in this paper are Perceptron predictor, Winnow predictor, and an ensemble training algorithm that uses combination of k-nearest neighbors and neural networks algorithm with instance bagging for predicting the labels. The details of the above algorithms are given in subsequent sections in this paper.

I have also done some preprocessing of data as some of the instances had noisy data while others had some missing data. Its details are given in the subsequent sections. This paper also does the comparison of accuracies obtained by each classification algorithm.

2 Classification

2.1 Machine Learning techniques used

The data is preprocessed so that noisy data and missing data in some instances can be handled. The machine learning techniques that are used in predicting the labels are Perceptron predictor, Winnow predictor, and an ensemble training algorithm that uses combination of k-nearest neighbors and neural networks algorithm with instance bagging. Details on how data is cleaned, and why a

given algorithm is implemented and used are described in the sections given below.

2.2 Data

The data used in this project is obtained from the competition titled "Give me some credit" in kaggle.com. The data file contains various parameters such as Monthly Income, Number of Dependents, age, number of open credit lines and loans etc. It is stored in csv format**.** Since the test data was not having labels, therefore in order to do training and predictions, I separated the train data in two groups. The train data contained 150,000 instances. I used 80% data for training the classifiers and remaining 20% for testing. The algorithms were also used to predict test data but since labels were not available for that, accuracy for it cannot be calculated. The data contains various parameters. Each parameter is described briefly below along with how it is processed to deal with missing data.

a) **Instance Number (column 1):** This contains the instance number data. It is left as is.

b) **SeriousDlqin2yrs (column 2):** This is of binary type. Our algorithm is used to predict this. This depicts whether a person experienced 90 days past due delinquency or worse.

c) **RevolvingUtilizationOfUnsecuredLines (column 3):** Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits. This is in percentage. The missing data was replaced with 0 and if any data in this column was greater than 100, it was replaced with data%100.

d) **Age (column 4):** Contains the age of the borrower in years. It is of integer type. This column didn't contain any missing data. However the data in this column which was greater than 100 was replaced with data%100.

e) **NumberOfTime30-59DaysPastDueNot Worse (column 5):** This column contains number of times borrower has been 30-59 days past due but no worse in the last 2 years. The missing data was replaced with 0.

f) **DebtRatio (column 6):** This field contains data in percentage form. It is obtained by dividing the sum of monthly debt payments, alimony, living costs with monthly gross income. This column didn't contain any missing data.

g) **MonthlyIncome (column 7):** This column contained the information about the monthly income of an individual. The missing data was replaced by the average of monthly income of all the instances.

h) **NumberOfOpenCreditLinesAndLoans (column 8):** This column contained information about the number of open loans such as car loans, house loans and lines of credit (ex. Credit card). The missing data was replaced with a zero.

i) **NumberOfTimes90DaysLate (column 9):** This column had information about the number of times an individual was late by 90 days or more in paying their bills. The missing value was replaced with 0.

j) **NumberRealEstateLoansOrLines (column 10):** This column contained information about number of mortgage and real estate loans including home equity lines of credit an individual have taken. The missing value was again replaced with zero.

k) **NumberOfTime60-89DaysPastDueNot Worse (column 11):** This field contains the information about the number of times borrower has been 60-89 days past due but no worse in the last 2 years. The missing values in this column were again replaced with a zero.

l) **NumberOfDependents (column 12):** This column contained information about the number of dependents in the family excluding themselves. The missing values in this column were again replaced with a zero.

2.3 Perceptron Predictor

The perceptron predictor was used initially to see if it can learn the data properly and that implementing neural networks for the given data would be a wise choice or not. Also according to Daniel et. al, the perceptron classifier is well suited for linearly separable data. Hence I implemented this first to see if we can gain good accuracy in prediction.

The perceptron predictor is a mistake driven algorithm i.e. updates to the weight vector of the per-

ceptron is made only when an example is misclassified. The algorithm for perceptron predictor is given in figure 1.

function PERCEPTRON-PREDICTOR
  w=0
  $\beta = 0.0$
  $\eta = 0.1$
  $num\_iterations = 10$
  for $(k = 1 \rightarrow num\_iterations)$ **do**
    for $(i = 1 \rightarrow n)$ **do**
      $receive - \mathbf{x_i}$
      $compute - \mathbf{w}.\mathbf{x_i}$
      if $(\mathbf{w}.\mathbf{x_i} \geq \beta)$ **then**
        $\hat{y}_i = 1$
      else
        $\hat{y}_i = -1$
      end if
      if $(\hat{y}_i \neq y_i)$ **then**
        $\mathbf{w}' = \mathbf{w} + \eta.y_i.\mathbf{x_i}$
      end if
    end for

Figure 1. Perceptron Predictor Algorithm

In this algorithm, the weight vector ***w*** is initialized to **0**. The threshold ***β*** is initialized to 0.0, learning rate ***η*** is assigned to 0.1, and number of training iterations is set to 10. This algorithm runs as many number of times as number of iterations is specified. For each example ***i***, the algorithm receives the feature vector x_i and computes its dot product with the weight vector ***w***. Now if this dot product is greater than the threshold, the predicted label is 1 otherwise the predicted label is -1. The algorithm then checks whether the predicted label matches the original label. If the predicted label matches the original label, the weight vector is kept as is. If the predicted label does not matches the original label, then the weight vector is updated according to the formula:

*New Weight Vector = Old Weight Vector + learning rate*original label*feature vector*

This algorithm is implemented by me for the given datasets and its accuracies by varying number of iterations on test data is recorded and listed in the results section.

2.4 Winnow Predictor

This algorithm is similar to perceptron predictor. The only changes between this algorithm and perceptron are the way this initializes its parameters and its update function for the weight vector. The Winnow predictor algorithm is given in figure 2.

function WINNOW-PREDICTOR
  w=1
  $\beta = n/2$
  $\eta = 0.1$
  $num\_iterations = 10$
  for $(k = 1 \rightarrow num\_iterations)$ **do**
    for $(i = 1 \rightarrow n)$ **do**
      $receive - \mathbf{x_i}$
      $compute - \mathbf{w}.\mathbf{x_i}$
      if $(\mathbf{w}.\mathbf{x_i} \geq \beta)$ **then**
        $\hat{y}_i = 1$
      else
        $\hat{y}_i = -1$
      end if
      if $(\hat{y}_i \neq y_i)$ **then**
        $\mathbf{w}' = \mathbf{w} * \eta^{y_i.sign(\mathbf{x_i})}$
      end if
    end for

Figure 2. Winnow Predictor Algorithm

In this algorithm, the weight vector ***w*** is initialized to 1. The threshold ***β*** is initialized to n/2 where 'n' is the number of instances, learning rate ***η*** is assigned to 0.1, and number of training iterations is set to 10. For each example ***i***, the algorithm receives the feature vector x_i and computes its dot product with the weight vector ***w***. Now if this dot product is greater than the threshold, the predicted label is 1 otherwise the predicted label is -1. The algorithm then checks whether the predicted label matches the original label. If the predicted label matches the original label, the weight vector is kept as is. If the predicted label does not matches the original label, then the weight vector is updated according to the formula:

*New Weight Vector = Old Weight Vector * (learning rate)* $^{original\ label * sign\ (feature\ vector)}$

Also if the new weight is greater than some fix quantity μ = 1.0e6, then the new weight is made equal to μ. Again this algorithm is also implemented and its performance is shown in the results section.

2.5 Ensemble Learning Algorithm – Neural Networks combined with *k*-nearest neighbors algorithm

Although the *Perceptron Predictor* algorithm seemed promising, the single layer perceptrons are only capable of learning linear data. For ex. Only one perceptron cannot be used to represent the XOR function. According to Chuanyi et. al, in order to obtain a classification system that have good generalization performance as well as efficiency in space and time, a learning method based on combinations of weak classifiers can be used. Weak classifiers are linear classifiers that can do just a little better than making random guesses. Also according to Igor et. al, we can achieve good performance if we use mean of labels of k-nearest neighbors while training the data instead of simply taking the label of example for which the label is misclassified.

Keeping all these points in mind, I came up with an algorithm that combined the k-nearest neighbor algorithm with neural networks. Neural networks are combination of perceptron. The *Perceptron Predictor algorithm* was preferred over *Winnow Predictor algorithm* for implementing the subclassifiers in ensemble learning algorithm because the former performed better than the latter. The algorithm implemented here is a two layer feed forward network. The input layer consists of training data. The next layer is called *hidden layer* which contains hidden nodes. The values from these hidden nodes are then combined to produce a final output. We have a parameter 'K' in the implementation that depicts the number of perceptron used in hidden layer. The algorithm also have a parameter 'k_nn' that defines how many instances are used in obtaining the mean value of the label using k-nearest neighbors algorithm. Here each perceptron 'k' has its own set of weight vectors $\boldsymbol{w_k}$. Each perceptron also have a weight $\boldsymbol{\mu_k}$ associated with it which is used when we combine the output from it. The prediction of label using this algorithm is given by the following equation:

$$\hat{y} = h\left(\sum_{k=1}^{K} \mu_k g(\mathbf{w_k}.\mathbf{x})\right)$$

Equation 1. Prediction function for Ensemble predictor.

Where ***h(z)*** is equal to 1 if z is greater than or equal to zero and zero otherwise, and ***g(z)*** is given by:

$$g(z) = \frac{z}{\sqrt{1+z^2}}$$

Here the function 'g' is an activation function which transforms non-linear inputs to output which is usually binary. Neural networks use sigmoid functions to approximate binary output.

The training of the neural networks is done using the 'k' perceptrons which we have in the hidden layer. Each perceptron is trained using aforementioned algorithm with the following two changes:

1) For the $\boldsymbol{k^{th}}$ sub classifier, we exclude every k^{th} instance from that perceptron's training data i.e. if i%K=k, then that instance is not included in training that sub classifier. This approach is known as instance bagging.

2) During the training of the sub classifier, if we predict a wrong label, then we don't use the original label from the same instance in learning. Instead, we use k-nearest neighbor algorithm to find k-nearest neighbors using Euclidean distance and take the mean of the labels of 'k_nn' number of instances. Now if the mean is greater than or equal to 0.5, we use original label as '1'. Otherwise we use '0' as the original label.

Now for learning the weights $\boldsymbol{\mu_k}$ associated with each sub classifier, we use the following equation:

$$E(\mu) = \sum_{i \in M} y_i \left(\sum_{k=1}^{K} \mu_k g(\mathbf{w_k}.\mathbf{x_i})\right)$$

Equation 2. Function for learning weights $\boldsymbol{\mu_k}$.

Where 'M' is the set of misclassified examples. The algorithm for updating the $\boldsymbol{\mu_k}$ associated with each sub classifier is given in figure 3.

Thus the algorithm for training the ensemble predictor is given in figure 4.

if $(y_i \neq \hat{y}_i \&\& y_i == 1)$ **then**
  for $(k = 1 \rightarrow K)$ **do**
    $\mu_k^{(new)} = \mu_k + \eta g(\mathbf{w_k}.\mathbf{x_i})$
  end for
else if $(y_i \neq \hat{y}_i \&\& y_i == 0)$ **then**
  for $(k = 1 \rightarrow K)$ **do**
    $\mu_k^{(new)} = \mu_k - \eta g(\mathbf{w_k}.\mathbf{x_i})$
  end for
else
  $\mu_k^{(new)} = \mu_k$
end if

Figure 3. Update algorithm for $\boldsymbol{\mu_k}$ of sub classifier 'k'.

function ENSEMBLE-PREDICTOR
  Train K Perceptron subclassifier using the method described above.
  $\mu = 0$
  for $(t = 1 \rightarrow training\_iterations(t))$ **do**
    for $(i = 1 \rightarrow num\_instances)$ **do**
      $receive - example - \mathbf{x_i}$
      predict label y_i acc.to eqn. 2
      If $y_i \neq \hat{y}_i$, then make an update to μ acc. to algo. in fig. 3
    end for
  end for
end function

Figure 4. Ensemble Predictor Algorithm.

The problem faced in implementing the above proposed algorithm was that *k-nearest neighbors* algorithm runs in $O(n^2)$. Hence the algorithm was taking a lot of time in training. Therefore in order minimize the time, one more parameter named *'num_instances_k_nn'* was introduced that limited the number of instances to be used for finding the 'k' nearest neighbor of the given instance. The number of instances for training *k-nearest neighbors* are chosen at random from the given instances and then finally *'k_nn'* number of instances were taken to obtain the mean for finding the label.

One other problem faced in implementing this algorithm was to decide whether the output from *k-nearest neighbor* should be fed into the *neural network* or the reverse must be done. So I experimented with both the models and found that former yielded better accuracy in predicting data as compared to latter.

3 Experimental framework and results

All the implementations were tested on system running on Intel core-i7 3610QM @ 2.3 GHz. The operating system used was Windows 7. Implementations were done using JAVA v1.7 using eclipse as IDE. The comparison of accuracies of different algorithms were recorded and compared.

3.1 Perceptron Predictor

The perceptron predictor was tested by varying the number of iterations in training and also by varying its learning rate. The number of instances in training data was 120,000 and that in test data was 30,000. The results are shown in table 1.

By seeing the results, we can say that the classifier is able to learn the data properly and give predictions with good accuracy.

Number of iterations	Learning rate (α)	Accuracy
10	0.01	93.19 %
50	0.01	93.19 %
100	0.01	93.17 %
200	0.01	93.17 %
500	0.01	93.17%
10	0.1	93.19 %
50	0.1	93.19 %
100	0.1	93.17 %
200	0.1	93.18 %
500	0.1	93.18 %

Table 1. Results for Perceptron Predictor Algorithm

From the results given above, we can say that increase in number of iterations decreases the accuracy of predictions while changing learning rate have almost no effect on predictions.

3.2 Winnow Predictor

The Winnow Predictor was also tested the same way Perceptron Predictor was tested. The results for the tests are shown in table 2.

It is clear from table 1 that the algorithm is able to learn the data and predict it with high accuracy. It can also be observed that increasing the number of iterations does not change the accuracy of predic-

tion while increasing the learning rate increases the prediction accuracy.

Number of iterations	Learning rate (α)	Accuracy
10	1.1	92.09 %
50	1.1	92.1 %
100	1.1	92.09 %
200	1.1	92.1 %
500	1.1	92.1%
10	1.5	92.98 %
50	1.5	92.97 %
100	1.5	92.97 %
200	1.5	92.98 %
500	1.5	92.97 %

Table 2. Results for Winnow Predictor Algorithm

3.3 Ensemble Predictor

The ensemble algorithm had various different parameters such as number of instances for k-nearest neighbors, 'k_nn' in k-nearest neighbors, learning rate (η), the number of perceptrons to be used (K), number of iterations (num_iter) etc. The parameters were varied and tested on the same datasets with which the other two algorithms were tested. The parameters in the subclassifiers (perceptron) were kept constant while varying the parameters of ensemble learning algorithm. The Learning rate (α) of the subclassifier was set to 0.01 and the number of iterations of subclassifier was set to 10. This is because the given settings yielded best accuracies in predicting the data as we can see from table 1.

The test results for Ensemble Predictor by varying its various parameters are given in table 3.

k_nn	K	η	num_iter	Accuracy
3	5	0.1	5	93.22 %
3	5	0.2	10	93.23 %
3	5	0.5	20	93.22 %
4	4	0.1	5	93.22 %
4	4	0.2	10	93.25 %
4	4	0.5	20	93.22 %
5	3	0.1	5	93.22 %
5	3	0.2	10	93.22 %
5	3	0.5	20	93.21 %

Table 3. Results for Ensemble Predictor Algorithm

From the above results, we can see that there is a very small performance gain in ensemble algorithm as compared to others. Also increase in learning rate and number of iterations increases accuracy minutely. The accuracies obtained is almost equivalent in all the cases.

3.4 Comparison of models

From the results given in table 1, 2, and 3; we can conclude that Perceptron Predictor algorithm performs better than Winnow Predictor algorithm and hence the subclassifier used in implementing the ensemble predictor algorithm is perceptron predictor. It can also be seen that ensemble predictor algorithm performs better than any other model and is able to predict the data with good accuracy.

4 Conclusion

I conclude that all the models describe the data quite well. It can also be concluded by seeing the results that Ensemble Predictor performs better than any other model on the given datasets. I believe that combination of the KNN and Neural Networks are interesting and could be applied in other types of datasets too.

5 Future work

Replacement of missing or unreasonable values could be performed by a more suitable procedure, such as maximum-likelihood based methods.

Another interesting approach in learning the data could be feeding the logistic model with results from other models, such as SVM or neural networks.

6 Comparison to proposal

According to the original proposal, all the tasks were implemented in this project except the logistic regression which I mentioned that I would like to achieve. I found that logistic regression algorithms was already implemented during the assignments. Also I had mentioned that I will implement Neural Networks using instance bagging as well as feature bagging. But again since this was already implemented in assignments, so I

came up with a combination of k-nearest neighbors and neural networks algorithm in the final implementation. In addition to the things mentioned in proposal, I also implemented data cleaning techniques for handling missing data as well as noisy data. Overall, everything was implemented up to my expectation in this project.

References

Bishop, C.M. 2006. *Pattern Recognition and Machine Learning.* Springer Science.

Mitchell, Tom 1997. *Machine Learning.* McGraw-Hill.

Zurada, J. M. 1992. *Introduction to artificial neural systems.* St. Paul: West.

Daniel A. Jimenez, Calvin Lin. 2001. Dynamic Branch Prediction with Perceptrons. *Association for Computing Machinery. Proceeding HPCA '01 Proceedings of the 7th International Symposium on High-Performance Computer Architecture Page 197.*

Chuanyi Ji, Sheng Ma. 1997. Combinations of Weak Classifiers. *IEEE transactions on neural networks, Vol. 8, No. 1.*

Igor V. Tetko. Associative Neural Networks. *Institut de Physiologie, Université de Lausann.*

Maytal Saar-Tsechansky, Foster Provost. 2007. Handling Missing Values when Applying Classification Models.

Mark Dredze. 2012. *Class Assignments.* Johns Hopkins University.