CSE 575 Class Project Report

Weather Forecasting

By

Balaji Sankar, 1209346459, bsankar@asu.edu

Karthik Subramanian, 1209360876, ksubra19 @asu.edu

Aarti Rao Manjeshwar, 1209163211, amanjesh@asu.edu

Shwetha Narayanan, 1209166097, snaray36@asu.edu

Ramanathan Nachiappan, 1210822532, rnachiap@asu.edu

Contents

[1 Introduction 1](#_Toc480647172)

[1.1 Project Background 1](#_Toc480647173)

[1.2 Problem Description 1](#_Toc480647174)

[1.2.1 Class Imbalance Problem 2](#_Toc480647175)

[1.2.2 Feature Selection for High Dimensional Data 2](#_Toc480647176)

[1.2.3 Categorical Variables for Classification 3](#_Toc480647177)

[1.2.4 Sampling methods to create simpler models 3](#_Toc480647178)

[1.3 Project Plan 3](#_Toc480647179)

[2 General Methodology 3](#_Toc480647180)

[2.1 Pre-Processing 4](#_Toc480647181)

[2.1.1 Categorical variables 4](#_Toc480647182)

[2.1.2 Sampling 5](#_Toc480647183)

[2.2 Learning Phase 5](#_Toc480647184)

[2.3 Performance and Cross Validation 5](#_Toc480647185)

[3 Feature Selection Process 6](#_Toc480647186)

[3.1 Stepwise/Adaptive Lasso Approach 6](#_Toc480647187)

[3.2 Information Gain Approach 7](#_Toc480647188)

[4 Modeling Analysis 9](#_Toc480647189)

[4.1 Logistic Regression 9](#_Toc480647190)

[4.1.1 Approach 10](#_Toc480647191)

[4.1.2 Results 10](#_Toc480647192)

[4.2 Training the Logistic Regression model using Gradient Methods 11](#_Toc480647193)

[4.2.1 Stochastic Gradient Methods 11](#_Toc480647194)

[4.3 Support Vector Machine(SVM) 12](#_Toc480647195)

[4.3.1 Approach 13](#_Toc480647196)

[4.3.2 Results 13](#_Toc480647197)

[4.4 Random Forests 14](#_Toc480647198)

[4.4.1 Approach 15](#_Toc480647199)

[4.4.2 Results 16](#_Toc480647200)

[4.5 Decision Trees with Boosting 16](#_Toc480647201)

[4.5.1 Approach 16](#_Toc480647202)

[4.5.2 Results 17](#_Toc480647203)

[4.6 Neural Network 17](#_Toc480647204)

[4.6.1 Approach 18](#_Toc480647205)

[4.6.2 Results 19](#_Toc480647206)

[4.7 Overall Performance 21](#_Toc480647207)

[5 Conclusions and Final Discussion 22](#_Toc480647208)

[6 Team Member Contribution 24](#_Toc480647209)

[Bibliography 26](#_Toc480647210)

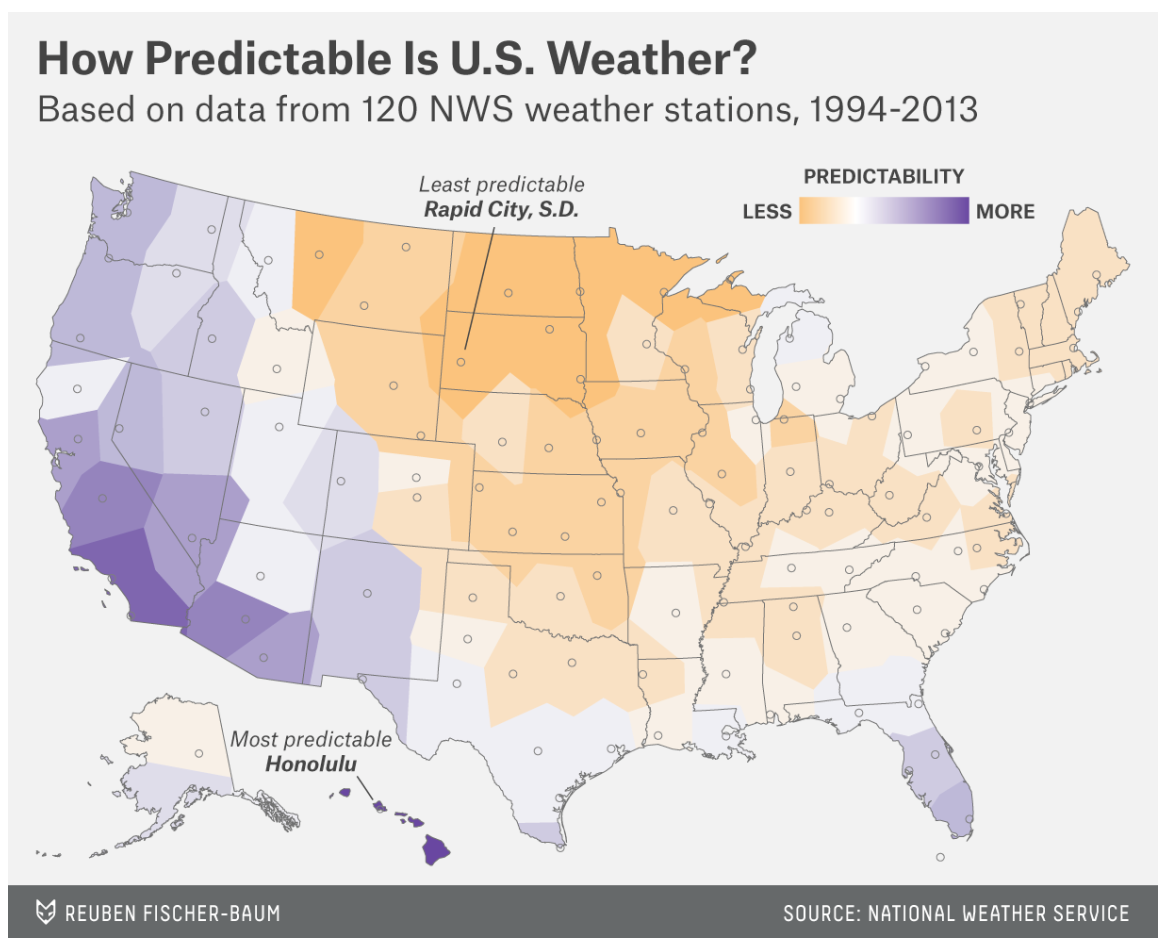
# Introduction

## Project Background

Our project “Weather Forecasting” deals with the process of predicting the temperature of a location with respect to the change in time. The present model of weather predictions is performed either by the traditional method of using information of atmospheric physics or by using machine learning models. The traditional method of using atmospheric physics involves sampling of the present state of the atmosphere and future state is calculated using a series of thermodynamic equations. The application of machine learning in this realm irked us in selecting this project which will give us insights to the practical application of prediction models applied to time series data which is humungous in nature. Building the models with such huge amounts of data is a challenge in terms of managing the computing resources which was another reason to select this project. Above all choosing weather prediction boils down to the factual influence in social movement and behavioral patterns such as travel patterns, buying patterns ranging from food to apparels.

## Problem Description

How predictable is the weather of the United States of America? And almost every member of the US will complain towards the unpredictability of weather. This forms the basis of this project. The project aims at understanding the various machine learning models applied to time series data on a broader spectrum and generating analysis from the results by narrowing it down to the realm of weather forecasting. The notion of our project is to predict the weather for a given hour using the time series data sampled on an hourly basis. This data includes both temperature and other weather related features which will be explained in detail in the data set section, and we intended to apply the models for univariate time series and multivariate time series patterns to put forth our predictions.



## Project Plan

In the upcoming sections, we will be discussing the timeline undertaken and course of project’s developments. Section 2 discusses about the dataset and feature selection. Section 3 explains the various approaches that we followed and results of each experiment. Section 4 consolidates the overall results and inferences of the various models used in the application. Eventually, section 5 concludes with the conclusions, takeaways and future prospects of this project.

# Data Set

## Pre-Processing

### Categorical variables

### Sampling

## Learning Phase

## Performance and Cross Validation

# Feature Selection Process

## Stepwise/Adaptive Lasso Approach

## Information Gain Approach

# Modeling Analysis

## Baseline Prediction

For any time series problem, baseline prediction model is essential to establish a benchmark of comparison in order to evaluate other models on our problem. Since we wanted to obtain a benchmark for our problem we considered using a naïve forecasting algorithm which is the persistent algorithm. The persistent algorithm which is equivalent to the *Zero Rule* algorithm used in supervised machine learning, uses the previous time step observation to predict the outcome at the next time step of the time series. The reason for opting to this method is because the baseline prediction method should be simple with no training or intelligence or background domain information and it should be trivial and deterministic in predicting the outcome.

### Approach

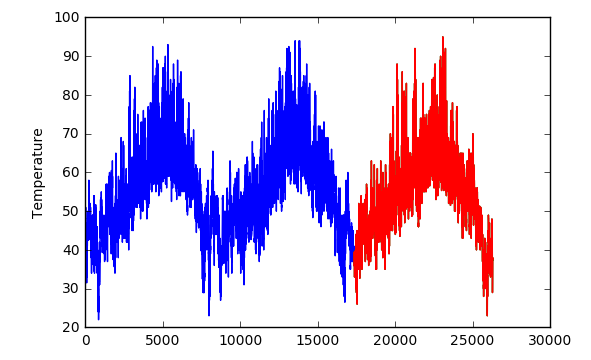
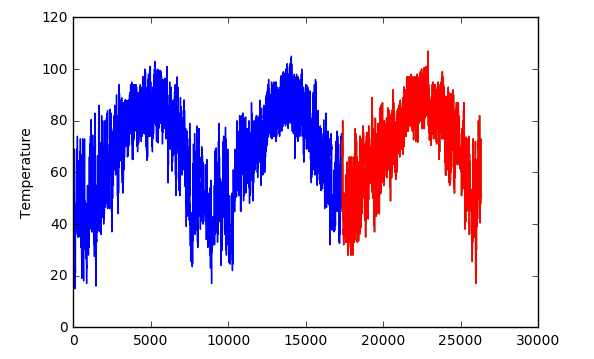
We implemented this model in Python. We first extracted the *Dry Bulb temperature* which is the actual temperature that we want to predict as a Series variable using the Pandas library, since this will be mapped in line with time thus forming a univariate time series. This univariate time series is then converted into a supervised learning problem by considering the observation in time step t-1 as the input and predicting the output at time step t+1. This is computed by creating a lagged representation. Considering this as the dataset, we split it in the ratio 2:1 for training and testing set. We defined the persistent model which returns the value provided as input, which states that when the t-1value is given as input, the next value will be provided as prediction thus building a model from the lagged representation.

### Results

This model uses naïve persistence and hence it does not require any learning. We perform a walk forward validation and use the lagged version to obtain the predictions. From the predictions, we calculated the mean squared error MSE by the formulae where represents the predicted value and represents the actual value.

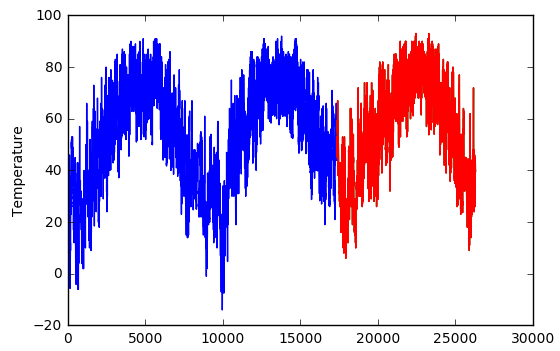
|  |  |
| --- | --- |
| **City** | **MSE** |
| Seattle | 3.17 |
| Dallas | 4.6 |
| Cincinnati | 6.67 |

This model is simple and deterministic since this learns nothing from the domain knowledge or the data but uses the naïve approach with a lagged dataset. These MSE values will be used thus be used as a benchmark and any machine learning model that is used for the weather forecasting will be compared with these values since these were evaluated with a model without any learning mechanisms. If the MSE value for any model is greater than these values, it will either be discarded or needs to be fixed to obtain better results.

Seattle

Dallas



Cincinnati

## AR Modeling

### Stochastic Gradient Methods

As mentioned earlier, one of the problems with our dataset is its actual size. Thus, we searched for a stochastic-based approach to improve the convergence time of our model through the range of available R-packages but were unable to find a satisfactory algorithm that could be applied. Therefore, we searched for an existing gradient descent method that could be adapted to our dataset and could be modified to consider a stochastic component. In this case, we were able to find an existing GitHub code that used a basic gradient descent method [13], which we then modified to work with our own data set. These modifications are clearly highlighted in our coding set**Error! Reference source not found.**. To this framework, we added the stochastic component and used the final outputted s to estimate its testing error and overall performance.

Importantly, one of the advantages with working with the base components of the gradient descent methods was that we were able to observe firsthand the inner workings of this technique. For example, in order to incorporate the stochastic component to the model, it was necessary to understand all parts of the working algorithm. After converting our categorical data set into a dummy variable representation, a stochastic version of the gradient decent method was applied. An important aspect of this phase was that we were able to observe the impact that the different parameters have on the convergence patterns of the s. For example, using a sample size of 500 per iteration (as part of the stochastic version), with , the theta values failed to converge and started to oscillate back and forth. However, when we decrease the value of , the convergence pattern looked totally different and converged relatively quickly to its final value (Figure 6).

|  |  |
| --- | --- |
|  |  |

Figure 2: Convergence of Thetas per Iteration (Stochastic Gradient Method)

Finally, based on the matrix multiplication between the outputted theta values and the values in our test set, we obtain the predicted set and were able to obtain a relatively satisfactory performance (Table 4).

Table 4: Predictive Performance from Stochastic Gradient Methods

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 57 | 4560 |
| 1 | 105 | 14104 |

As one can observe, the accuracy measure of 75.4%, and an F1 score of 85.8 which is comparable against the results obtained using regularized logistic regression outputted from the *“glm”* function in R.

## Support Vector Machine(SVM)

Support Vector Machine, based on the given learning sample, constructs a model, which is something like a hyperplane or even a set of hyperplanes in a higher dimensional space. These models; hyperplanes are used for classification, regression and related tasks. Our goal is to find a good hyperplane which has the largest functional margin, i.e. largest distance from the nearest training sample point. The support vector machine aims to maximize this margin, because intuitively the larger the margin the lower the error of classification.

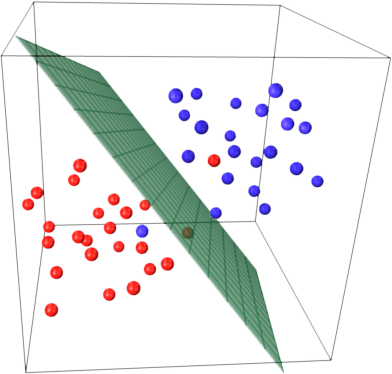


Figure 3: SVM hyperplane

As evident from the figure sometime it is not possible to completely separate two samples using a linear model, in these cases we use a non-linear kernels introduced by Boser [14][15], which maps the original problem defined in finite dimensional space into higher dimensions, making the separation easier to establish. The extended hyperplane is best defined as a set of points whose dot product with a vector (that defines the hyperplane) is constant. The vector defining the hyperplane is chosen based on *αi* which is a representation of *xi*, the feature vector. This mapping is based on the relation , where is the kernel function, which is larger the closer *x* is to *y*. Support Vector Machine also gives another important flexibility component, where the margin of classification can be either hard, where there is no room for any error, or soft [16], where there is some room for a sample portion of test samples to be wrongly classified. Figure 8 illustrates the difference between a soft margin and hard margin Support Vector Networks which is a version of SVM.

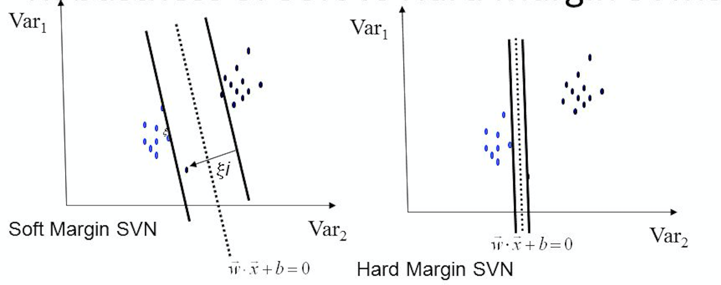


Figure 4: Soft and Hard Margin SVN

Our particular problem, we observed, is tailor made for SVM. Binary classification, with a large number of sparse observations is a condition the SVM thrives in. We believed that if we get a good support vector space, we can potentially speed up our classification process, especially with the anonymous dataset that we had obtained.

### Approach

For our particular problem we chose to go with two variants of support vectors, one was with linear kernel and the other one with radial kernel, with both of them being a soft margin classifier. The linear kernel was run to analyze the performance of the given data set for a linear classifier, while the radial provided a base to analyze the performance on a non-linear classification model. We choose the soft margin type as there are high chances that in a data set this big there may be many instance of misclassification in the training sample.

We use the implementation of SVM provided in the e1071 package of R, after doing many comparisons between the various available implementation and ease of use, thanks to Hornik [17]. This provides us with a flexible yet stable implementation.

*svm(target~., Train, kernel="radial", type="C-classification", cost=C, gamma=γ)*

This is the basic call to the svm function provided in the e1071 package to train the model on the given training set. We change the type between *“linear”* and *“radial”* as required. The cost and the gamma value determine the decision boundary of the classifier. The SVM implementation in this particular package uses the *coordinate descent* [18]approach to get a near optimal coefficient vector.

### Results

We ran a N-fold cross validation on the training set to determine the best possible soft margin parameters. We cross validated against 36 possible combinations of cost and gamma parameters to obtain the best result. This was obtained for the particular value of   
*C = 100, γ = 0.01* with a Radial kernel. When we ran this configuration for a particular data set we got a very impressive result as depicted in Table 5. The rows represent the actual class while the columns represent the predicted class.

Table 5: SVM sample result

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 617 | 1567 |
| 1 | 1290 | 5672 |

Based on the result obtained we get an accuracy of 68.7% and Precision of 78.4, with recall of 81.47 and F1 Score of 79.87.

We made some interesting observations while we performed the classification using SVM. Even with high cost value (greater than 10) and low gamma values (less than 0.01), we get a large number of support vectors, almost 94% of the observations were touted as support vectors. We performed closer inspection and proposed that the the reason for this phenomenon may be the fact that the observations are very closely spaced. To test our hypothesis, we decided to test the performance on a linear kernel with a lower cost and gamma value, what was surprising was that the number of support vectors did not decrease by much, it dropped to 89%, but we took a huge hit on the overall accuracy and precision, going down from 65.43 to 61.24 and from 75.62 to 71.72 respectively. After this round of results, we came to the conclusion that the classes were randomly spaced, for a linear kernel to correctly classify with high accuracy.

Figure 5: Accuracy vs Cost Value for different Gamma Values

Next step in our process was to move to the radial kernel, which in theory should give us a better result. Our problem was compounded by the fact that the data set is anonymous. This allowed us no room to maneuver. We selected only those features that gave the maximum information gain, and even then faced the problem of categorical variables, with multitude of possible values. The radial kernel helped improve the result. We got a very good result as show above, with a high cost value and a low gamma value, but even then we had an issue with the number of support vectors, around 82%. Our final conclusion was that SVM is not a great method for this particular data set with its skewed distribution where one class made up 75% of the recorded observation, even methods like Over sampling or under sampling really don’t help as outlined by Japkowicz [19].

## Random Forests

Decision tree methods classify by constructing a tree, it sorts the data from root to the leaf nodes, thus providing a classification. Each node in the decision tree corresponds to an attribute of the dataset and it represents one of all possible values of that attribute. Decision tree creates a classification rule by forming the disjunctions of conjunctions of constraints on the attribute values.

Random forests is a technique similar to the decision tree, where we construct many classification trees. Each classification tree gives a classification result which is termed as votes for the classification. The algorithm chooses the classification which has the most number of votes. At each node some *m* random variables are selected out of the training data and the best split on these *m* is used to split the nodes [20]. The error rate of the random forest depends on the correlation between any two trees in the forest and the strength of the individual tree. The selection of parameter *m* (split candidates) influences the error rate of the decision trees hugely. A higher value for *m* increases the correlation between the trees and the strength of the individual tree. It is important to choose the optimal value of *m* to achieve the better performance.

Random forests algorithm performs well in case of binary classification problems where we need to classify the data into two discrete classes. It works well with the data that consists of discrete values that are represented by attribute value pairs. Since Random forests algorithm constructs many decision trees by splitting the nodes based on random variables, it usually performs well on large datasets. Also since it uses all the variables present in the dataset, it minimizes the loss of information and increases the classification performance [21]. The random forest constructed can be stored and applied on new data for prediction. In some of the problems, the dataset will have large number of missing values and it acts as a major obstacle for many classification algorithms. But Random forests generally perform well on the datasets that contain missing values.

In our problem the dataset we have is a large dataset with 114,000 observations with many missing values. Many of the variables in the dataset are discrete, categorical variables, also the data set has a large number of variables (131). Random forests algorithm would be a better choice to fit the data. The classification problem we have is also binary where we need to classify the data as yes or no (0 or 1). The dataset has huge class imbalance as well, where around 70% of the data is towards the class 1, which becomes a challenging task to train the data set without overfitting. All these reasons contributed to our choice of Random forest as a classification approach.

### Approach

We have used the *“randomForests”* package available in R to implement the Random forest algorithm. We have implemented the K fold cross validation to determine the optimal value for the number of split candidates (*mtry*). The training data is split into three folds and we trained with two folds & cross validated with the other fold in each iteration. Then the average accuracy is used to determine the performance for a particular value of mtry. We tried for five values of mtry which are 5,10,25,50,100. The number of trees we used for training is 500.

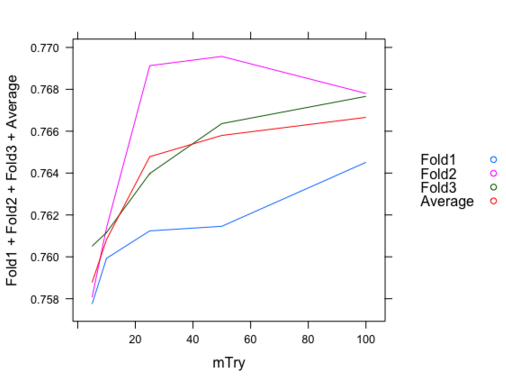
****

Figure 6: Random forest algorithm

### Results

When we tried for different values of *mtry* we identified that the accuracy was initially increasing as we increased the value of *mtry*. After a certain point the there was no major change in the performance. The higher accuracy was achieved for *mtry* value around 20. This result is shown in the graph in Figure 10. We selected the value of *mtry* as 10 as it gave better F1 score than other values.

Table 6: Random Forest sample results

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 78 | 2106 |
| 1 | 57 | 6905 |

This approach produced an accuracy of 76.4% and F1 score of 86.4 on the test dataset.

## Decision Trees with Boosting

The dataset in our problem has more missing values and huge class imbalance in it. Because of the class imbalance, the model was biased towards one class. Therefore, the use of boosting approach could potentially perform better. In this case, the general framework of the boosting depends on the iterative selection of wrongly classified data points. Similar to the random forest in its sampling methods for each of the decision tree constructed, boosting methods tends to add more weights to the training examples that were misclassified after a particular round. In other words, after a boosting round is complete, data points with larger weights have a higher probability of being selected for next boosting round. In this case, the final ensemble is obtained by aggregating the base classifiers obtained from each boosting round [22].

### Approach

We used the implementation of a generalized boosting algorithm proposed by Ridgeway [23] for training the data. The algorithm is based on the decision tree model. We used a package called *“gbm”* in R which is an implementation of Freund and Schapire's AdaBoost algorithm [24] and Friedman's gradient boosting machine. The package has implementations with the Adaboost’s exponential loss function [25] and the Friedman’s gradient descent algorithm [26]. With exponential loss, penalizing the misclassifications with large values would improve the performance.

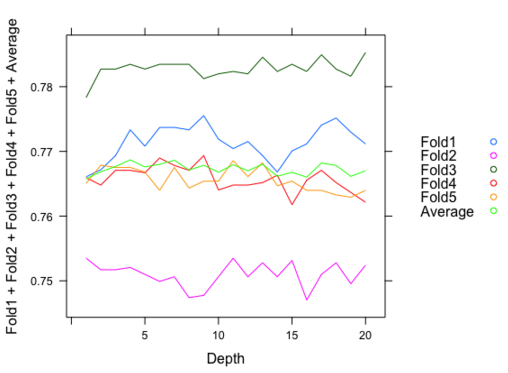


Figure 7: Decision tree with Boosting(GBM)

We used this *“gbm”* package and cross validated the training data to determine the optimal value for the depth parameter, which indicates the maximum depth of the variable interactions. We used 5 folds to train and cross validate the model. We trained with the number of trees as 1000. We have cross validated the model for 20 different depth values. The result we obtained in this method is shown in Figure 11.

### Results

The graph in the Figure 11 shows the performance of the gbm model on the different folds of training data for different values of interaction depth parameter. The accuracy was initially increasing as the depth value increases and then it decreased after a certain value. The accuracy was high with the depth value of 10.

Table 7: Decision Trees with Boosting sample results

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 251 | 1933 |
| 1 | 164 | 6798 |

We achieved an accuracy of 77.1% and F1 score of 86.5 on the test data. The boosting model gave the best performance when compared to the other algorithms as it was able to handle the class imbalance that was present in the data. The algorithm was also faster than other methods with a better performance overall.

## Neural Network

Neural networks are based generally on biological neuron systems. The simplest structure in the neural network system is the perceptron, which consists of two types of nodes: input nodes, which is used to represent the input attributes, and an output node, which is used to represent the model output [22]. The idea is that the perceptron computes its output value by performing a weighted sum on its inputs minus a bias factor. In our particular problem, each output node is assigned a function that ranges from 0 to 1 depending on the input value. In this case, the task is to determine how to assign weights to each individual node, such that at the end, the model is able to correctly classify a given data set. As we worked on the model development aspect of neural networks, we attempted to exploit the fact that our dataset consists of both categorical and continuous variables, the latter of which has multi-modal distribution. We also attempt to find an adequate neural network structure and activation functions for its nodes.

### Approach

The Architecture used for Neural Networks to train was a 3-layer neural network. The neural network consisted on 1 hidden layer 1 input layer and 1 output layer. The input to Neural Network consisted 118 units as there were 118 features, after we removed about 13 features, as they do not contribute towards the classification (Figure 12). This was then passed to the Hidden Layer of the input Neural Network. Each of the Neural Network activation unit was a logistic sigmoid function and we used 100 such activation units to train the neural network. The reason we used logistic sigmoid as the activation unit and not *tanh* a.k.a hyperbolic tangent activation was because we wanted the output to be ranged from [-1,1]. This would make it easier for the output not to have any bias parameter and simply use 0 as the threshold for classification. Also logistic sigmoid is just another way of representing the softmax function and we wanted a “soft thresholding” as compared to *tanh* function which is very abrupt. The lower values of *tanh* are linear and thus the output would be significantly constant even if there was a large difference in the input for negative values. Furthermore, 60 epochs were used, which represents the number of times our particular set of training vectors were used to update the weights of the neural network.

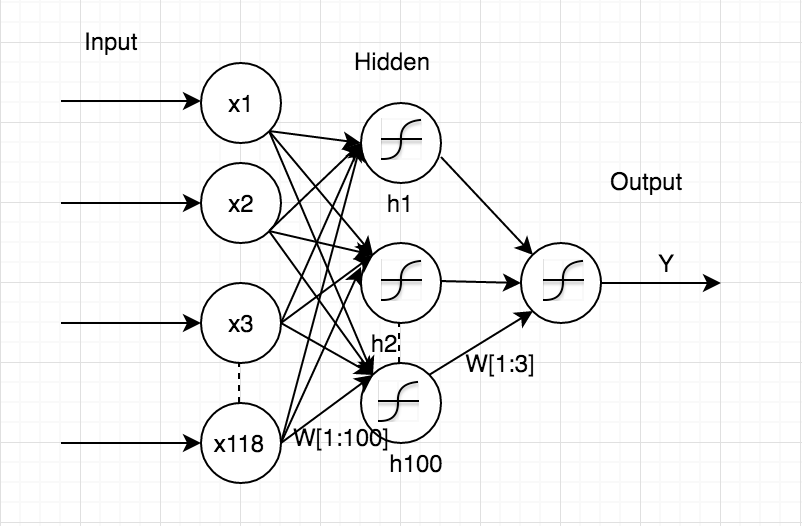


Figure 8: Model for Neural Networks

The Neural Net model with features where *t* ranges from 1:118, using a random variable which indicates whether the claim could or could not be accelerated. The Neural Network, with one layer of sigmoidal hidden units had 100 units. The hidden layer will have a set of weights mapping from its inputs to its hidden units. The output layer that predicts the output where the parameters of our model are Θ = {W [1], W [2]}. The hidden unit forms output by a logistic sigmoid over its weighted input.

This is the equation for every unit in the hidden layer.

The output of this equation is given to the output layer which decides the class based on the following equation

The Mean square error is calculated by where is the expected output of the data-sample and is the predicted output for the given data-sample.

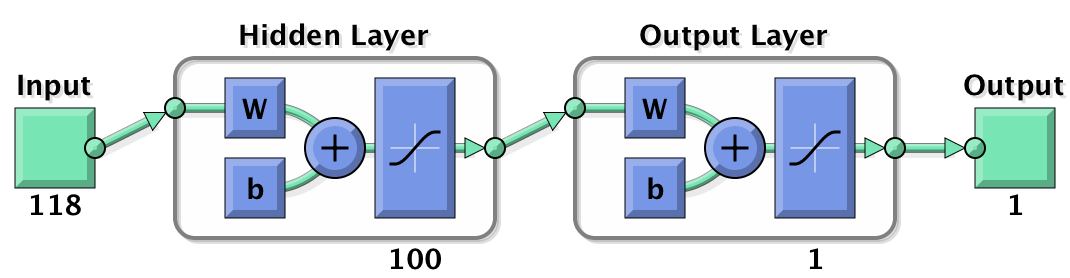


Figure 9: Neural Network Architecture for training

The Network used was the classic Feed-forward back propagation network. The algorithms used to train the neural network was Levenburg-Marquardt. This algorithm basically predicts the least curve fitting model by finding the MSE which is given by

### Results

The data was split into 60-40 where 60% of the data at random was used to train the Neural Network and 40% of the data was split to [20-20] for testing and cross-validating the dataset. The network did not have any bias term added to it and there was no drop out performed while training the model. Output was measured using mean square error, training of the model took 6 hour 30 minutes to complete.

The change in the gradient shows that the Neural indeed converged to the estimated minimum of and also that, there are not many oscillations while trying to converge. The validation check confirms that at every iteration the gradient was descending and trying to converge. Figure 14 explains the whole process of the Network for all the 60 epochs, at which the Neural Network converged.

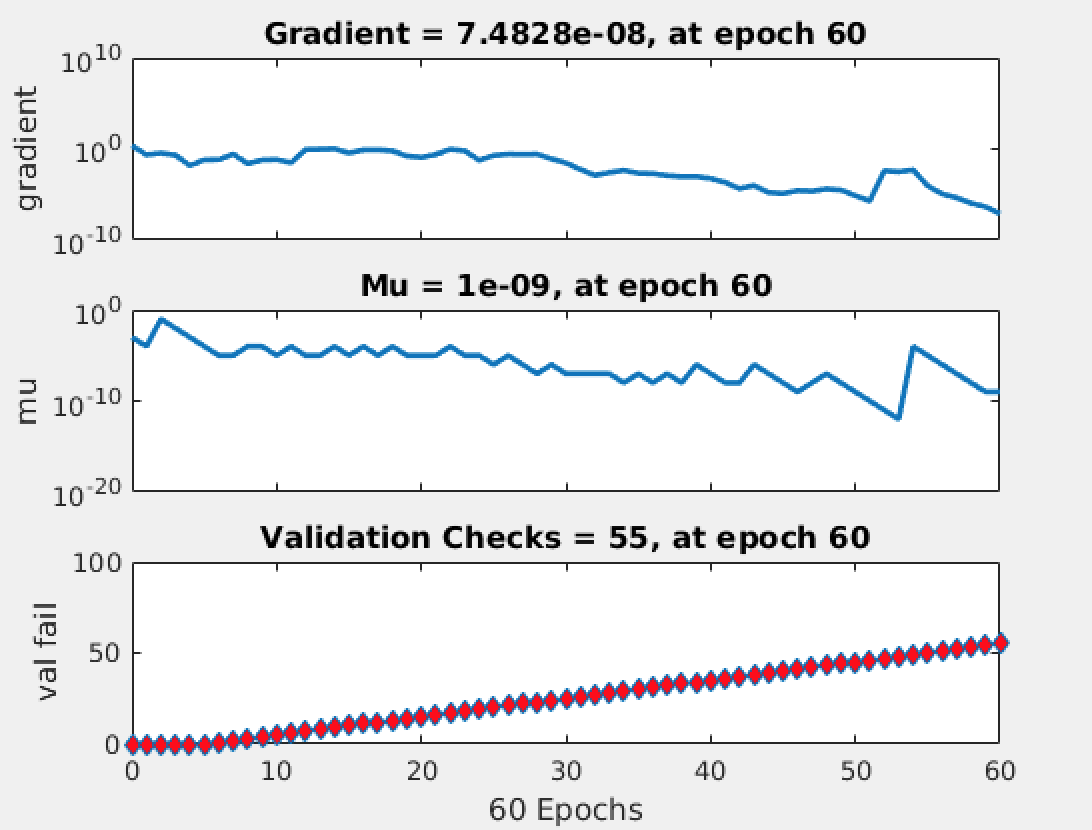


Figure 10: Gradient Parameters for the Neural Network

The Regression test shows that the global minimum of gradient descent did not fit the model well. However, the model fitted during the training phase behaved similarly to the one in test and validation, as shown below. The R value, which is a statistic measure of goodness of fit (mean squared error between observed value and fitted value), is similar in all four cases.

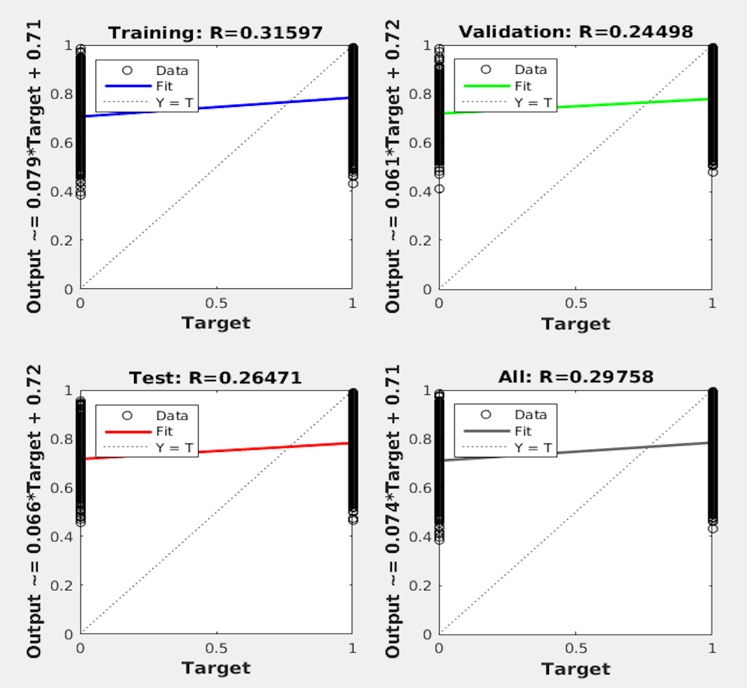


Figure 11: Regression Analysis of the Model

Performance testing here shows that the MSE was minimum for training and testing till the 5th epoch and deviated from there roughly by 0.23 percent, as shown in Figure 16.

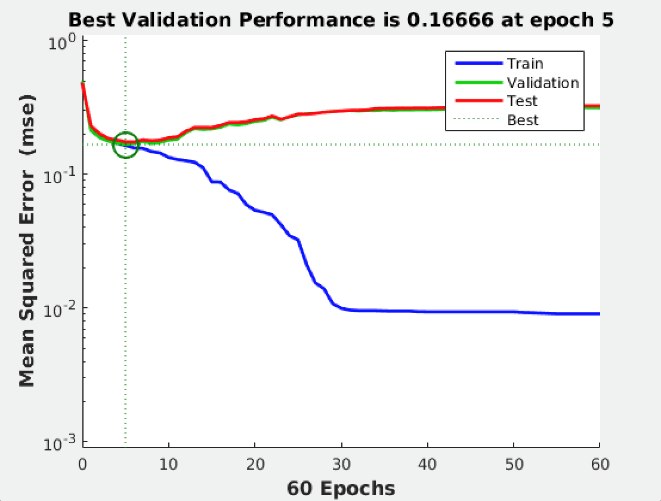


Figure 12: Performance of the model at each epoch

Finally, the classification results from our neural network were consistent with the results of the previously discussed. The results are shown in the table given below.

Table 8: Neural Networks sample results

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 523 | 1661 |
| 1 | 516 | 6446 |

In this case, the accuracy for the neural network algorithm was estimated to be 76.19%, while the F1 score was 85.4. Again, this is comparable to the results for the rest of the modeling approaches.

## Overall Performance

The performance of our dataset was deemed satisfactory. As mentioned in the introduction, the difference in log loss between the top Kaggle performer and the median was approximately 0.05. Thus, we knew that any marginal gain in performance for the accuracy measure over the testing set would be important. In this case, we observed that class imbalance is an important characteristic of our data set, since much fewer data points were selected for additional information (i.e. 0’s), which resulted in some difficulty being able to construct the models and having enough data points to correctly determine the parameters needed. Furthermore, given that in order to be able to run some of the data models (given CPU memory constraints), samples were taken from the original data set. In this case, we hypothesize that models that could potentially reduce the amount of bias would perform best. Therefore, initially we thought that ensemble-based methods would give the best results, such the combination Friedman Gradient Boosting/AdaBoost and Random Forest algorithms.

After model development and tuning, we found that our initial hypothesis was not far off. Based on the result summary presented in Table 9, we find that the best performer, in terms of accuracy over the testing set, was classification decision trees with boosting with an accuracy of 77.1%. This was almost one full percent over the second-best performer, Random Forest, with an accuracy of 76.4%. Interestingly, the worst performer over the testing set was support vector machines with an accuracy measure of 68.7%. Now, if we observe the F1 scores which also considers the recall measure, Random Forest was actually the best performer with a score of 86.5 and was followed closely by our boosting algorithm and logistic regression with scores of 86.4 and 86.3, respectively. Again, our support vector machine model did not produce good results with a score of 79.9. Finally, based on these results, our recommended model for this particular problem is classification decision trees with boosting given that it has both a high accuracy, as well as F1 score.

Table 9: Model Metrics Comparison

|  |  |  |
| --- | --- | --- |
|  | Accuracy | F1-Score |
| SVM (C=100, gamma=0.01) | 68.7% | 79.9 |
| Classification Decision Trees w/Boosting (depth=10) | 77.1% | 86.4 |
| Random Forest (mtry = 10, Trees = 500) | 76.4% | 86.5 |
| Logistic Regression | 76.1% | 86.3 |
| Logistic Regression (using Stochastic Gradient) | 75.4% | 85.8 |
| NN (Hidden Layer(Units) = 1(100), Activation function = logsigmoid, Epochs=60) | 76.2% | 85.4 |

Next, we finalize this report by discussing additional insights into our modeling framework and possible extensions that could have been taken to improve our predictive performance. We also discuss general concepts that our team learned through this process and what other teams could potentially learn from our experience.

# Conclusions and Final Discussion

As explained in our introduction, this project involved several different problems that we had to deal with that ranged from feature selection to the construction of adequate predictive models. In this case, we believe that our approach was appropriate with respect to the needs of the project. In the feature selection phase, we took two main approaches for selecting an optimal subset of variables that had the most predictive power. From the statistical perspective, we were able to reduce the number of features to roughly 9. Most importantly, using an adaptive lasso technique under a logistic regression framework available in JMP Statistical Software, we were able to identify the specific factor lcevels most important to the model, which in some cases improved our predictions. The second approach was more information theory related techniques which included information gain and the Boruta algorithm. Based on these methods, we were able to corroborate some of the findings from our first approach, while also gaining additional insights into other variables that may have predictive power. Although, we believe that our approach was appropriate, there are many other things that could have been applied to this phase of the problem, given that the field is so extensive. This includes the potential application of meta-heuristic technique (e.g. swarm and ant colony optimization) that could have given other type of results and could have reduced algorithm running times.

In the model development process, all team members gained a greater appreciation for creating models that are both predictive but are also efficient. Given that our data set was large and the number of variables expanded (when recoding from categories to indicators), we had to find ways to improve the convergence times of our models. We also wanted to compare the predictive power of different modeling techniques so that we had several options when selecting the final model to use. Again, although, we believe that our approach was adequate we could also have expanded our list of options. For example, we could have considered using a stochastic version of support vector machines, sequential minimal optimization, to be able to make the model converge using a full data set. However, given that the accuracy was so low for the full model approach, we did not pursue this path. Finally, there are several aspects of our modeling process that could be of use to people working on similar type projects. This includes our general methodology framework of data exploration, imputation, feature selection, and modeling framework analyses. Since we wanted to show that ensemble methods are actually the best choice for this kind of problem, interested readers could also look at our specific problem and could potentially decide to strictly expand on the use of ensemble methods to improve on the predictive performance of our models. Furthermore, we stress the importance for considering statistical analysis in the sampling of data points, feature selection, and model development.

# Team Member Contribution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | ASU ID | Email | Specific Contribution | (%) |
| Hector Flores | 1000872235 | [hector.flores@asu.edu](mailto:hector.flores@asu.edu) | * Modified existing gradient descent method for our data set (expanded for stochastic components) for logistic regression * Feature Selection Process, including adaptive lasso techniques, information gain, and Boruta algorithms * Worked on SVM, Dtrees, and random forests * Report: Sections 3, 4.2.1, 4.7, and 5, report structuring, proofread | 16.67% |
| Jayaprakash Jayakumar | 1209340128 | [jjayaku1@asu.edu](mailto:jjayaku1@asu.edu) | * Worked on regularized logistic regression model * Worked on Random Forest models * Decision trees with boosting * Cross-Validation for SVM * Report: Sections 4.1, 4.4, 4.5, report structuring * Cross-Validation for SVM | 16.67% |
| Sudarshan Venkat Ram | 1209590886 | [svram@asu.edu](mailto:svram@asu.edu) | * Data imputation, Sampling * Worked on Random Forest models * Boosting models * SVM models | 16.67% |
| Sudharsh Subbaraman | 1207464111 | [ssubbar2@asu.edu](mailto:ssubbar2@asu.edu) | * SVM models * Cross-Validation for Boosting and SVM models * Logistic regression model * Report: Sections 4.3, contributions to section 1.1, 2, structuring and proofreading | 16.67% |
| Akshay Iyangar | 1208604211 | [aiyangar@asu.edu](mailto:aiyangar@asu.edu) | * Neural network models * Data imputation * SVM models | 16.67% |
| Neha Prasad | 1208662958 | [nhprasad@asu.edu](mailto:nhprasad@asu.edu) | * Neural network models * Handling of categorical variables * Data analysis | 16.67% |

# Bibliography

[1] L. A. Jeni, J. F. Cohn, and F. D. L. Torre, “Facing Imbalanced Data–Recommendations for the Use of Performance Metrics,” in *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*, 2013, pp. 245–251.

[2] M. L. Bermingham, R. Pong-Wong, A. Spiliopoulou, C. Hayward, I. Rudan, H. Campbell, A. F. Wright, J. F. Wilson, F. Agakov, P. Navarro, and C. S. Haley, “Application of high-dimensional feature selection: evaluation for genomic prediction in man,” *Sci. Rep.*, vol. 5, p. 10312, May 2015.

[3] A. Agresti and I. Liu, “Strategies for Modeling a Categorical Variable Allowing Multiple Category Choices,” *Sociol. Methods Res.*, vol. 29, no. 4, pp. 403–434, May 2001.

[4] W. K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[5] D. Yates, D. S. Moore, and D. S. Starnes, *The Practice of Statistics: TI-83/89 Graphing Calculator Enhanced*. Macmillan, 2002.

[6] S. van Buuren, “Multiple imputation of discrete and continuous data by fully conditional specification,” *Stat. Methods Med. Res.*, vol. 16, no. 3, pp. 219–242, Jun. 2007.

[7] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artif. Intell.*, vol. 97, no. 1–2, pp. 273–324, Dec. 1997.

[8] Y. Saeys, I. Inza, and P. Larrañaga, “A review of feature selection techniques in bioinformatics,” *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, Oct. 2007.

[9] T. Mehmood, K. H. Liland, L. Snipen, and S. Sæbø, “A review of variable selection methods in Partial Least Squares Regression,” *Chemom. Intell. Lab. Syst.*, vol. 118, pp. 62–69, Aug. 2012.

[10] M. B. Kursa and W. R. Rudnicki, “Feature Selection with the Boruta Package,” *J. Stat. Softw.*, vol. 36, no. 11, Sep. 2010.

[11] D. G. Kleinbaum and M. Klein, *Logistic Regression: A Self-Learning Text*. Springer Science & Business Media, 2010.

[12] S. Domínguez-Almendros, N. Benítez-Parejo, and A. R. Gonzalez-Ramirez, “Logistic regression models,” *Allergol. Immunopathol. (Madr.)*, vol. 39, no. 5, pp. 295–305, Sep. 2011.

[13] smc77, “Logistic Regression with Gradient Descent.” 2011.

[14] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, 1992, pp. 144–152.

[15] W. H. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[16] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[17] A. Karatzoglou, D. Meyer, and K. Hornik, “Support Vector Machines in R,” 2005. [Online]. Available: http://epub.wu.ac.at/1500/. [Accessed: 02-May-2016].

[18] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, “A Dual Coordinate Descent Method for Large-scale Linear SVM,” in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, pp. 408–415.

[19] N. Japkowicz, “The class imbalance problem: Significance and strategies,” *Proc Int’l Conf Artif. Intell.*, 2000.

[20] A. Liaw and M. Wiener, “Classification and regression by randomForest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[21] K. J. Archer and R. V. Kimes, “Empirical characterization of random forest variable importance measures,” *Comput. Stat. Data Anal.*, vol. 52, no. 4, pp. 2249–2260, Jan. 2008.

[22] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[23] G. Ridgeway, “gbm: Generalized boosted regression models,” *R Package Version*, vol. 1, no. 3, 2006.

[24] Y. Freund, R. E. Schapire, and others, “Experiments with a new boosting algorithm,” *ICML*, vol. 96, pp. 148–156, 1996.

[25] J. Elith, J. R. Leathwick, and T. Hastie, “A working guide to boosted regression trees,” *J. Anim. Ecol.*, vol. 77, no. 4, pp. 802–813, Jul. 2008.

[26] J. H. Friedman, “Stochastic gradient boosting,” *Comput. Stat. Data Anal.*, vol. 38, no. 4, pp. 367–378, Feb. 2002.