Telstra Network Disruption Prediction

Vijayalakshmi Vedantham, Vertika Srivastava, Sakshi Maskara, Nidhi Singh

20th November 2016

Contents

Chapter 1

Introduction

Telstra is Australia's leading provider of mobile phones, mobile devices, home phones and broadband internet.

1.1 Problem Description

The objective of the project is to predict the severity of service disruptions on their network at a time at a particular location. The data set features are obtained from service logs. The severity is divided into three classes(0,1 and 2). Here 0 means no fault and 2 is the most severe disruption.

1.2 Data Description

Each row in the main data set (train.csv, test.csv) represents a location and a time point. They are identified by the "id" column, which is the key "id" used in other data files. Telstra has stored all the information in the following files:

1. train.csv : train set for fault severity
2. test.csv : the test set for fault severity
3. event_type.csv : event type related to the main dataset [31170 (rows) x 2 (columns)]
4. log_feature.csv : features extracted from log files [58671(rows) x 3(columns)]
5. resource_type.csv : type of resource related to the main dataset [21076 (rows) x 2 (columns)]
6. severity_type.csv : severity type of a warning message coming from the log [18552 (rows) x 2 (columns)]

Chapter 2

Data Preprocessing

2.1 Merging the Dataset

Our initial data set was spread across six different files. Before merging the files, we needed to clean up the files to make it easier to merge. This involved converting features of the form "resource_type x","event_type x","location x","feature x" etc into categorical variables, since numbers are easier to deal with than string values. We did this by a simple string split function.

Next we created a dataframe with the index as severity.csv's index. The intution behind this is that the severity file will contain all the ids. Next we started merge all the files one by one into one large master file using left join since all the files had an id feature.

We concatenated train and test files because all the features for train and test were coming from the other files, which we combined together first and then separate out the train and test for the final fit and prediction.

2.2 Feature Enginnering

We counted the number of times each location was found in the dataset and added a column as the count to each location id.

In the data description, it is mentioned that a location and a time value is associated with each training and test instance. But this time information is not present in any of the files. When we were merging the file, we saw the lcoation id's appeared in sorted order. We wanted to check if there was any other order within the sorted order so we created a nummbering column which had values from 1,2,3...n for each location id, We also create a reverse_numbering feature

and gave it values from n,...2,1. Since these values were not normalized, we created normalized versions of these variables too.

We create a count for number of resources used by each id.Then we created a frequency based encoding for the features. We created a column for each resource type and added their count (for each id) as the value. If the id did not contain the resource id then we filled in a 0. Thus the multi-row non-unique id as index file resource_type file was converted into a file with unique id as index.

We performed similar operations for event types too. We create a count for number of events for each id and then created a frequency based encoder.

We converted the volume into log scale in order to match it with the distributions of other features(otherwise the range was very large), then we added the aggregate features for the volume for each id - such as min,max,count, avg and sum.

Chapter 3

Models

3.1 Random Forest Classifier

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

A number of weakly learnt models can be combined to form a good model. Random forests is a way of combining multiple decision trees which are trained on different parts of training set and different parts of feature set. Therefore, there is no chance of overfitting.

3.2 Extra Trees Classifier

In extremely randomized trees, randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

It essentially consists of randomizing strongly both attribute and cut-point choice while splitting a tree node.

In the extreme case, it builds totally randomized trees whose structures are independent of the output values of the learning sample. The strength of the

randomization can be tuned to problem specifics by the appropriate choice of a parameter.

3.3 XGBoost Classifier

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

1. Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
2. Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
3. Regularized Gradient Boosting with both L1 and L2 regularization

The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

1. Sparse Aware implementation with automatic handling of missing data values.
2. Block Structure to support the parallelization of tree construction.
3. Continued Training so that you can further boost an already fitted model on new data.

Chapter 4

Implementation

We have implemented different models with different parameters. Here is an overview of the three different models that we have applied.

4.1 Random Forests

The parameters that we have mainly adjusted is the number of trees (as known as n_estimators) that will be learnt. Each tree is learnt by a subset of training set and a subset of features.

The no. of trees are decided on the basis of memory used, time taken to compute the model and the accuracy of prediction.

There are other parameters also like max_features but these didn't show a significant difference. So, these features remained at the default setting.

We plotted three graphs to check which n_estimators will suit best to the model. In the first graph Figure 3.1 we plotted, no. of trees(n_estimators)(which is varied from 1 to 3000) on x-axis to multiclass log-loss on y-axis.

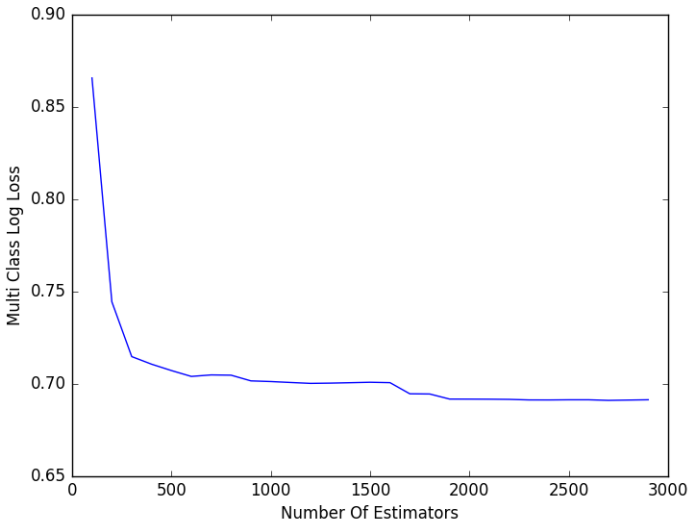

Figure 4.1: plots for n_estimators from 1 to 3000 to multiclass-log loss

In the second graph Figure 3.2 we plotted, no. of trees(n_estimators) (which are varied from 2400 to 2850) on x-axis to multiclass log-loss on y-axis.

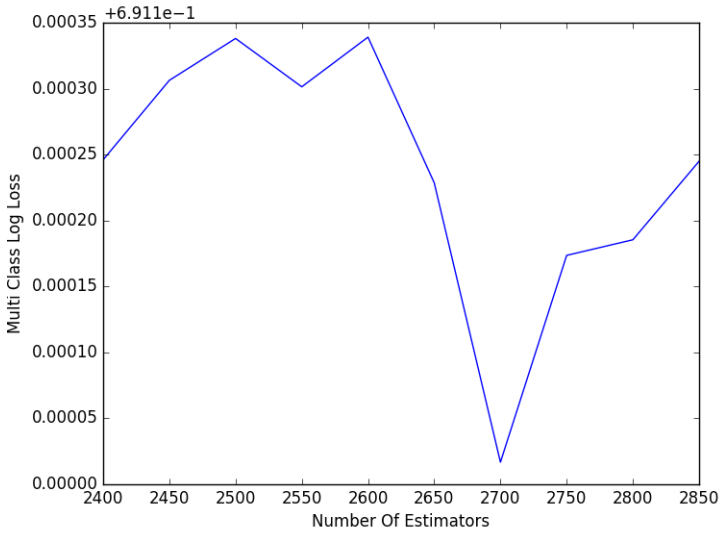

Figure 4.2: plots for n_estimators from 2400 to 2850 to multiclass-log loss

In the third graph Figure 3.3 we plotted, no. of trees(n_estimators) (which are varied from 2680 to 2715) on x-axis to multiclass log-loss on y-axis.

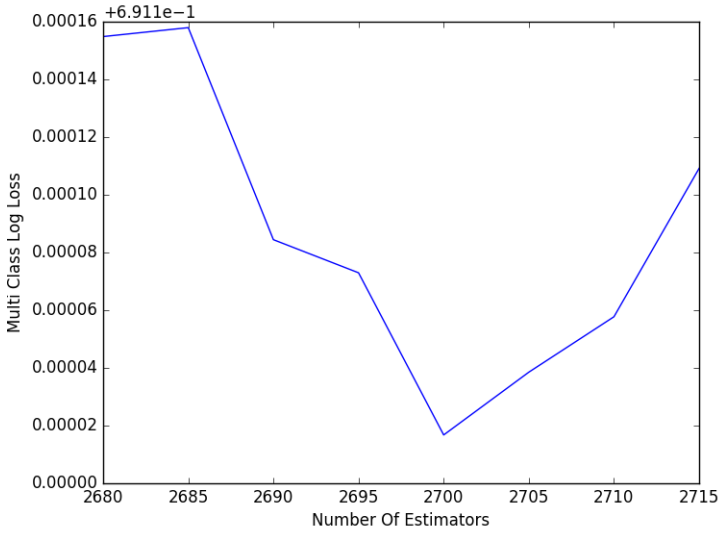

Figure 4.3: plots for n_estimators from 2680 to 2715 to multiclass-log loss

4.2 Extra Trees Classifier

The main parameters to adjust when using these methods is n_estimators and max_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max_features=n_features for regression problems, and max_features=sqrt(n_features) for classification tasks (where n_features is the number of features in the data).

In addition, note that in random forests, bootstrap samples are used by default (bootstrap=True) while the default strategy for extra-trees is to use the whole dataset (bootstrap=False).

We have plotted three graphs to determine the optimum value of n_estimators. These are plotted against M Log Loss Error. The first graph Figure 3.4 is plotted in the range of 0 to 2000.

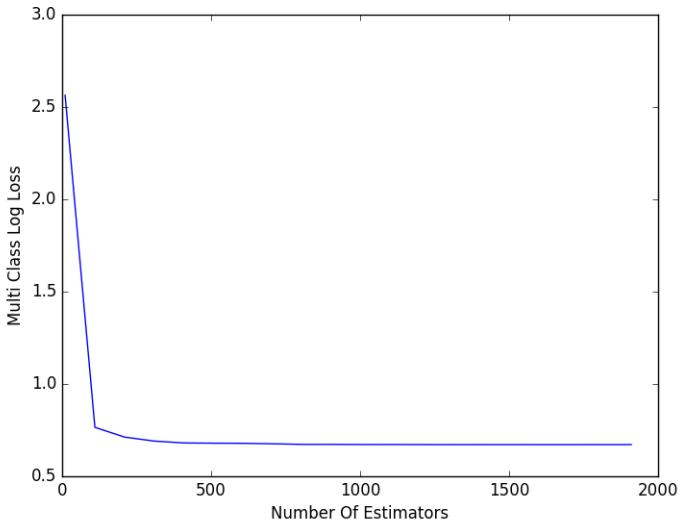

Figure 4.4:

We discovered that the graph has a knee in the range of 0 to 500. So, we plotted another graph Figure 3.5 in this range. Here, we observed that the graph is decreasing in the range of 340 to 460.

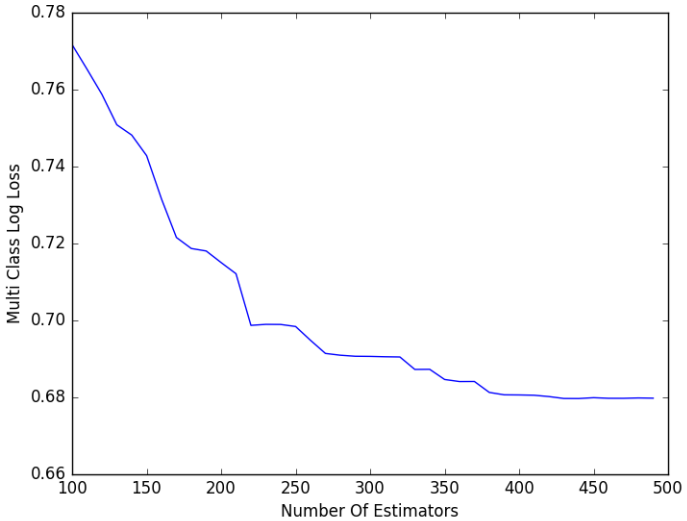

Figure 4.5:

Therefore, we plotted another graph Figure 3.6 in the range of 340 to 460. Finally, we got the minimum value of n_estimators as 440.

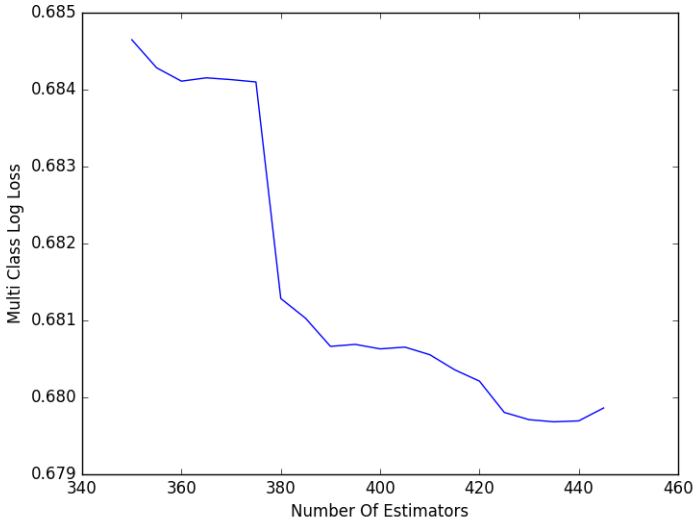

Figure 4.6:

4.3 XGBoost Classifier

We have implemented XGBoost model on our dataset using parameters like eta, num_class, max_depth and nthread.We have used multi:softprob as we wanted probability of each of the 3 class for each test dataset. We have used mlogloss for evaluation metric.

nthread: This is used for parallel processing.Using number of cores of the system for this parameter.

eta: It is the learning rate.

num_class: It is the number of classes to be used for classification for the dataset.

eval_metric: It is the metric to be used for validation data.

objective: The objective function used is multi:softmax. It is the multiclass classification using the softmax objective, returns predicted class.

max_depth: It is the maximum depth of a tree.

We have plotted three graphs to determine the optimum value of three different features. These are plotted against M Log Loss Error. The first graph Figure 3.7 is plotted between max-depth and multi class log loss.

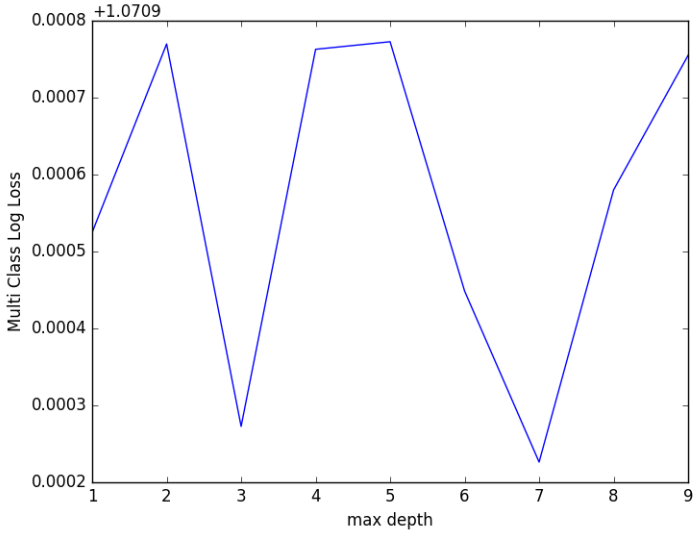

Figure 4.7:

We plotted another graph Figure 3.8 between n_threads and multi class log loss.

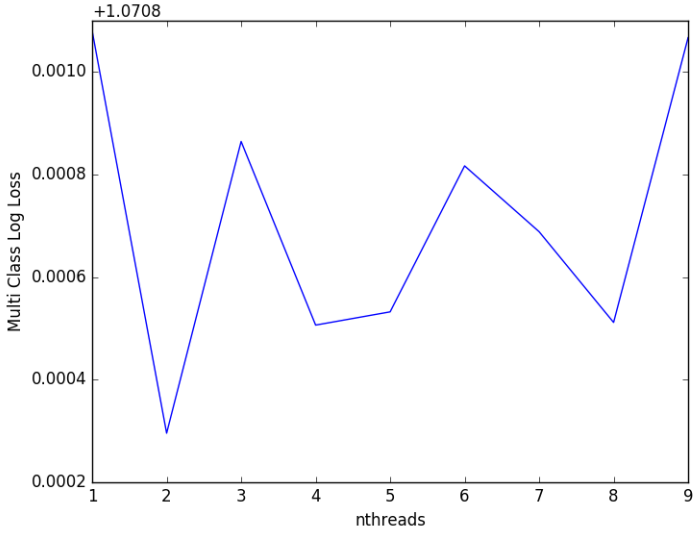

Figure 4.8:

We plotted another graph Figure 3.9 for another feature known as rounds.

This graph is plotted between rounds and multi class log loss.

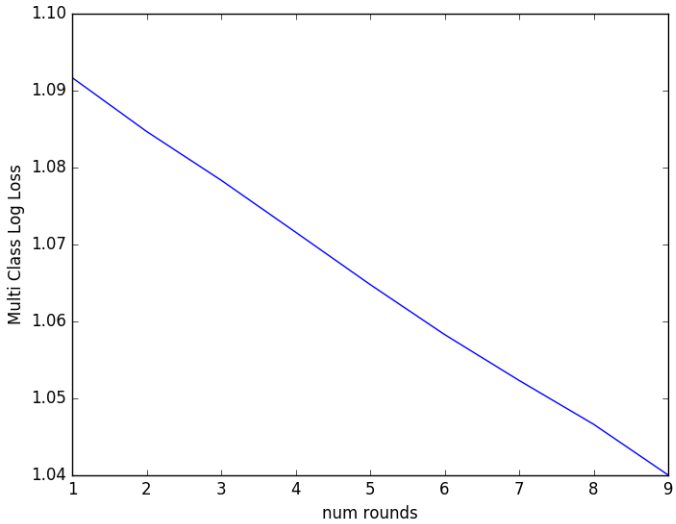

Figure 4.9:

Chapter 5

Observations

By applying the three selected models with selected parameters, the accuracy on training set and test set is mentioned in the table 5.1

Model	**Training Set Accuracy**	**Test Set Accuracy**
XGBoost	77.88	73.60
Random Forest	100	70.08
Extra trees Classifier	100	69.89

Table 5.1: Observations with different models

Chapter 6

Conclusion

We have implemented different ensembling methods like XGBoost, Random Forest and Extra Trees Classifier to predict the severity of disruption. We also implemented Neural Networks but it was giving very less accuracy so we dropped it. We have also done considerable data preprocessing and feature engineering to the raw data that was present in multiple data files.

Chapter 7

References

1. "Brainwash: A Data System for Feature Engineering" By Michael Anderson , Dolan Antenucci , Victor Bittorf , Matthew Burgess , Michael Cafarella , Arun Kumar , Feng Niu , Yongjoo Park , Christopher Ré , Ce Zhang.

2. Thomas G. Dietterich. "Ensemble Methods in Machine Learning." Multiple Classifier Systems, LBCS-1857

3. Thomas G. Dietterich. "Machine Learning Research Four Current Directions." AI Magazine

4. Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." The Annals of Statistics 29.5 (2001): 1189-232. Web

5. "Combining Information Extraction Systems Using Voting and Stacked Generalization." By Georgios Sigletos , Georgios Paliouras , Constantine D. Spyropoulos , Michalis Hatzopoulos

6. "Multi class version of Logarithmic Loss metric. https://www.kaggle.com/wiki/MultiClassLogLoss"