The Title

By

Firstname Middlename Surname (email@aims.ac.rw)

June 2017

AN ESSAY PRESENTED TO AIMS RWANDA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE IN MATHEMATICAL SCIENCES

AIMS
African Institute for
Mathematical Sciences
RWANDA

DECLARATION

This work was carried out at AIMS Rwanda in partial fulfilment of the requirements for a Master of Science Degree.

I hereby declare that except where due acknowledgement is made, this work has never been presented wholly or in part for the award of a degree at AIMS Rwanda or any other University.

Scan your signature

Student: Firstname Middlename Surname

Scan your signature

Supervisor: Firstname Middlename Surname

ACKNOWLEDGEMENTS

This is optional and should be at most half a page. Thanks Ma, Thanks Pa. One paragraph in normal language is the most respectful.

Do not use too much bold, any figures, or sign at the bottom.

DEDICATION

This is optional.

Abstract

A short, abstracted description of your essay goes here. It should be about 100 words long. But write it last.

An abstract is not a summary of your essay: it's an abstraction of that. It tells the readers why they should be interested in your essay but summarises all they need to know if they read no further.

The writing style used in an abstract is like the style used in the rest of your essay: concise, clear and direct. In the rest of the essay, however, you will introduce and use technical terms. In the abstract you should avoid them in order to make the result comprehensible to all.

You may like to repeat the abstract in your mother tongue.

Contents

1. Introduction

1.1 Problem Statement

Risk management is one of the fundamental tasks of insurance companies. The insurance industry should constantly adopt new technologies to address new risk types and trends affecting people's lives. We depend on insurance for several reasons but it all scales down to the basic principle: minimizing risk in that clients pay a fee, and in exchange, insurers cover any costs that could arise with future calamities.

Clients provide extensive information to identify risk classification and illegibility, including scheduling medical exams, family medical history, credit history, behavioural risk factors, and so on, making the whole insurance process lengthy. The outcome is that the clients are turned off. This constitutes the main reason why majority of the households do not own individual life insurance (Web, Accessed March 2017b).

Life underwriting has its own modelling challenges making insurers to turn to predictive analytics to curb the problems. It is worth noting that auto underwriting has achieved remarkable success with predictive modelling unlike life underwriting where modelling is a new skill (pred).

Predictive modeling: It is defined as the process of developing models such that the model's prediction accuracy on future or unseen data can be understood and quantified. Therefore, predictive modelling is a combination of machine learning, pattern recognition and data mining Kuhn and Johnson (2013).

Predictive modelling has its roots in actuarial science where analysts seek to determine the right price for the right risk(rate making) and avoid adverse selection Frees et al. (2014).

(pred) Building a predictive model requires:

- The availability of a sufficiently rich dataset where the predictive variables that correlate with the target can be identified.
- An application by which results from the model are translated to business actions.
- A clearly defined target variable.
- A large number of observations to build the model so as surfacing relationships can be separated from random noise.

The above requirements are easily met with auto insurance. To clearly understand the challenges faced in life insurance, we compare life underwriting and auto underwriting (pred).

- Auto insurers can make underwriting corrections if mistakes are made through rate increases in subsequent renewals of policies whereas life insurers must price policies appropriately from the outset.

- For the auto insurer, the amount of insurance loss of a six-month contract is a target variable for the model. Life insurance is sold through long duration contracts, usually over a period of 10, 20 or more years. Due to the fact that the contribution of a given risk factor to mortality could change with time, it is not sufficient to analyse mortality experience over a short period of time.

- Accessing historical data that can be used in modelling life insurance is a challenge. Not all life insurers record underwriting data in an electronic format; The available underwriting data that has been implemented in recent years is not available electronically or in a machine readable form. Even when such data has been captured for years, the content of the older data may be different from the data gathered for current applicants.

- Life underwriting is subject to psychological biases and inconsistencies of human decision-making thus predictive models help to curb this challenge.

- Life insurance claims have low frequency compared to auto insurance claims. Modelling statistically significant variation in either auto claims or mortality requires a large sample of loss events. Therefore auto insurers have ample data to build robust models using loss data while life insurers will find the data recorded in at similar times frames insufficient for modelling.

  Given that the target variable and data volume in life insurance is a concern, insurers are utilizing underwriting decisions as the target variable as they contain a lot of information, expert judgement, do not require long developing periods as in insurance claims and are abundant in supply.

1.2 Objective

I am working on data from Prudential Life Insurance where the challenge is trying to make purchasing of life insurance easier by developing a predictive model that accurately classifies risk using a more automated approach Web (Accessed March 2017a). The data I am working on is from kaggle which is a platform for data science competitions. The host provides raw data and a description of the problem. Those participating in the competition then train algorithms where highly performing models can be adopted for predicting similar trends in the future.

1.3 Trends in Insurance

Underwriting

This is the process of understanding and evaluating risk in insuring a life or property. This ability is gained not only through theoretical study but also as a result of years of experience dealing with similar risks and mastering the art of paying claims on those risks. It is the traditional way of pricing and classifying risks in insurance (Macedo, 2009).

Dickson, Hardy, and Waters (2013) An insurance life office will have a premium rate schedule for a given type of policy. This rates depend on **the size of the policy and rating factors**. To establish an applicants risk level, a proposal form giving information on relevant rating factors such as age, gender, any dangerous hobbies, occupation, smoking habits, and health history is filled. The purpose of underwriting is to classify potential policy holders into homogeneous risk categories and assess what additional premium would be appropriate if risk factors indicate that standard premium rates would be too low.

Disadvantages

- There is the risk of **adverse selection** by policy holders if the underwriting is not strict. This means that very high-risk individuals will buy insurance in disproportionate numbers leading to excessive losses.
- The underwriting process could be lengthy and costly.
- Both the insurer and policy holder may assume 'utmost good faith' such that in case of loss and important information was held back or false, then the full sum assured may not be paid by the insurer in case the client claims from the insurance.

Thus, the use of predictive models makes the underwriting process faster, more economical, more efficient and more consistent when the model is used to analyze a set of underwriting requirements.It is also worth noting that models are not subject to bias in the same way that underwriters ,who do not always act with perfect consistency or optimally weigh disparate pieces of evidence, are.

1.4 Overview of Predictive Modelling

In predictive modelling, two situations arise:

- One is required to fit a well-defined parametrized model to the data using a learning algorithm that can find parameters on the large dataset without over-fitting. In this case, lasso and elastic-net regularized generalised linear models are a set of modern algorithms which meet this need because they are fast, work on huge datasets and avoid over-fitting automatically.
- One needs to accurately predict a dependent variable. A learning algorithm that automatically identifies the structures, interactions and relationships in the data is needed. In this case, ensembles of decision trees (known as 'Random Forests') have been the most successful algorithm in modern times and basically this is what my work entails.

(Berry and Linoff, 1997) Learning problems can be roughly categorized as either supervised or unsupervised.

Supervised learning: For each observation of the predictor measurement(s) $x_i, i = 1, 2, ..., n$, there is an associated response measurement y_i Gareth, Daniela, and Trevor (2014). We fit a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations(predictions) and understand the relationship between the response and the predictors. Traditional statistical learning methods such as linear regression and logistic regression as well as modern approaches such as boosting and support vector machines work in the supervised learning domain.

Unsupervised learning: For every observation $i = 1, 2, ..., n$, we observe a vector of measurements x_i but no associated response y_i. A linear regression model cannot be fit because there is no response variable to predict. In this case we seek to find the relationship between the variables. One statistical tool that can be used is cluster analysis. Clustering ascertains whether the observations fall into distinct groups.

Supervised learning can be grouped into **Regression** and **Classification** problems.

Regression Vs Classification Problems

Variables can be grouped into **quantitative** or **qualitative** (categorical) Gareth, Daniela, and Trevor (2014). Quantitative variables take on numerical values eg. height while qualitative variables take on values in one of the different classes eg. gender.

Problems with a quantitative response are referred to as regression problems while those involving a qualitative response are referred to as classification problems. Predicting a qualitative response for an observation is called classifying the observation since it involves assigning the observation to a class. Three of the most widely-used classifiers: logistic regression, k-nearest neighbors and Linear Discriminant Analysis. More computer-intensive methods are trees, random forests, support vector machines and boosting Gareth, Daniela, and Trevor (2014).

Machine Learning: This is a method of teaching computers to improve and make predictions based on data. It is teaching a program to react to and recognize patterns through analysis, self training, observation and experience (Hackeling, 2014).

In the classification setting, we have a set of training observations $(x_1, y_1), ..., (x_n, y_n)$ that we can use to build a classifier. We want our classifier to perform well not only on the training data, but also on test observation not used to train the classifier. Here, I introduce some of the most commonly used classifiers.

Logistic Regression

Models the probability that Y, the dependent variable belongs to a particular category (one of two categories eg. 'yes' or 'no').

The model

Modelling the relationship between $p(X) = pr(Y = 1/X)$ and X. For convenience we use the generic coding 0 and 1 for the response. To use linear regression to represent this probabilities we have, $p(X) = \beta_0 + \beta_1 X$ which gives the left hand side of the logistic function.

However, there is a problem with this approach in that predicting of values close to zero would yield negative probabilities and if we were to predict large values, we would get probabilities bigger than 1 which defies the law of probability that probability values should fall between 0 and 1.

To prevent this,we model $p(X)$ using the logistic function that gives outputs between 0 and 1 for all values of X.

$$p(X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$$

We notice that for lower values, we now predict the probability of default as close to but never below 0. Likewise for high values, we predict a default probability close to but, never above 1.

Manipulating the equation gives;

$$\frac{p(X)}{1 - p(X)} = \exp(\beta_0 + \beta_1 X) \tag{1.4.1}$$

where the LHS is called odds and takes values between 0 and ∞. Taking the logarithms on both sides yields:

$$log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X \tag{1.4.2}$$

The LHS is called the log-odds or logit which is linear in X.

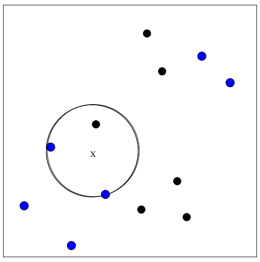
K-nearest Neighbors

KNN can handle both binary and multi-class data.

Consider having N training objects, each of which is represented by a set of attributes X_n and a label Y_n. Suppose we want to classify new objects X_{new}, we first find the K training points closest to X_{new}. Y_{new} is then set to be the majority class amongst these neighbors.

The figure below provides an illustrative example of the KNN approach Gareth, Daniela, and Trevor (2014).

Figure 1.1: KNN approach using K=3.



The goal is to predict the point labelled X. Suppose we choose $K = 3$, KNN will first identify the three observations closest to X, as shown in the diagram. The point consists of two blue points and a red point resulting in estimated probabilities of $\frac{2}{3}$ for the blue class and $\frac{1}{3}$ for the black class. KNN will predict that the point X belongs to the blue class.

One draw back of KNN is the issue of ties: Two or more classes having equal number of ties. Therefore, for binary classification, a good solution is to always use an odd number of neighbors.

Choosing K: If K is too small, the classification is heavily influenced by mislabelled points(noise). This problem is rectified by increasing K which regularises the boundary.

What about if K is too big? As we increase K, we are using neighbours further away from X_{new} which is useful upto a certain point as it has a regularizing effect that reduces the chances of overfitting. However, when we go too far, we loose the true pattern of the data we are attempting to model. Therefore, to find the best value of K, we use cross validation Gareth, Daniela, and Trevor (2014).

Linear Discriminant Analysis

Linear Discriminant Analysis is used:

- LDA is popular when we have more than two response classes Gareth, Daniela, and Trevor (2014).

Using bayes' theorem for classification

Suppose we wish to classify an observation into K classes, where $K \geqslant 2$ and the qualitative response variable Y can take on K distinct ordered values.

π_k: Denotes the prior probability that a randomly chosen observation comes from class K of the response variable Y.

$f_k = Pr(X = x/Y = k)$: Denote the density function of X for an observation from class K.

Bayes theorem states that:

$$P_k(X) = Pr(Y = k/X = x) = \frac{\pi_k f_k(x)}{\sum \pi_i f_i(x)} \tag{1.4.3}$$

$P_k(x)$: Posterior probability that an observation $X = x$ belongs to class K.

π_k is computed if we have a random sample of Y_s from the population. We therefore compute the fraction of training observations that belong to class K. $f_k(x)$ is estimated so we can develop a classifier that approximates the bayes classifier.

Linear Discriminant Analysis for $p = 1$. (We have only one predictor)

We are required to find an estimate for $f_k(x)$ that we can plug into (1.4.3) inorder to estimate $p_k(x)$. We then classify an observation for which $p_k(x)$ is greatest. Gareth, Daniela, and Trevor (2014) To estimate $f_k(x)$ the following assumptions are made:

- $f_k(x)$ is normal.

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right) \tag{1.4.4}$$

  μ_k and σ_k^2 are the mean and variance parameters for class k.

- Shared variance term across all K classes, $\sigma_1^2 =, ..., = \sigma_K^2$

Plugging equation (1.4.4) to equation (1.4.3) yields:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{i=1}^{K} \pi_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_i)^2\right)} \tag{1.4.5}$$

Taking the logs of equation (1.4.5) and rearranging the terms gives:

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k) \tag{1.4.6}$$

It is not possible to calculate the bayes classifier in real-life. We still need to estimate the parameters $\mu_1, ..., \mu_K, \pi_1, ..., \pi_K$ and σ^2. LDA method approximates the bayes classifier by plugging the estimates for π_k, μ_k and σ^2 into equation (1.4.6).

The following estimates are used:

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y_i=k} x_i \tag{1.4.7}$$

$$\hat{\sigma} = \frac{1}{n-K} \sum_{K=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu})^2 \tag{1.4.8}$$

$$\tag{1.4.9}$$

n: Number of training observations n_k: Total number of training observations in class k where μ_k is the average of all training observations from class k.

$\hat{\sigma}^2$: Weighted average of the sample variances for each of the k classes.

In the case where additional information is not present, LDA estimates π_k using the proportion of training observations that belongs to the k^{th} class.

$$\pi_k = \frac{n_k}{n} \tag{1.4.10}$$

LDA classifier plugs the estimates equation (1.4.8) and equation (1.4.9) into equation (1.4.6), assigning an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x\frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + log(\hat{\pi}_k) \tag{1.4.11}$$

is largest.

The classifier is linear due to the fact that the discriminant functions $\hat{\delta}_k(x)$ are linear functions of x Gareth, Daniela, and Trevor (2014).

I will use random forest for my analysis which i will discuss in depth in the subsequent chapter.

2. The Second Chapter

Methodology

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest(Breiman,2001). The goal of Random forest is creating a predictive model that predicts the value of a target variable based on given input variables where one of the input variable is represented by each interior node and the values of the input variable is represented by edges.

Bagging and Random forests

Bagging and random forests use trees as building blocks to constructing more powerful models Gareth, Daniela, and Trevor (2014).

Bootstrap: It is a widely used statistical tool used to quantify uncertainty associated with given estimators. It can easily be applied to a wide range of statistical learning methods even those whose measure of variability is difficult to obtain.

Bootstrap aggregation / Bagging: This is the basic principle behind the training algorithm for random forests which reduces the variance of a statistical learning method.

Consider the set of n independent observations denoted by $C_1, C_2, ..., C_3$ each with variance σ^2. Therefore the variance of the mean $\bar{Z}$ of the observation is given by $\frac{\sigma^2}{n}$. Averaging a set of observations reduces the variance. To reduce the variance and increase the prediction accuracy of a statistical learning method, we sample many training sets from the population, build a separate prediction model and average the resulting predictions. Therefore, calculate $\hat{f}^1(x), \hat{f}^2(x), ..., \hat{f}^B(x)$ using B separate training sets and average them so as to obtain a single low-variance statistical model given as:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x) \tag{2.0.1}$$

However, this is not practical because we cannot have multiple training sets so the bootstrap approach is used where repeated samples from the single training dataset are sampled.In this method, B different bootstrapped training datasets are generated and we train our method on the b^{th} bootstrapped training set to obtain $\hat{f}^{\star b}(x)$ and then average all the predictions to obtain:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{\star b}(x) \tag{2.0.2}$$

This is called **bagging**. (When trees are repeatedly fit to bootstrapped subsets of the observations)

So given a training set $X = x_1, ..., x_2$ with responses $Y = y_1, ..., y_n$, bagging repeats B times and selects random samples **with replacement** of the training set and fits trees to the sample. Trees are grown deep and are not pruned therefore each individual tree has high variance but low bias. Finding the average of the B trees reduces the variance.

How can bagging be extended to a classification problem where Y is qualitative? Given a test observation, a predicted class can be recorded by each of the B trees and a majority vote is recorded. The overall prediction is the most commonly occurring class among the B predictions Gareth, Daniela, and Trevor (2014).

Out of Bag Error Estimation

This is a method of estimating the test error of a bagged model without the need of cross validation. Averagely, each bagged tree makes use of around $\frac{2}{3}$ of the observation and the other $\frac{1}{3}$ of the observation is not used to fit a bagged tree. This observations are referred to as out of bag observation. The response for the i^th observation can be predicted using each of the tree in which that observation was out of bag which will yield around $\frac{B}{3}$ predictions for the i^th observation. To obtain a single prediction for the i^th observation, we take a majority vote of the predicted responses. This gives a single OOB prediction for the i^th observation.

The OOB prediction is obtained for the n observations and the classification error is computed. The OOB error is an estimate for the test error for the bagged model since each of the observation has the response predicted using only the trees that were not fit using that observation Gareth, Daniela, and Trevor (2014). Therefore, the OOB method for test error estimation is convenient when bagging on large datasets.

For each observation $Z_i = (x_i, y_i)$ we build a random forest predictor by averaging the trees corresponding to bootstrap samples in which z_i did not appear. The training is terminated when the error stabilizes.

Variable importance measures

When a large number of trees are bagged, it is no longer possible to represent the statistical learning procedure using a single tree, and it is not also clear which variables are most important to the procedure. Although the collection of bagged trees is difficult to interpret than a single tree, an overall summary of the importance of each predictor can be obtained using the gini index for bagging classification trees. At every tree split, the improvement in the split criterion is the measure of importance attributed to the splitting variable and is the accumulated over all the trees for each variable separately Hastie, Tibshirani, et al..

Gini index: This is the expected error rate of the system. Calculating the gini index for each attribute helps one to get the splitting attributes. The 'best' split according to the Gini gain criterion is the split with the largest Gini gain Strobl, Boulesteix, and Augustin (2007).

Random Forests

Random forest is an improvement over bagged trees by providing a small adjustment to the system that decorrelates the trees. In building the decision trees, **a random sample of m predictors is chosen as split candidates from the set of p predictors** each time a split in a tree is considered. The split is allowed to use only one of those m predictors.

The number of predictors considered at each split is approximately equal to the square root of the total number of predictors, $m \approx \sqrt{p}$. A new sample of m predictors is taken at each split.

Suppose there is one very strong predictor in the dataset and a number of other moderately strong predictors. Then most or all of the predictors will use the strong predictors in the top split in the collection of the bagged trees. Consequently, all of the bagged trees will look quite similar to each other and therefore the predictions from the bagged trees will be highly uncorrelated. Bagging will not lead to a reduction in the variance over a single tree in this setting Gareth, Daniela, and Trevor (2014).

How does random forest overcome this problem? By ensuring that each split considers only a subset of the predictors. Averagely, $\frac{(p-m)}{p}$ of the splits will not consider the strong predictor and the other predictors will have a chance. This is referred to as **decorrelating the trees**. This approach makes the average of the resulting trees less variable and more reliable.

The main difference between bagging and Random Forest is the choice of the predictor subset size m. If a random forest is built using $m = p$, this amounts to bagging. Random Forest using $m = \sqrt{p}$ leads to a reduction in **test error** and **OOB** over the bagging technique. It is helpful to use a small value of m when building a random forest if we have a large number of uncorrelated predictors. Random forest does not overfit just like bagging if we increase B Gareth, Daniela, and Trevor (2014).

Implementing the Random forest algorithm

- Loading the Data: The code loads the train(rawdata1) and test(test2) data into the jupyter notebook. The scope of the project is to predict the feature response which are the different categories of risk in the test data.

```
raw_data1=pd.read_csv('train.csv')  #loading the train data
test2=pd.read_csv('test.csv')  #loading the test data
```

- Shape of the data: The train data is composed of 59,381 observations and 128 features while the test data is composed of 19765 observations and 127 features.

```
raw_data1.shape
(59381, 128)
```

```
test2.shape
(19765, 127)
```

- To list the features in the data:

```
raw_data1.columns.values
```

- Factorize string variable: Product Info 2 is a string categorical variable, we transform this feature to enumerate type using the factorize function. The factorize functions returns a list of unique values (or categorical labels) in the product Info 2 column.

```
raw_data['Product_Info_2'] = pd.factorize(raw_data['Product_Info_2'])[0]
raw_data['Product_Info_2']
```

- Missing values: From sklearn we import imputer. Where the missing values is Nan, we choose the imputation strategy as mean and the axis is set to 0 meaning that we want to impute the mean values along the columns. The strategy can also be mode in case we replacing categorical missing values. We therefore choose the most occurring value or the median value along the axes.

```
imp=Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit_transform(raw_data,y='Response')
```

- Splitting the data into into train and test.

  From sklearn we import model selection which splits the dataset into random train and test subsets.

  Test size: Gives the proportion of the dataset that is included in the test split.

  Random state: Pseudo-random number generator state that is used for random sampling.

```
train_raw, test_raw=model_selection.train_test_split(raw_data,
test_size=0.4, random_state=100)
```

- To prepare the data for modelling, we drop the features 'Id' and 'Response'.

```
t1=train_raw.drop(0,axis=1)    #Train_raw dataset
t2= test_raw.drop(0,axis=1)     #Test_raw dataset
t1=t1.drop(127,axis=1)
t2=t2.drop(127,axis=1)
```

- Assigning the response and explanatory variables to numpy array.

```
ob=list(t1.columns)
def choose_columns(data):
    ret_X= np.array(data.loc[:,ob]) #Explanatory variables
    ret_Y=data.values[:,-1]
    return ret_X, ret_
```

1. We model the data using the Random Forest algorithm.

   From sklearn.ensemble we import the RandomForestClassifier.

```
import sklearn.ensemble as en
RF= en.RandomForestClassifier(n_estimators= 250, criterion='gini',
   max_depth=None,min_samples_split=2, min_samples_leaf=1,max_features='auto',
   max_leaf_nodes=None, min_impurity_split=1e-08,bootstrap=True,
   oob_score=False,n_jobs=1, random_state=None, verbose=0,warm_start=True)
```

Description of the parameters that I tuned for the best score (Web, Accessed May 2017).

n_estimators: The number of trees in the forest.

Criterion: "gini", measures the quality of a split.

max_depth: The maximum depth of the tree. Takes on integer values or none. If the value is none, then all nodes are expanded until all leaves are pure.

Min_samples_split: Minimum number of samples required to split an internal node.

Min_samples_leaf: The minimum number of samples required to be at a leaf node.

Max_features: Number of features to consider when looking for the best split. If it is auto, then the max features is equal to the sqrt(n_features).

Max_leaf_nodes: This parameter grows trees with max leaf nodes in the best first fashion.

Min_impurity_split: This is the threshold for early stopping in the tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

Bootstrap: It takes on boolean with default =True. If true, the algorithm makes use of bootstrap samples when building the trees.

Oob_score: It takes on boolean with default =True. If true, the algorithm uses the out of bag samples to estimate the generalization accuracy.

n_jobs: Default=1. The number of jobs that should run in parallel for bothfit and predict. If -1, then the number of jobs is set to the number of cores.

Random state: If none, it means that the random number generator is the random state instance used by np.random.

Verbose: Default=0. Controls the verbosity of the tree building process. That is if the verbose is set to a higher number, more information about the tree building process will be seen.

Warm_start: bool,(default=False). When set to true,the solution of the previous call to fit and add more estimators to the ensemble is reused. Otherwise, just fit a whole new forest.

- Fit the Random Forest classifier on the train data.
- Then predict the response feature for the test data that was split using the model selection code.
- The same feature transformations are done on the test data provided by kaggle. This is the data we are supposed to predict the response variable and evaluate the score using the quadratic weighted kappa metric.

Quadratic weighted kappa metric: Web (Accessed March 2017b) It can be used to quantify the amount of agreement between the predictions from an algorithm and some trusted labels of the same objects in machine learning. It is a chance adjusted index of agreement and measures the agreement between two ratings.

Calculation of quadratic weighted kappa metric.

- We construct an $N \times N$ histogram matrix O. O_{ij} corresponds to the number of applications which received a rating i by A and j by B.
- We calculate an $N \times N$ matrix of weights, w, based on the difference between the rater's score.

$$w_{ij} = \frac{(i-j)^2}{(N-1)^2} \tag{2.0.3}$$

- An $N \times N$ histogram matrix of expected ratings, E, is calculated, making the assumptions that there is no correlation between the rating scores. This is calculated as the outer product between each rater's histogram vector of ratings and normalized such that E and O have the same sum.
- The quadratic weighted kappa is calculated from these three matrices as:

$$\frac{1 - \sum_{ij} w_{ij} o_{ij}}{\sum_{ij} w_{ij} E_{ij}} \tag{2.0.4}$$

The metric works well for a highly imbalanced classification task. The metric varies from 0 (random agreement) to 1 (complete agreement). In case there is less agreement between the raters than expected by chance, this metric may go below 0. The data has 8 possible ratings and each application is characterized by a tuple (ea, eb), that corresponds to the scores by rater A, actual risk and rater B, predicted risk.

My predictions on kaggle scored **0.53074**.

Xgboost Algorithm

The full name is eXtreme Gradient Boosting. It is a variant of gradient boosting, a tree model based supervised learning algorithm. It includes an efficient linear model solver and a tree learning algorithm that supports a variety of objective functions including ranking, classification and regression Chen and He (2015). This boosting approach learns slowly unlike fitting a large decision tree to the data which likely amounts to overfitting the data.

Features of Xgboost

- Customization: Xgboost supports customized objective function and evaluation function.
- Sparcity: Xgboost accepts sparse input for both the tree and linear booster, and is optimized for sparse input.
- Input type: Xgboost takes several types of input data. The recommended type is xgb.Dmatrix.
- Speed: It can automatically do parallel computation and faster than gradient boosting machine.
- Performance: Has better performance on different datasets.

XG Boost paramers

The parameters can be grouped into (He):

1 General parameters.

Under general parameters we have number of threads.

2 Booster parameters.

- Stepsize.
- Regularization.

3 Task parameters.

- Objective.
- Evaluation metric.

Model Specification

Training objective.

This model is a collection of decision trees. Supposing we have K trees, the model is given by (He):

$$\sum_{k=1}^{K} f_k. \tag{2.0.5}$$

With all the prediction trees, we predict by:

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i) \tag{2.0.6}$$

x_i: Feature vector of the i^th data point.
The prediction at the t^th step can be defined as:

$$\hat{y_i t} = \sum_{k=1}^{t} f_k(x_i) \tag{2.0.7}$$

To train the model, we need to optimize a loss function (He). We use;

- Rooted mean squared error for regression.

$$-L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2. \tag{2.0.8}$$

- Logloss for binary classification.

$$-L = -\frac{1}{N} \sum_{i=1}^{N} (y_i log()p_i) + (1 - y_i) log(1 - p_i). \tag{2.0.9}$$

- mlogloss for multi-classification.

$$-L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} log(p_{i,j}). \tag{2.0.10}$$

Regularization is an important part of the model. The regularization term controls the complexity of the model thereby preventing overfitting.

$$\Omega = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} wj^2. \tag{2.0.11}$$

λ: Regularization parameter.
γ: Minimum loss reduction required to make a further partition on a leaf node of a tree.
T: The number of leaves.
wj^2: The score of the j^th leaf.
Combing the loss function and the regularization function, we get the objective function of the model.

$$obj = L + \Omega. \tag{2.0.12}$$

where the loss function controls the predictive power and regularization controls the simplicity (He).

General parameters.

- nthread: Number of parallel threads.
- Booster:
    - gblinear: linear function.
    - gbtree: tree based model.

xgb.Dmatrix: This is the data structure used by the XGBoost algorithm. XGBoost prepossess the input data and labels it into an xgb.Dmatrix object before implementing it to the training algorithm.

An xgb.DMatrix contains:

- Prepossessed training data.
- Missing values.
- Data weight.

If the training process is to be repeated on the same data set, the xgb.DMatrix works well as it saves on the prepossessing time (He).

Implementing the XGBoost algorithm (Web, Accessed March 2017c).

-
  ```
  def eval_wrapper(yhat, y):
      y = np.array(y)
      y = y.astype(int)
      yhat = np.array(yhat)
      yhat = np.clip(np.round(yhat), np.min(y), np.max(y)).astype(int)
      return quadratic_weighted_kappa(yhat, y)
  ```

  This function calculates the quadratic weighted kappa metric.

  np.clip: Takes three arguments (np.clip(a, a min, a max, out=None)). It clips (limits) the values in an array. Given an interval, values outside the interval are clipped to the interval edges.

  np.round: (a,decimals=0,out=None). Evenly rounds to the given number of decimal places.

-
  ```
  def get_params():

      params = {}
      params["objective"] = "reg:linear"
      params["eta"] = 0.05
      params["min_child_weight"] = 360
      params["subsample"] = 0.85
  ```

```
    params["colsample_bytree"] = 0.3
    params["silent"] = 1
    params["max_depth"] = 7
    plst = list(params.items())

    return plst
```

This code sets up a dictionary of parameters for the tree booster. Objective

- ”reg:linear”: default option, Linear regression.
- ”binary: Logistic”: Outputs probability. It performs logistic regression for binary classification.
- ”multi:softmax”: Uses the softmax objective for multiclass classification.

Eta/Learning rate: We can directly get weights of new features after each boosting step. Eta shrinks the feature weights and makes the boosting process more conservative. Eta is the stepsize shrinkage used in update to prevent overfitting. It is in the range of [0,1], the default is 0.3.

Min_child_weight: This is the minimum sum of instance weight needed in a child. It ranges from [0,∞] and the default is 1.

The tree building process will give up further partitioning if the tree partition step results in a leaf node with the sum of instance weight less than the Min_child_weight.

Subsample: It is the subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly samples half of the data instances and grows trees thus prevents overfitting. It ranges from (0,1], the default value being 1. This parameter makes the model more robust and avoids overfitting.

colsample_bytree: This is the subsample ratio of columns when constructing each tree. The range is (0,1] and the default is 1. Both subsample and colsample_bytree cannot be set to 0.

Silent: The default=0. 0 means printing running messages, 1 means silent mode.

max_depth: This is the maximum depth of a tree. Increasing the max_depth value makes the model more complex and more likely to overfit. The range is from [1,∞], the default is 6.

-
  ```
  def apply_offsets(data, offsets):
      for j in range(num_classes):
          data[1, data[0].astype(int)==j] = data[0, data[0].astype(int)==j] +
          offsets[j]
      return data
  ```

This function applies an offset to selected predictions that are generated from the XGBoost model. In data[0], we have the original prediction values while in data[1], we have the same predictions where the offset is applied. There are a total of 8 offsets stored in the offsets list variable. These offsets apply to predictions that have as their integer value, matching the position of the offset in the offset list. Predictions generated from XGBoost are offset to a value that increases the score.

-
  ```
  # global variables
  columns_to_drop = [’Id’, ’Response’]
  xgb_num_rounds = 720
  num_classes = 8
  missing_indicator = -1000
  ```

  We drop the columns Id and Response so as to prepare the data for modelling. xgb_num_rounds is the number of times we train the model.

-
  ```
  train = pd.read_csv("train.csv")
  test = pd.read_csv("test.csv")
  ```

  Load the train and test data.

-
  ```
  all_data = train.append(test)
  all_data.shape
  ```

  Combining the train and test data.

-
  ```
  all_data[’Product_Info_2_char’] = all_data.Product_Info_2.str[0]
  all_data[’Product_Info_2_num’] = all_data.Product_Info_2.str[1]
  ```

  Creating new columns of Product_Info_2 where one column is a string variable and the other column is a numeric variable.

-
  ```
  all_data[’Product_Info_2’] = pd.factorize(all_data[’Product_Info_2’])[0]
  all_data[’Product_Info_2_char’] =pd.factorize(all_data[’Product_Info_2_char’])[0]

  all_data[’Product_Info_2_num’] = pd.factorize(all_data[’Product_Info_2_num’])[0]

  all_data[’BMI_Age’] = all_data[’BMI’] * all_data[’Ins_Age’]

  med_keyword_columns=all_data.columns[all_data.columns.str.startswith
  (’Medical_Keyword_’)]

  all_data[’Med_Keywords_Count’] = all_data[med_keyword_columns].sum(axis=1)
  ```

  We feature engineer our data by factorizing the string variables into numeric variables.

  Interaction is defined as how the overall effect on the response of one explanatory variable is dependent on the level of one or more explanatory variables (Fitzmaurice, 2000). That is interaction occurs when the effect of one explanatory variable is dependent on a certain level of another explanatory variable .

  If no interaction is observed between two explanatory variables, then the overall effect of one explanatory variable is constant across all values of the other. BMI is related to age in that a higher BMI is found among the older age group and lower among the young age group, reducing thereafter in younger age groups Yanai, Kon, Kumasaka, and Kawano

(1997). Therefore, there is an interaction between age and BMI on the response variable. We therefore include the interaction term in the data.

-
  ```
  all_data.fillna(missing_indicator, inplace=True)
  all_data['Response'] = all_data['Response'].astype(int)
  ```

  The missing values in the data are filled with the value -1000 and the Response column which is float is converted to integer. Using a value that is not in the range of the data to fill in the missing data allows the model to split between the rest of the data and the missing data.

-
  ```
  train = all_data[all_data['Response']>0].copy()
  test = all_data[all_data['Response']<1].copy()
  ```

  This code splits the train and test data from all_data.

-
  ```
   xgtrain = xgb.DMatrix(train.drop(columns_to_drop, axis=1),
              train['Response'].values, missing=missing_indicator)
  xgtest = xgb.DMatrix(test.drop(columns_to_drop, axis=1),
              label=test['Response'].values,missing=missing_indicator)

  ```

  This code converts the data to xgb data structure.

-
  ```
   plst = get_params()
  model = xgb.train(plst, xgtrain, xgb_num_rounds)

  ```

  We train the model by specifying the parameters and the number of times to train the model.

-
  ```
  train_preds = model.predict(xgtrain)
  print('Train score is:', eval_wrapper(train_preds, train['Response']))
  test_preds = model.predict(xgtest)
  ```

  We get the train and test predictions and calculate the score on the train data.

-
  ```
  offsets=np.array([-1.5,-2.6,-3.6,-1.2,-0.8,-0.1,0.6,3.6])
  offset_preds = np.vstack((train_preds, train_preds, train['Response'].values))
  offset_preds = apply_offsets(offset_preds, offsets)
  print('Offset Train score is:', eval_wrapper(offset_preds[1], train['Response']))
  ```

  The offsets are generated after optimization by the fmin_powell function. The train score is evaluated by the eval_wrapper function which outputs the quadratic weighted kappa value.

-
  ```
  def score_offset(data, bin_offset, sv, scorer=eval_wrapper):
      data[1, data[0].astype(int)==sv] = data[0, data[0].astype(int)==sv] +
      bin_offset
      score = scorer(data[1], data[2])
      return score
  ```

The score _offset code gets the value from data[0], which is the original prediction and where we will apply the offsets. The line astype(int)== sv is the subset of the array where the prediction value matches the sv. Here, the sv is the position of the offset in the given offset's list. The offset is then applied ie the code ('+ bin_offset') and the result stored in the corresponding offset prediction (data[1,data[0].astype(int)==sv]) Data[1] is the prediction where an offset can be applied whereas the values in data[2] are the labels against which the offsets are scored.

```
from scipy.optimize import fmin_powell
opt_order = [0,1,2,3,4,5,6,7]
for j in opt_order:
    train_offset = lambda x: -score_offset(offset_preds, x,j)
    offsets[j] = fmin_powell(train_offset, offsets[j], disp=False)
    print (offsets[j])
```

The code train_offset = lambda x: -score_offset(offset_preds, x,j) sets the value for the data and sv variables in the score offset function and leaves the bin_offset variable to be optimized. This code allows for the optimization of the score_offset function by the fmin_Powell function. The kappa metric works well with larger numbers while the fmin_Powell function optimizes to the smallest value.

fmin_Powell function: Minimizes a function using the modified Powell method by using a modified Powell directional search algorithm to find the minimum of a function of one or more variables.

-
  ```
  # apply offsets to test
  data = np.vstack((test_preds, test_preds, test['Response'].values))
  data = apply_offsets(data, offsets)
  ```

  We apply the offsets to the test data.

- `final_test_preds = np.round(np.clip(data[1], 1, 8)).astype(int)`

  the values in data[1] where offsets have been applied to are clipped to the interval edges and the rounded off to integer values. The final test predictions are submitted to kaggle for scoring.

  The predictions scored **0.66289**.

3. Third Chapter

3.1 Results and Discussions

3.1.1 Data Description. .

The data provided is divided into train and test data set (Web, Accessed March 2017a). There is a sample submission file that gives instructions on how predictions are submitted on the kaggle website. The train data is composed of 53,381 entries(clients) and a response variable. The total number of features is 128. The test data is composed of 19,765 entries(clients) and a total of 127 features. The predictions range from 1 to 8 where 1 represents the lowest risk level and 8 represents the highest risk level.

Variable	Description
Id	A unique value associated with an application.
Product_Info_1-7	A set of normalized variables relating to the product a client applied for
Ins_Age	Normalized age of applicant
BMI	Normalized BMI of applicant
Wt	Normalized weight of applicant
Ht	Normalized height of applicant
InsuredInfo_1-6	Normalized variables providing information about the applicant.
Employment_Info_1-6	Normalized variables relating to the employment history of the applicant.
Family_Hist_1-5	Normalized variables relating to the family history of the applicant.
Insurance_History_1-9	Normalized variables relating to the insurance history of the applicant.
Medical_Keyword_1-48	Dummy variables relating to the presence of/absence of a medical keyword being associated with the application.
Medical_History_1-41	Normalized variables relating to the medical history of the applicant.
Response	This is the target variable, an ordinal variable relating to the final decision associated with an application

The following variables are all categorical (nominal):

Product_Info_1, Product_Info_2, Product_Info_3, Product_Info_5, Product_Info_6, Product_Info_7, Employment_Info_2, Employment_Info_3, Employment_Info_5, InsuredInfo_1, InsuredInfo_2, InsuredInfo_3, InsuredInfo_4, InsuredInfo_5, InsuredInfo_6, InsuredInfo_7, Insurance_History_1, Insurance_History_2, Insurance_History_3, Insurance_History_4, Insurance_History_7, Insurance_History_8, Insurance_History_9, Family_Hist_1, Medical_History_2, Medical_History_3, Medical_History_4, Medical_History_5, Medical_History_6, Medical_History_7, Medical_History_8, Medical_History_9, Medical_History_11, Medical_History_12, Medical_History_13, Medical_History_14, Medical_History_16, Medical_History_17, Medical_History_18, Medical_History_19, Medical_History_20, Medical_History_21, Medical_History_22, Medical_History_23, Medical_History_25, Medical_History_26, Medical_History_27, Medical_History_28, Medical_History_29, Medical_History_30, Medical_History_31, Medical_History_33, Medical_History_34, Medical_History_35, Medical_History_36, Medical_History_37, Medical_History_38, Medical_History_39, Medical_History_40, Medical_History_41

The following variables are discrete:

Medical_History_1, Medical_History_10, Medical_History_15, Medical_History_24, Medical_History_32 Medical_Keyword_1-48 are dummy variables.

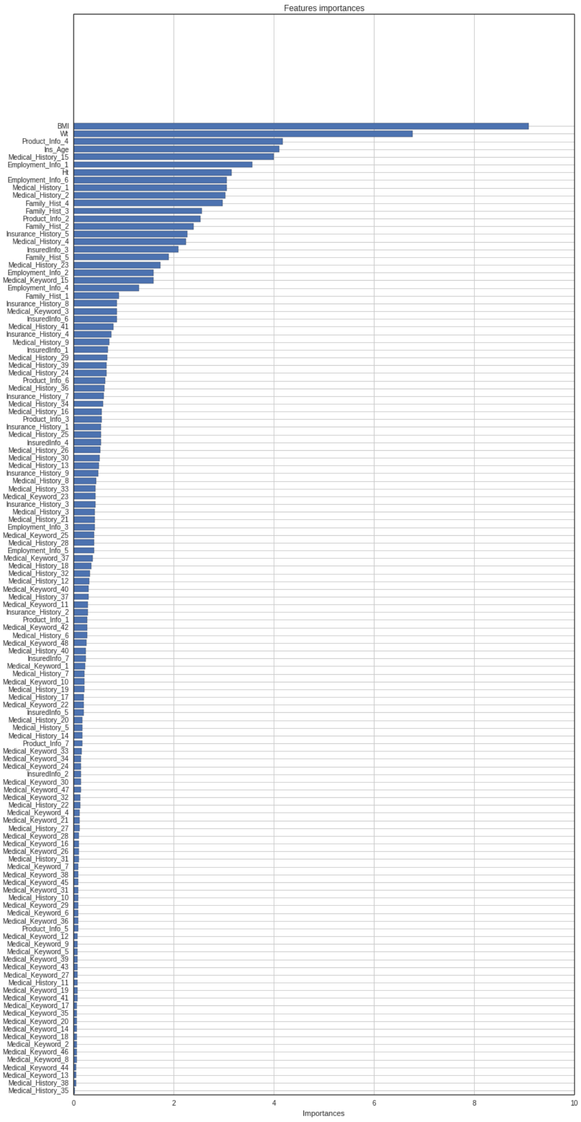
The following variables are continuous:

Product_Info_4, Ins_Age, Ht, Wt, BMI, Employment_Info_1, Employment_Info_4, Employment_Info_6, Insurance_History_5, Family_Hist_2, Family_Hist_3, Family_Hist_4, Family_Hist_5

Data Visualization

A plot of the feature importances lists the most important features in descending order.

Figure 3.1: Feature Importance



The figure above is a variable importance plot for the insurance data. Variable importance is computed using the mean decrease in the gini index and expressed relative to the maximum.

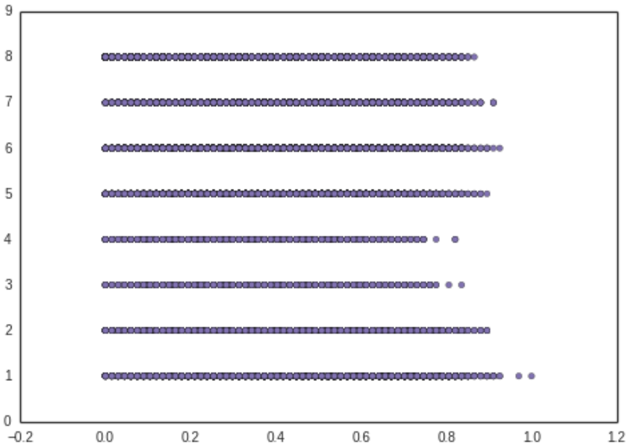
The code below outputs the five most important and least important variables in ascending order respectively.

```
importances =pd.DataFrame({'features' :m.columns,
                          'importances' : RF.feature_importances_})
importances.sort_values(by='importances',ascending=False).head(5)
importances.sort_values(by='importances',ascending=False).tail(5)
```

The first 5 important predictors	
BMI	0.089700
Wt	0.068185
Product_info_4	0.041591
Ins_Age	0.040372
Medical_History_15	0.039763
The last 5 less important predictors	
Medical_History_35	0.000219
Medical_History 38	0.000512
Medical_History_13	0.000551
Medical_History_44	0.000612
Medical History 8	0.000622

The scatter plot below shows the relationship between the various responses and the normalized Ins Age.

Figure 3.2: Scatter plot of Age Vs Response



The risk prediction classes are evenly distributed across the ages of the clients with a few outliers.

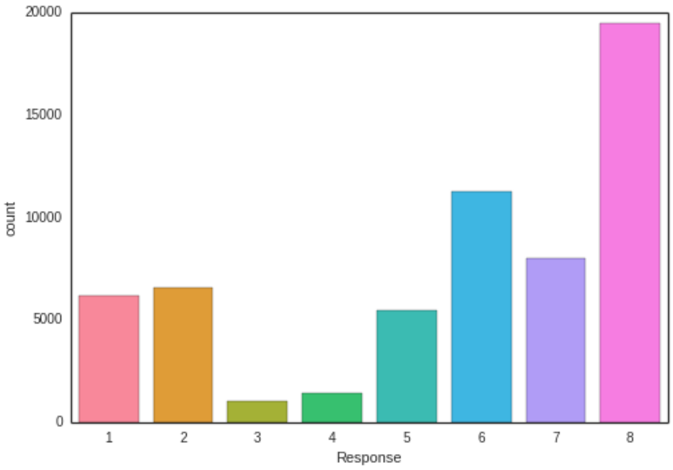
Descriptive analysis of the Response feature.

count	59381.000000
mean	5.636837
std	2.456833
min	1.000000
25 %	4.000000
50 %	6.000000
75 %	8.000000
max	8.000000

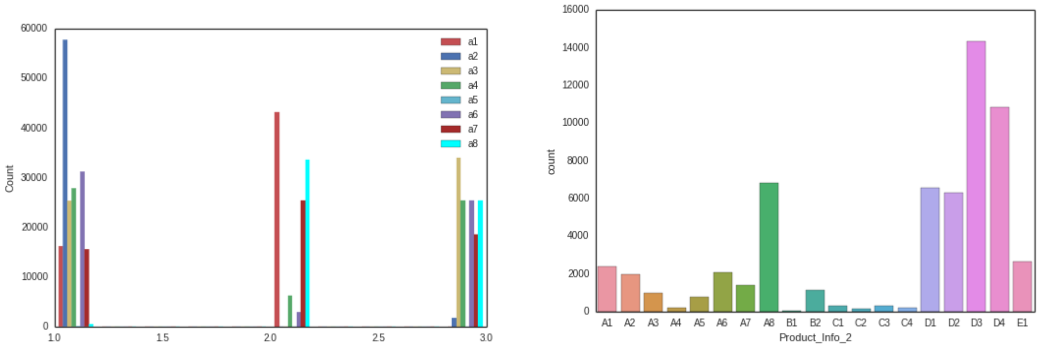
The mean risk prediction is a roughly 6.

The response classes are imbalanced as shown in the plot below.

Figure 3.3: Class Imbalance of the response variable



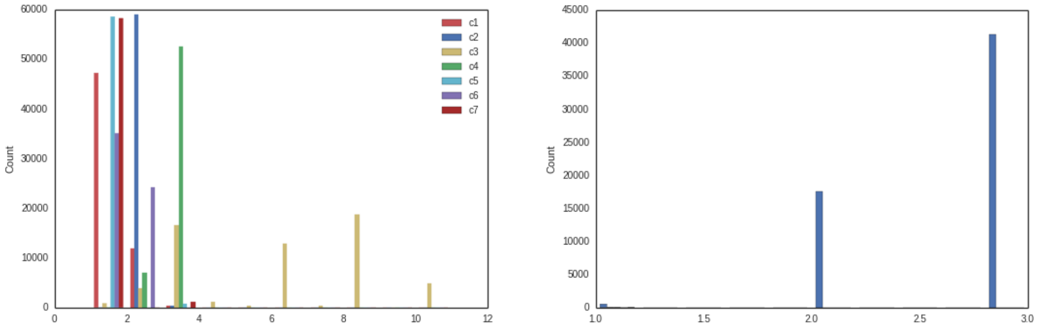
Most clients fall under the risk class 8.



(a) Insurance_history plot (b) Product_Info_2

Figure 3.4: Feature data plots

Plot(a): Insurance History_2, 3, 4, 7, 8 and 9 take on the categorical values 1, 3, 2. Insurance History_1 takes on the categorical values (1, 2) while Insurance History_5 takes on a range of values almost close to 0. Plot(b) shows the distribution of the categorical variable Product_Info_2 where the product D3 was highly preferred.

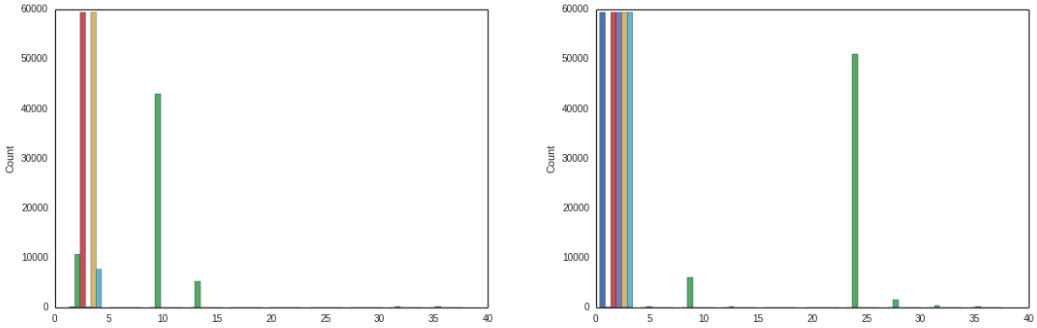


(a) Insurance information plot

(b) Family history plot

Figure 3.5: Feature data plots

Plot(a) InsuredInfo_1 takes on the categorical values (1, 2, 3), InsuredInfo_2 and 4 takes the values (2, 3). InsuredInfo_5 and 6 takes the values (1, 3). InsuredInfo_6 takes the values (2, 1). InsuredInfo_3 takes on a range of values. Plot(b) Family_Hist_1 takes on the values (2, 3, 1) while the rest of the features take on values close to 0.



(a) Employment information plot

(b) Insurance history plot

Figure 3.6: Feature data plots

Plot(a): Product_Info_2, 5, 6, and 7 takes on the categorical variables (1, 2), (2, 3), (1, 3)and (1, 3, 2) respectively. Product_Info_3 and 4 take on different values. Plot(b): Employment_Info_3 and 5 take on categorical values (1, 3) and (3, 2) respectively . Employment_Info_1, 2, 4 and 6 have no unique values.

Using the random forest algorithm to predict the risk levels of the test data, the model scored 0.53074.

Weakness of Random forest

- Random forest may overfit noisy datasets: If the number of variables is large and the fraction of relevant variables small, this algorithm tends to perform poorly with small m.

At each split the chance can be small that the relevant variables will be chosen Hastie, Tibshirani, et al..

- Having a large number of trees makes the algorithm slow for real time prediction

To improve my score on kaggle, I employed the Xgboost algorithm whose full name is eXtreme Gradient Booosting. It is a variant of gradient boosting, which is a tree model based supervised learning algorithm. Unlike fitting a single large decision tree to the data, which could amount to overfitting, the boosting approach instead learns slowly. It includes an efficient linear model solver and a tree learning algorithm.

XGBoost proved to be a reliable large scale tree boosting system as the model scored 0.66289 using the quadratic weighted kappa metric, an improvement from Random forest algorithm.

4. Conclusion

4.1 Guide on parameter tuning

Tuning: It is defined as choosing the best parameters to optimize the performance of an algorithm.

It is really not possible to give a universal set of accepted parameters that can optimize an algorithm. Parameters have to be tuned to achieve good results.

Key points on parameter tuning

- To control overfitting.
- To deal with imbalanced data.
- To trust the cross validation.

The bias-variance trade off is the most important concept in controlling overfitting and applies to both classification and regression problems. This trade off elaborates why we have no universal optimal learning method for algorithms. Basically, finding an optimal bias-variance tradeoff is difficult. Despite this, acceptable solutions can be found, e.g., use of cross validation or regularization (Sethu, 2007).

Bias-variance trade off: This is the difficulty experienced in reducing sources of error arising from erroneous assumptions in the algorithm resulting to missing out of relevant relationships between the target variable and the features whereas variance is the error arising from the sensitivity and small fluctuations in the training sets which causes overfitting. The two types of errors makes it difficult for a supervised learning to generalize unseen data (Sethu, 2007).

How do we counter the bias-variance trade off in XGBoost?

We can group the booster specific parameters as below and tune the given parameters accordingly (He).

- To control the model complexity: max_depth, min_child_weight and gamma.
- Robust to noise: subsample, colsample_by tree.

How to deal with imbalanced data among classes.

If one is interested in a model that can only predict the right probability, then the dataset cannot be rebalanced and therefore set the parameter max_delta_step to a finite number (say 1). This will help in convergence.

On the other hand, if one is interested in a model that ranks, then balance the negative and positive weights by the scale_pos_weight parameter and use the "auc" as the evaluation metric.

Use early.stop.round to detect if the model is continuously getting worse on the test set.

Reduce the step size eta and increase nround simultaneously if overfitting is observed (He).

How does one build a winning algorithm in a kaggle competition?

To score highly on kaggle, one needs to consistently focus on:

- Parameter tuning.
- Model ensembling.
- Feature engineering.

(He) Why XGBoost is the winning algorithm for kaggle competitions.

- It is efficient as it allows for parallel computing and can be run on a cluster.
- It is accurate: It outputs good results for most datasets.
- Feasibility: It provides a platform for tunable parameters.
- XGBoost is easy to use and install with a highly developed R and python interface.

4.2 Future outlook.

To increase the accuracy of a machine learning algorithm, model ensembling is a vital technique.

Ensemble methods are learning algorithms that construct a set of classifiers and then classifies new data points by taking a (weighted) vote of their predictions. Recent algorithms include error-correcting output coding, bagging and boosting.

Ensembles can be created from:

- Submission files.
- Stacked generalization/blending.

Creating ensembles from submission files.

This method ensembles kaggle csv submission files. It is a quick way of ensembling already existing model predictions as one only needs the predictions on the test set and the model is not retrained.

Model ensembling reduces the error rate. Ensembling low correlated model predictions works better that is, there is an increase in the error-correcting ability if there is a lower correlation between model members.

Creating ensembles from stacking multiple learning models.

This is an ensemble method where models are combined using another machine learning algorithm eg logistic regression, train the machine learning algorithm with the training dataset thereby generating a new dataset using these models. The new generated dataset is used as the input for the combiner machine algorithm.

It is worth noting that the same training dataset is not used for prediction. So to overcome this, the training dataset is split before training the base algorithm. Stacking reduces the generalization error/ out-of-sample error which is a measure of how accurately unseen data can be predicted by an algorithm.

Appendix

An average essay may contain five chapters, but I didn't plan my work properly and then ran out of time. I spent too much time positioning my figures and worrying about my preferred typographic style, rather than just using what was provided. I wasted days bolding section headings and using double slash line endings, and had to remove them all again. I spent sleepless nights configuring manually numbered lists to use the LaTeX environments because I didn't use them from the start or understand how to search and replace easily with texmaker.

Everyone has to take some shortcuts at some point to meet deadlines. Time did not allow to test model B as well. So I'll skip right ahead and put that under my Future Work section.

4.3 This is a section

Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Some essays may have 3, 5 or 6 chapters. This is just an example. More importantly, do you have at most 35 pages? Luck has nothing to do with it. Use the techniques suggested for writing your essay.

Now you're demonstrating pure talent and newly acquired skills. Perhaps some persistence. Definitely some inspiration. What was that about perspiration? Some team work helps, so every now and then why not browse your friends' essays and provide some constructive feedback?

References

Prudential life insurance assessment. Webots, https://www.kaggle.com/c/prudential-life-insurance-assessment/data, Accessed March 2017a.

Prudential life insurance assessment. Webots, https://www.kaggle.com/c/prudential-life-insurance-assessment#description, Accessed March 2017b.

Prudential life insurance assessment. Webots, https://www.kaggle.com/zeroblue/xgboost-with-optimized-offsets/output, Accessed March 2017c.

3.2.4.3.1.sklearn.ensemble.randomforestclassifier. Webots, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#examples-using-sklearn-ensemble-randomforestclassifier, Accessed May 2017.

Michael J Berry and Gordon Linoff. *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc., 1997.

Tianqi Chen and Tong He. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 2015.

David CM Dickson, Mary R Hardy, and Howard R Waters. *Actuarial mathematics for life contingent risks*. Cambridge University Press, 2013.

Garrett Fitzmaurice. The meaning and interpretation of interaction. *Nutrition*, 16(4):313–314, 2000.

Edward W Frees, Richard A Derrig, and Glenn Meyers. *Predictive modeling applications in actuarial science*, volume 1. Cambridge University Press, 2014.

James Gareth, Witten Daniela, and Hastie Trevor. An introduction to statistical learning: With applications in r., 2014.

Gavin Hackeling. *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd, 2014.

Trevor Hastie, Robert II Tibshirani, et al. The elements of statistical learning: data mining, inference, and prediction/by trevor hastie, robert tibshirani, jerome frieman. Technical report.

Stan Hatko. The bank of canada 2015 retailer survey on the cost of payment methods: Nonresponse. Technical report, Bank of Canada Technical Report, 2017.

Tong He. Xgboost extreme gradient boosting. Technical report.

Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

Lionel Macedo. The role of the underwriter in insurance. *Primer Series on Insurance,(8)*, 1, 2009.

Charles Nyce and API CPCU. Predictive analytics white paper. *American Institute for CPCU. Insurance Institute of America*, pages 9–10, 2007.

pred. Predictive modelling for life insurance. www.soa.org/files/pdf/research-pred-mod-life-batty.pdf, Accessed April 2017.

Vijayakumar Sethu. Computational learning theory., 2007.

Carolin Strobl, Anne-Laure Boulesteix, and Thomas Augustin. Unbiased split selection for classification trees based on the gini index. *Computational Statistics & Data Analysis*, 52(1): 483–501, 2007.

M Yanai, A Kon, K Kumasaka, and K Kawano. Body mass index variations by age and sex, and prevalence of overweight in japanese adults. *International Journal of Obesity & Related Metabolic Disorders*, 21(6), 1997.