PREDICTIVE MODELLING IN LIFE INSURANCE.

By

JULIANA TULA TALAI (juliana.talai@aims.ac.rw)

June 2017

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: Juliana Tula Talai

Scan your signature

Supervisor: Dr. Yabebal Fanataye

ACKNOWLEDGEMENTS

Abstract

The predictive modelling concept is a commonly accepted framework for building insurance models. It involves using data in forecasting future unseen events, capturing and exploiting the relationships between the explanatory variables and the target variable, which we intend to predict. The introduction sheds light on the conditions and setbacks that have made predictive modelling in life insurance lag behind other insurance sectors.

The aim of this research is to understand the steps involved in building a predictive model and make a comparison between two known algorithms, the Random Forest and the XGBoost. We then explore why XGBoost is a better algorithm in predicting risk as compared to Random Forest.

Contents

Appendix 34

References 36

1. Introduction

Insurance is a contract between an individual (insured) and the insurance company (insurer) to pay the individual a specific sum once the loss being covered occurs. The individual in turn pays a specific amount of payment periodically, premium, during the contract term. Traditionally, insurance companies estimated the appropriate fees to charge or determined the cost of insurance by analysing a single factor such as age of the insured. Basically, the process of determining risk depended on how accurately the insurance company is able to forecast a policy's ultimate cost, how to price its products to maximize profits and in turn avoid losses. With changing technology, insurance operations have become more advanced and analysis methods have shifted from univariate analysis to multivariate analysis where several factors like characteristics of the area where the risk is located, what other policies the insured has, medical history among others are used to determine appropriate premiums Nyce and CPCU (2007).

Modern day insurers use predictive analytics to determine additional information that may correlate with a potential insurance outcomes. This has been made possible due to the large quantity of data generated by insurers. The scientific guidance from predictive models has assisted insurers make expert decisions in claim management, premium audit, underwriting and fraud detection Mike Batty, 2017.

1.0.1 Definition. Predictive modelling 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. It 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, a process known as rate making and avoid adverse selection Frees et al. (2014).

The advantages of predictive analytics are: new premium growth, improved underwriting efficiency, accurate pricing, reduction of fraud and improved customer retention. However, this technique entails some disadvantages like cost of implementation is too high, the need for clean, accurate data, inadequate preprocessing of the data, over fitting the model to the existing data and inadequate model validation.

To overcome this problems, the foundation for an effective predictive model has been laid down by Kuhn and Johnson (2013). They include: Deep knowledge of the problem context, relevant data, intuition, adequate techniques for data preprocessing, visualization as well as modelling tools for handling a range of possible scenarios.

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 Macedo (2009). 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.

The requirements for building a predictive model as stipulated by Mike Batty, 2017 are: 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 and a large number of observations to build the model so as surfacing relationships can be separated from random noise.

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 (Mike Batty, 2017). To clearly understand the challenges faced in life insurance, we compare life underwriting and auto underwriting (Mike Batty, 2017).

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

We are working on data from Prudential Life Insurance where the challenge is trying to make purchasing of life insurance easier by understanding the algorithms and procedures used in building a predictive model that accurately classifies risk using a more automated approach. We compare the Random Forest and XGboost algorithms. The data we are 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).

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 Dickson, Hardy, and Waters (2013) .

Disadvantages of Underwriting.

1. 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.

2. The underwriting process could be lengthy and costly.

3. 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:

1. 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.

2. 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.

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

1.4.1 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.

1.4.2 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.

1.4.3 Regression versus 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).

209 **1.4.4 Machine Learning.** : This is a method of teaching computers to improve and make
210 predictions based on data. It is teaching a program to react to and recognize patterns through
211 analysis, self training, observation and experience (Hackeling, 2014).

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

215 1.5   Classifiers

216 We introduce some of the most commonly used classifiers.

217 **1.5.1 Logistic Regression.**
218 Models the probability that Y, the dependent variable belongs to a particular category (one of
219 two categories eg. 'yes' or 'no').

220   The model

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

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

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

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

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

231 Manipulating the equation gives;

$$\frac{p(X)}{1 - p(X)} = \exp(\beta_0 + \beta_1 X) \tag{1.5.1}$$

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

$$log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X \tag{1.5.2}$$

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

235 **1.5.2 K-nearest Neighbors.**

236 KNN can handle both binary and multi-class data.

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

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

Figure 1.1: KNN approach using K=3.
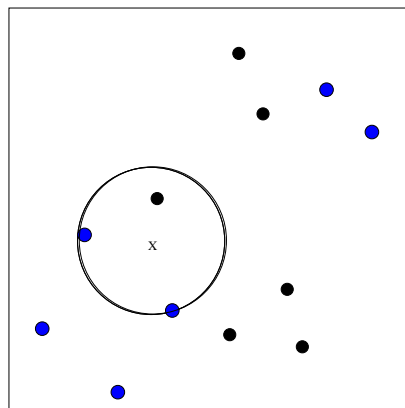
242 The goal is to predict the point labelled X. Suppose we choose $K = 3$, KNN will first identify
243 the three observations closest to X, as shown in the diagram. The point consists of two blue
244 points and a black point resulting in estimated probabilities of $\frac{2}{3}$ for the blue class and $\frac{1}{3}$ for the
245 black class. KNN will predict that the point X belongs to the blue class.

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

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

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

255 1.5.3 Linear Discriminant Analysis.

256 Linear Discriminant Analysis is used when we have more than two response classes Gareth,
257 Daniela, and Trevor (2014).

258   Using bayes' theorem for classification

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

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

263 $f_k = Pr(X = x/Y = k)$: Denote the density function of X for an observation from class K.

264 Bayes theorem states that:

$$P_k(X) = Pr(Y = k/X = x) = \frac{\pi_k f_k(x)}{\sum \pi_i f_i(x)} \qquad (1.5.3)$$

265 $P_k(x)$: Posterior probability that an observation $X = x$ belongs to class K.

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

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

270 We are required to find an estimate for $f_k(x)$ that we can plug into (1.5.3) inorder to estimate
271 $p_k(x)$. We then classify an observation for which $p_k(x)$ is greatest. Gareth, Daniela, and Trevor
272 (2014) To estimate $f_k(x)$ the following assumptions are made:

273   1. $f_k(x)$ is normal.

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right) \qquad (1.5.4)$$

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

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

276 Plugging equation (1.5.4) to equation (1.5.3) yields:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{i=1}^{K} \pi_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_i)^2\right)} \qquad (1.5.5)$$

277   Taking the logs of equation (1.5.5) and rearranging the terms gives:

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k) \tag{1.5.6}$$

278   It is not possible to calculate the bayes classifier in real-life. We still need to estimate the param-
279   eters $\mu_1, ..., \mu_K, \pi_1, ..., \pi_K$ and σ^2. LDA method approximates the bayes classifier by plugging
280   the estimates for π_k, μ_k and σ^2 into equation (1.5.6).

281   The following estimates are used:

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y_i=k} x_i \tag{1.5.7}$$

$$\hat{\sigma} = \frac{1}{n-K} \sum_{K=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu})^2 \tag{1.5.8}$$

$$\tag{1.5.9}$$

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

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

285   In the case where additional information is not present, LDA estimates π_k using the proportion
286   of training observations that belongs to the kth class.

$$\pi_k = \frac{n_k}{n} \tag{1.5.10}$$

287   LDA classifier plugs the estimates equation (1.5.8) and equation (1.5.9) into equation (1.5.6),
288   assigning an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x\frac{\hat{\mu_k}}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + log(\hat{\pi}_k) \tag{1.5.11}$$

289   is largest.

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

2. Methodology

We employed the Random Forest and XGboost algorithm to predict the risk classes of the test data. The evaluation metric used to score how accurately the models were able to classify the risk classes is the quadratic weighted kappa metric, which is calculated by kaggle once the submission of predictions is made to the website. The training set comprises of the data we will use to build the model while the test or validation set is used for evaluating the performance of a final set of candidate model.

2.1 Quadratic weighted kappa metric

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 Web (Accessed March 2017b).

Calculation of quadratic weighted kappa metric.

1. 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.

2. We calculate an $N \times N$ matrix of weights, w, based on the difference between the rater's score.

$$w_{ij} = \frac{(i-j)^2}{(N-1)^2} \tag{2.1.1}$$

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.

4. The quadratic weighted kappa is calculated from these three matrices as:

$$\frac{1 - \sum_{ij} w_{ij} o_{ij}}{\sum_{ij} w_{ij} E_{ij}} \tag{2.1.2}$$

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.

9

2.2 Overview of Random Forest

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.

Figure 2.1: Structure of a decision tree
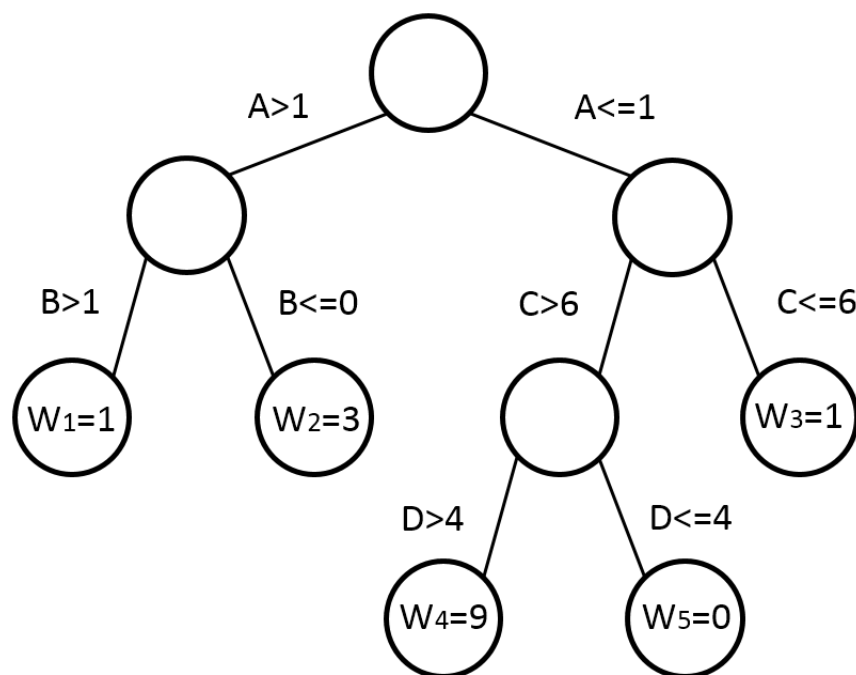
The internal nodes split the flow of the data points by one of the features where the condition on the edge specifies what data can flow through. Data points which reach to a leaf will be assigned a weight. The weight is the prediction. Every time a split is made, we convert a leaf to an internal node.

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

Bootstrapping is a widely used statistical tool used to quantify uncertainty associated with given estimators. Suppose a model is fit to a set of training data. We denote the training set by $Z = (Z_1, Z_2, ..., Z_N)$ where $z_i = (x_i, y_i)$. The idea here is to draw datasets randomly with replacement from the training data having each sample the same size as the original training set. This is done B times producing B bootstrapped datasets Hastie, Tibshirani, et al..

Bootstrap aggregation / Bagging 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 $\dfrac{\sigma^2}{n}$. 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:

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x) \tag{2.2.1}$$

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 bth bootstrapped training set to obtain $\hat{f}^{\star b}(x)$ and then average all the predictions to obtain:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{\star b}(x) \tag{2.2.2}$$

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).

2.2.2 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 $\dfrac{2}{3}$ of the observation and the other $\dfrac{1}{3}$ 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 ith observation can be predicted using each of the tree in which that observation was out of bag which will yield around $\dfrac{B}{3}$ predictions for the ith observation. To obtain a single prediction for the ith observation, we take a

majority vote of the predicted responses. This gives a single OOB prediction for the ith 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.

2.2.3 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 then accumulated over all the trees for each variable separately Hastie, Tibshirani, et al..

2.2.4 Definition.
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).

2.2.5 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, $\dfrac{(p - m)}{p}$ 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).

Random Forest therefore uses fully grown decision trees (low bias, high variance). It manages the error reduction task by reducing the variance. The trees are made uncorrelated so as to maximize the decrease in variance, but the algorithm cannot reduce bias (which is slightly higher than the bias of an individual tree in the forest). Hence the need for large unprunned trees so that the bias is as low as possible. High variance is due to the correlation in the predictors. Bagging therefore succeeds in smoothing out the variance resulting to a reduction in the test error leading to improved predictions.

2.3   Implementing the Random forest algorithm

2.3.1 Data splitting. This is how data is allocated to model building (train set) and performance evaluation (test set). The core interest is to predict the risk of new clients by not using the same population as the data used to build the model. In other words, we are testing how well the model extrapolates to a different population. A small test set would have limited utility as a judge of performance. If the size of the test set is small, the test set may not have sufficient precision to make reasonable judgement.

In this model, the training and testing sets are homogeneous, therefore random sampling techniques are used to create similar data sets. To split the train data, a random sample is chosen from the data. The function model selection from sklearn is used to split the data into train and test in the proportion of 60% to 40% respectively.

2.3.2 Data preprocessing. Centering and scaling: To center a predictor variable, we take the mean of that predictor value and subtract from all the values of that predictor. Due to centering, the predictor has 0 mean. This result is the divided by the standard deviation of the predictor. Scaling the data makes the predictor values have a common standard deviation of 1. These transformations improve the numerical stability of the calculations. In this data, some predictors like weight and height have been normalized. The only disadvantage of normalizing is the loss of interpretability as the values are no longer in the original units.

Missing values: In the data, some predictors have missing values. Missing values are more often than not related to predictor variables than the sample. Due to this, the amount of missing could be concentrated in a subset of predictors rather than occurring randomly across the predictors.

To deal with with missing values, we use the (Predictive) Value Imputation (PVI) where estimated values are imputed where missing values occur before the model is applied (Saar-Tsechansky and Provost, 2007). The widely used approach is to replace the missing value with the predictor's mean value or mode value if the attribute is categorical. These values

are estimated from the training set and imputed along the columns. The strategy we used in our data is to impute the missing values with the mean value of the predictor where missing values occur (where the value reads Nan). The categorical predictors have no missing 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.

2.3.3 Model performance. To prepare the data for modelling, we drop the features 'Id' and 'Response', assign the response and explanatory variables to a numpy array then we train the model using the train data and evaluate the performance of the model on the test data.

The following scikit learn parameters were tuned to achieve 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.

2.4   Overview of Xgboost Algorithm

The full name is eXtreme Gradient Boosting. It is a variant of gradient boosting, a tree 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.

2.4.1 Boosting. Combines the outputs of many weak classifiers to produce powerful classifiers in an iterative fashion. A weak classifier is a classifier whose error rate is better than random guessing. Boosting therefore applies weak classification algorithm sequentially to repeatedly modified versions of the data, producing a sequence of weak classifiers. Classifiers are then trained on weighted versions of the dataset and combined to produce a final prediction. These predictions are then combined to produce a final prediction Hastie, Tibshirani, et al.

2.4.2 Gradient boosting. Uses the direction of quickest improvement (pseudo gradient). A decision tree is picked as close as possible to the pseudo gradient and added to the model. Here, we are picking the decision tree to be an improvement in the direction we can consider the best way to improve the overall loss function. Gradient boosting has shrinkage and regularization built in Li (2016).

2.4.3 Regularization. In addition to size of constituent trees, the number of boosting iterations reduces the training risk such that for the number of iterations large enough, this risk can be made small. In the context of boosting, shrinkage is scaling the contribution of each tree by a factor $0 < v < 1$ when it is added to the current approximation. Smaller values of v implies more shrinkage and more training risk for the same number of iterations. Therefore, v, which controls the learning rate, and the number of iterations control the prediction risk on the training data. These parameters are dependent on each other in that smaller values of v for the same training risk yield larger values of the number of iterations. It was found that smaller values of v yield better test error results and large values for the number of iterations are required. The strategy is thus to set v to be small ($v < 0.1$) and choose the number of iterations by early stopping Hastie, Tibshirani, et al.

2.4.4 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.

530   Performance: Has better performance on different datasets.

531   **2.4.5 XG Boost parameters.** The parameters can be grouped into (He (2015)):

532   General parameters: Under general parameters we have number of threads. nthread is
533   defined as the number of parallel threads.

534   Booster parameters: Under booster parameters we have the Stepsize and Regularization
535   parameters.

536   1. gblinear: linear function.

537   2. gbtree: tree based model.

538   Task parameters: Here we have the objective and the evaluation metric.

539   **2.4.6 Model Specification.**

540

541   Training objective.

542   This model is a collection of decision trees. Supposing we have K trees, the model is given
543   by He (2015):

$$\sum_{k=1}^{K} f_k. \tag{2.4.1}$$

544   With all the prediction trees, we predict by:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{2.4.2}$$

545   x_i: Feature vector of the $i^t h$ data point.

546   The prediction at the $t^t h$ step can be defined as:

$$\hat{y_i}t = \sum_{k=1}^{t} f_k(x_i) \tag{2.4.3}$$

547   To train the model, we need to optimize a loss function He (2015). We use;

548   1 Rooted mean squared error for regression.

$$-L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{2.4.4}$$

549   2 Logloss for binary classification.

$$- L = -\frac{1}{N} \sum_{i=1}^{N} (y_i log()p_i) + (1 - y_i)log(1 - p_i). \tag{2.4.5}$$

550   3 mlogloss for multi-classification.

$$- L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} log(p_{i,j}). \tag{2.4.6}$$

551   Regularization is an important part of the model. The regularization term controls the
552   complexity of the model thereby preventing over fitting.

$$\Omega = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} wj^2. \tag{2.4.7}$$

553   λ: Regularization parameter.

554   γ: Minimum loss reduction required to make a further partition on a leaf node of a tree.

555   T: The number of leaves.

556   wj^2: The score of the jth leaf.

557   Combing the loss function and the regularization function, we get the objective function of
558   the model.

$$obj = L + \Omega. \tag{2.4.8}$$

559   where the loss function controls the predictive power and regularization controls the sim-
560   plicity He (2015).

561   **2.4.7 xgb.Dmatrix.** : This is the data structure used by the XGBoost algorithm. XGBoost
562   prepossess the input data and labels it into an xgb.Dmatrix object before implementing it
563   to the training algorithm.

564   An xgb.DMatrix contains:

565   1. Prepossessed training data.

566   2. Missing values.

567   3. Data weight.

568   If the training process is to be repeated on the same data set, the xgb.DMatrix works well
569   as it saves on the prepossessing time He (2015).

2.5   Implementing the XGBoost algorithm

2.5.1 Data preprocessing. Factorize string variables: Just like the Random Forest, we transform Product_info_2, a string variable into numeric variables.

Adding an Interaction term: 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.

Using a combination of predictors can be more effective than using the individual values. More often than not, data engineering is informed by the modeller's understanding of the data and problem and not derived from any mathematical technique Kuhn and Johnson (2013).

2.5.2 Applying offsets. An offset is a model variable with a pre-specified coefficient. Modelling is affected by the real-world fact that insurance data is often incomplete, generally dirty and inconsistently coded. Due to this, modellers are forced to exclude variables that are important in predicting. This will bias the estimate of the model if this omitted variables correlate with the target variable and one or more of the other modelling variables. This is known as omitted variable bias Yan, Guszcza, Flynn, and Wu (2009). To curb this phenomenon, we accommodate the effect of omitted variables in the model design. A traditional approach to this problem is to adjust the model's target variable. It is therefore important to consider an offset when performing multivariate analysis so as to avoid the omitted variable bias. The offsets are generated after optimization by the fmin_Powell function which determines the minimum numeric value between arguments.

2.5.3 Model performance. To prepare the data for modelling, we drop the features 'Id' and 'Response'. We train the model using the train data and evaluate the performance of the model on the test data.

The following scikit learn parameters were tuned to achieve the best score He (2015):

1. "reg:linear": default option, Linear regression.

2. "binary: Logistic": Outputs probability. It performs logistic regression for binary classification.

3. "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 $[0,\infty]$ 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 $[1,\infty]$, the default is 6.

We train the model by specifying the number of times to train the model. We then predict the target variable on the test data provided, submit the result to the kaggle website and the accuracy score is calculated.

₆₂₈ 3. Results and discussions

₆₂₉ In this chapter, we give a brief description of the data and visualize the different features of the
₆₃₀ data with plots. We present the results and outline some key points on the XGBoost algorithm.

₆₃₁ 3.1 Data description

₆₃₂ The data provided is divided into train and test data set (Web, Accessed March 2017a). There
₆₃₃ is a sample submission file that gives instructions on how predictions are submitted on the kaggle
₆₃₄ website. The train data is composed of 53,381 entries(clients) and a response variable. The total
₆₃₅ number of features is 128. The test data is composed of 19,765 entries(clients) and a total of
₆₃₆ 127 features. The predictions range from 1 to 8 where 1 represents the lowest risk level and 8
₆₃₇ represents the highest risk level.

₆₃₈ **3.1.1 Data visualization.**
₆₃₉ The scatter plot below shows the relationship between the various responses and the normalized
₆₄₀ Ins Age.

Figure 3.1: Scatter plot of Age vs Response
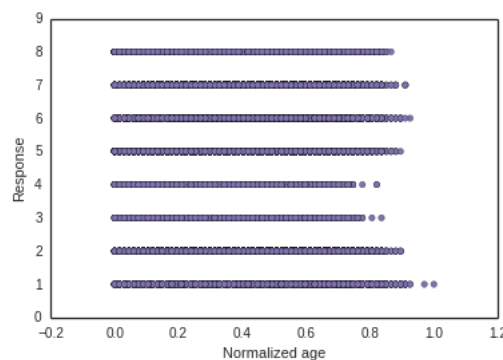
₆₄₁ The risk prediction classes are evenly distributed across the ages of the clients with a few outliers.

₆₄₂ The plot of the feature importances lists the most important features in descending order.

₆₄₃ The figure below is a variable importance plot for the insurance data. Variable importance is
₆₄₄ computed using the mean decrease in the gini index and expressed relative to the maximum.
₆₄₅ Variable importance is the overall quantification of the relationship between the predictor and
₆₄₆ the outcome. With a large number of predictors, the exploratory analysis of the predictors may
₆₄₇ not be feasible and may concentrate on those with strong relationships with the outcome. We
₆₄₈ therefore rank predictors so as to sift through large amounts of data. A substantial drop in the
₆₄₉ performance of the model indicates that an important feature has been dropped.

₆₅₀ The subsequent table lists the five most important features and the five least important features.

Figure 3.2: Feature importance plot
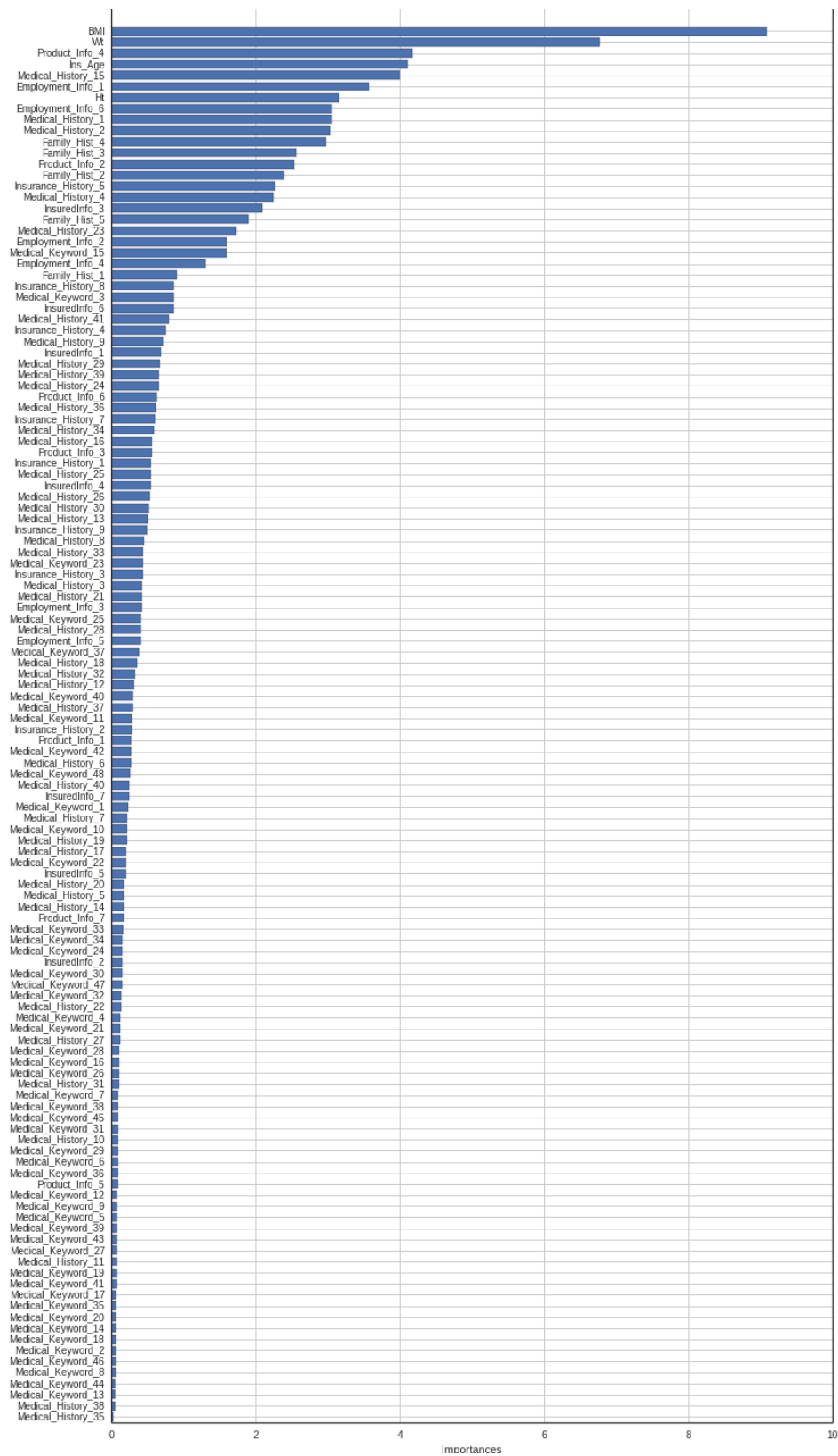
The first 5 important predictors	
BMI	0.089700
Wt	0.068185
Product_info_4	0.041591
Ins_Age	0.040372
Medical_History_15	0.039763
The last 5 less important predictors	
Medical_History_35	0.000219
Medical_History 38	0.000512
Medical_History_13	0.000551
Medical_History_44	0.000612
Medical History 8	0.000622

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

Figure 3.3: Class Imbalance of the response variable
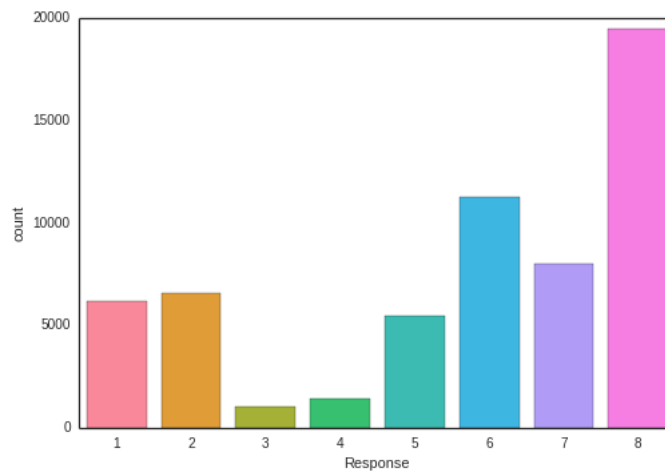
Most clients fall under the risk class 8 while the least number of clients fall under the risk class 3.

**Class imbalance**: Data classification is difficult because of the unbounded size and the imbalanced nature of the data. Class imbalance occurs where one of the classes has more samples than other classes. This leads to most algorithms ignoring or misclassifying the minority sample(those that rarely occur but are very important) and focussing on the classification of major samples. A dataset is said to be highly skewed if samples from one class is higher than the other Longadge and Dongre (2013).

Threshold metrics help to curb the class imbalance problem. These metrics have a threshold level such that examples above the threshold are predicted as positive and the rest are negative. We used the quadratic weighted kappa metric which works well with unbalanced datasets by measuring the among observers Jeni, Cohn, and De La Torre (2013).
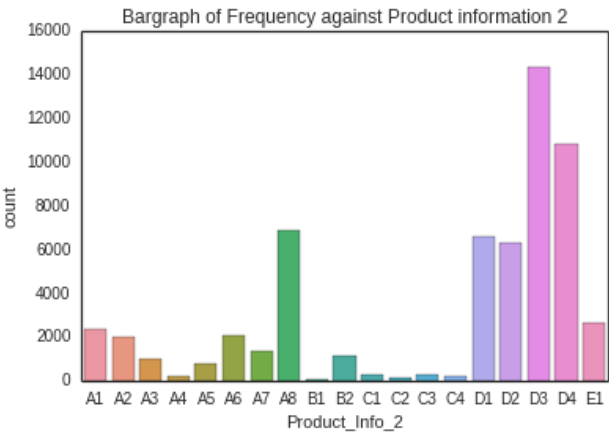
665 Descriptive analysis of the Response feature.

666

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

667 The mean risk prediction is a roughly 6.
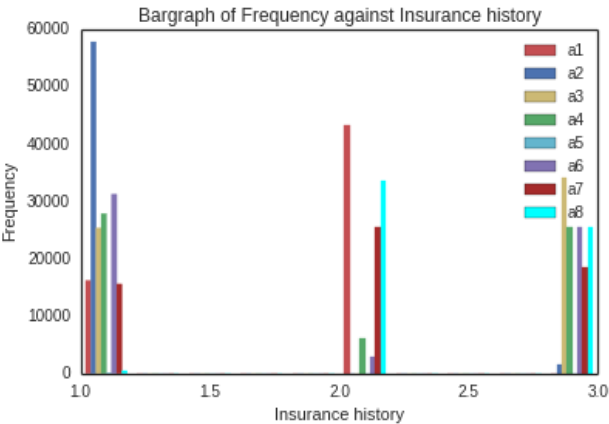
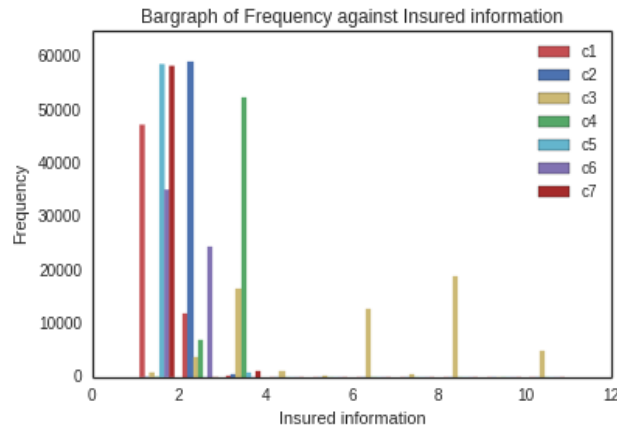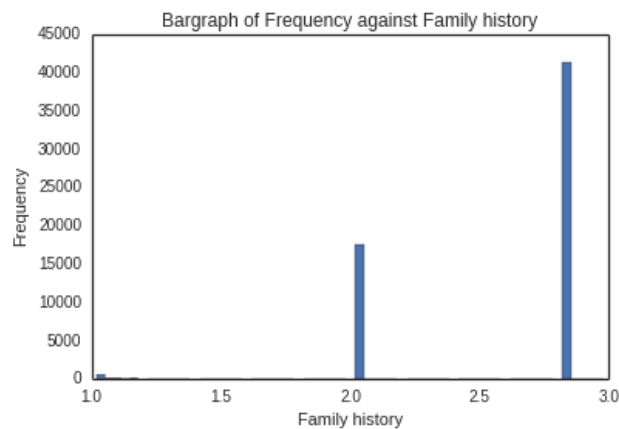
(a) Insurance history plot

(b) Product Information 2

Figure 3.4: Plots of count against insurance history and product information 2 showing the distribution of the two features.

668 Plot(a): Insurance_History_2, 3, 4, 7, 8 and 9 take on the categorical values 1, 3, 2. Insur-
669 ance_History_1 takes on the categorical values (1, 2) while Insurance History_5 takes on a range

670   of values almost close to 0. Insurance_History_2, 1 and 3 have the highest distribution of the
671   categorical variable 1, 2 and 3 respectively while Insurance_History_8, 7 and 2 have the lowest
672   distribution of the categorical variable 1, 2 and 3 respectively. Plot(b) shows the distribution of
673   the categorical variable Product_Info_2 where the product D3 was highly preferred.


(a) Insurance information plot

(b) Family history plot

Figure 3.5: Plots of count against insurance information and family history showing the distribution of the two explanatory variables.

674   Plot(a) InsuredInfo_1 takes on the categorical values (1, 2, 3), InsuredInfo_2 and 4 takes the
675   values (2, 3). InsuredInfo_5 and 6 takes the values (1, 3). InsuredInfo_6 takes the values (2,
676   1). InsuredInfo_3 takes on a range of values. InsuredInfo_1, 5 and 6 take on the highest values
677   of the categorical variable 1 while InsuredInfo_3 takes the highest value of the variable 8 and
678   InsuredInfo_4 takes on the highest value of categorical variable 3. Plot(b) Family_Hist_1 takes on
679   the values (2, 3, 1) while the rest of the features take on values close to 0. The highest value
680   for Family_Hist_1 was recorded for the categorical variable 3.
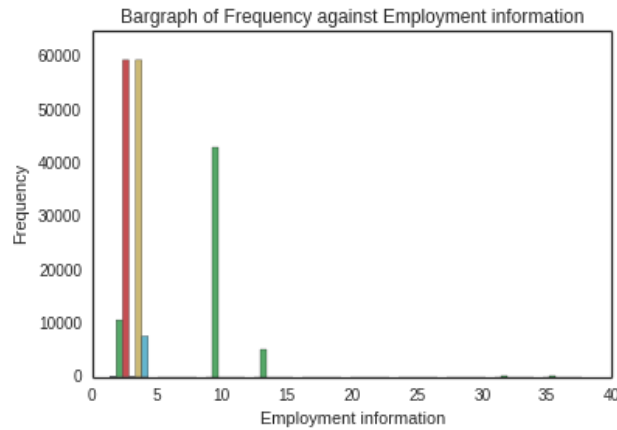
(a) Employment information plot
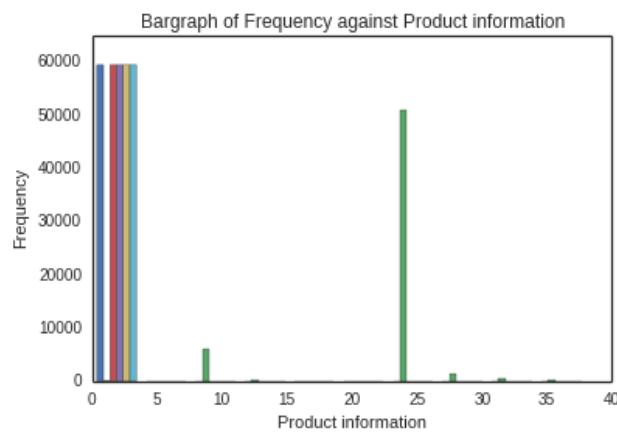
(b) Product information plot

Figure 3.6: Plots of count against employment information and product information showing the distribution of the two features.

Plot(a): 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. Employment_Info_3 and 5 have the highest values as the categorical variables 1 and 2. Employment_Info_2 takes on the highest value as 9. Plot(b): Product_Info_1, 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. Product_Info_1, 5, 6 and 7 take on the highest values of the categorical variables 1, 2, 3 and 1 respectively. Product_Info_3 takes on the highest value as 26.

3.2   Model evaluation

Using the random forest algorithm to predict the risk levels of the test data, the model scored 0.53074 as evaluated by the quadratic weighted kappa metric.

Weakness of Random forest

1. 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..

2. 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 Boooosting. 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.

3.3   Key points on XGboost

3.3.1 Guide on parameter tuning.
Tuning 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. Therefore, parameters have to be tuned to achieve good results.

3.3.2 Key points on parameter tuning.

1. To control over fitting where over fitting is a scenario where an algorithm fits well on the training data but is less accurate on unseen or the test data.

2. To deal with imbalanced data where one class has more samples than the other.

3.3.3 Definition. Bias-variance trade off 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).

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).

3.3.4 How to counter the bias-variance trade off in XGBoost.

We can group the booster specific parameters as below and tune the given parameters accordingly (He, 2015).

1. To control the model complexity: max_depth, min_child_weight and gamma.

2. Robust to noise: subsample, colsample_by tree.

3.3.5 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, 2015).

3.3.6 How to build a winning algorithm in a kaggle competition.

To score highly on kaggle, one needs to consistently focus on He (2015):

1. Parameter tuning.

2. Model ensembling which is combining two or more models so as to achieve a more accurate prediction.

3. Feature engineering which is defined as generating addition relevant features from the raw data so as to improve the predictive power of an algorithm.

3.3.7 Why XGBoost is the winning algorithm for kaggle competitions..

The following are the reasons why XGBoost is a winning algorithm (He, 2015).

1. It is efficient as it allows for parallel computing and can be run on a cluster.

2. It is accurate: It outputs good results for most datasets.

3. Feasibility: It provides a platform for tunable parameters.

4. XGBoost is easy to use and install with a highly developed R and python interface.

4. Conclusion

We first showed the procedure used in building a predictive model which include: pre-processing the predictor data, estimating the model parameters, selecting predictors for the model, evaluating the model performance and fine tuning the class prediction rules.

We then make a comparison between the Random Forest and the XGBoost algorithms. Despite Random Forest being an efficient algorithm on large datasets, handling thousands of variables without variable deletion, it has been observed to over fit datasets with noisy classification. On the other hand, XGBoost has proved to build higher accuracy models due to its regularization feature which reduces over fitting. Additionally, XGBoost offers a randomization parameter used to decorrelate the individual trees and in turn reduce the overall variance of the additive tree model. XGBoost also has an inbuilt method of handling missing values. The modeller is required to initialize a different value different from other observations and pass it as a parameter. XGBoost then learns which path to take for missing values in future data. The computing time in Random Forest algorithm is longer as compared to XGBoost. In conclusion, XGBoost is an algorithm that carefully takes into account the bias-variance tradeoff in every aspect of the learning process, making XGBoost a superior algorithm to Random Forest.

4.1 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:

1. Submission files.

2. 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.

28

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

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

5. Appendix

5.1 Random forest algorithm.

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


test2.shape
(19765, 127)


raw_data1.columns.values


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


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


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


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)


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_Y


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)
```

5.2 XGBoost algorithm

```
817  def eval_wrapper(yhat, y):
818      y = np.array(y)
819      y = y.astype(int)
820      yhat = np.array(yhat)
821      yhat = np.clip(np.round(yhat), np.min(y), np.max(y)).astype(int)
822      return quadratic_weighted_kappa(yhat, y)


823  def get_params():
824
825      params = {}
826      params["objective"] = "reg:linear"
827      params["eta"] = 0.05
828      params["min_child_weight"] = 360
829      params["subsample"] = 0.85
830      params["colsample_bytree"] = 0.3
831      params["silent"] = 1
832      params["max_depth"] = 7
833      plst = list(params.items())
834
835      return plst


836  def apply_offsets(data, offsets):
837      for j in range(num_classes):
838          data[1, data[0].astype(int)==j] = data[0, data[0].astype(int)==j] +
839          offsets[j]
840      return data


841  # global variables
842  columns_to_drop = ['Id', 'Response']
843  xgb_num_rounds = 720
844  num_classes = 8
845  missing_indicator = -1000


846  train = pd.read_csv("train.csv")
847  test = pd.read_csv("test.csv")


848  all_data = train.append(test)
849  all_data.shape


850  all_data['Product_Info_2_char'] = all_data.Product_Info_2.str[0]
851  all_data['Product_Info_2_num'] = all_data.Product_Info_2.str[1]
```

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

853

854  all_data['Product_Info_2_char'] =pd.factorize(all_data['Product_Info_2_char'])[0]

855

856  all_data['Product_Info_2_num'] = pd.factorize(all_data['Product_Info_2_num'])[0]

857

858  all_data['BMI_Age'] = all_data['BMI'] * all_data['Ins_Age']

859

860  med_keyword_columns=all_data.columns[all_data.columns.str.startswith('Medical_Keyword_'

861

862  all_data['Med_Keywords_Count'] = all_data[med_keyword_columns].sum(axis=1)


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


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


867   xgtrain = xgb.DMatrix(train.drop(columns_to_drop, axis=1),
868            train['Response'].values, missing=missing_indicator)
869  xgtest = xgb.DMatrix(test.drop(columns_to_drop, axis=1),
870            label=test['Response'].values,missing=missing_indicator)

871



872   plst = get_params()
873  model = xgb.train(plst, xgtrain, xgb_num_rounds)

874



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


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


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

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


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


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

5.3   Data description.

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,

909 Medical_History_28, Medical_History_29, Medical_History_30, Medical_History_31, Medical_History_33,
910 Medical_History_34, Medical_History_35, Medical_History_36, Medical_History_37, Medical_History_38,
911 Medical_History_39, Medical_History_40, Medical_History_41

912 **The following variables are discrete**:

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

915 **The following variables are continuous**:

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

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.

Tong He. Xgboost extreme gradient boosting. Technical report, 2015.

László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251. IEEE, 2013.

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

Cheng Li. A gentle introduction to gradient boosting. *URL: http://www. ccs. neu. edu/home-/vip/teach/MLcourse/4_ boosting/slides/gradient_boosting. pdf*, 2016.

Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*, 2013.

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

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

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

956   Maytal Saar-Tsechansky and Foster Provost. Handling missing values when applying classification
957   models. *Journal of machine learning research*, 8(Jul):1623–1657, 2007.

958   Vijayakumar Sethu. Computational learning theory., 2007.

959   Carolin Strobl, Anne-Laure Boulesteix, and Thomas Augustin. Unbiased split selection for clas-
960   sification trees based on the gini index. *Computational Statistics & Data Analysis*, 52(1):
961   483–501, 2007.

962   Jun Yan, James Guszcza, Matthew Flynn, and Cheng-Sheng Peter Wu. Applications of the offset
963   in property-casualty predictive modeling. In *Casualty Actuarial Society E-Forum, Winter 2009*,
964   page 366, 2009.

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