Predicting Kaggle Restaurant Annual Revenue with Support Vector Machine and Random Forest

Kevin Pei, Sprott School of Business, 100887176
Rene Bidart, Faculty of Mathematics and Statistics, 100F49907

Prepared for: Dr. Shirley Mills
Faculty of Mathematics and Statistics
April 14, 2015

Abstract

Kaggle is an open-competition, open-community website for data scientists to congregate and compete in solving big data challenges. In this report, we explore one Kaggle competition hosted by Tab Food Investments (TFI) where participants are asked to predict annual revenue of Turkish restaurants. The data set itself contains many inherent problems which we present insight and offer potential solutions to. Using a simple ensemble method between Random Forest and Support Vector Machine, we were placed 60th out of 1746 participants (at this time of writing) with a root mean squared error of 1,658,797.

1 Introduction

KAGGLE is an online website designed to provide data scientists an open platform to learn, collaborate and compete on big data challlenges often hosted by corporations with large data sets. The primary focus of Kaggle is on predictive analytics where contestants are often given a training data set to fit their model and a test data set to submit their predictions to be evaluated. The evaluation process consists of collecting a subset of the test set prediction and calculating a score, usually on the errors of the predictions. Multiple submissions from each user are allowed and the best score from all submissions are displayed on the public leaderboards. There are often prizes linked to top placements on the public leaderboards, giving participants a large incentive to compete and improve.

In this report, we ourselves become competitors and focus on the TFI Kaggle competition. The report will document our analysis and discovery of the training/test data sets given to us by TFI as well as the quantitative models we employ to predict the test set. Supporting R code will be provided to show detailed procedures and the report is presented in the same chronological order as we have approached the problem. The structure of this paper is as follows: we introduce and describe the TFI competition/data set in section 2; problems with data set variables and solutions

are presented in section 3; section 4 details the primary models within the study; ensemble methods are discussed within section 5; section 6 presents our results of our actual test set performance along with the models and changes employed; section 7 concludes along with a brief discussion on limitations and future considerations.

2 TFI Competition

The TFI Kaggle competition can be framed as a supervised learning problem where the objective is to develop a model and a set of preprocess procedures to accurately predict a cross-sectional sample of Turkish restaurant revenues collected in a given year. These restaurants are diverse in that each has a different style of service e.g Burger king, Sbarro, Popeyes, etc.

2.1 Evaluation

The score measure presented in the competition is the root mean squared error of the test set revenue.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Where $\hat{y}_i$ is the predicted revenue of the ith restaurant and y is the actual revenue of the ith restauarant.

2.2 Data Set

The data set TFI provides is peculiar in many aspects and we present more specific problems in later sections. The TFI data set consists of a training and test set with 137 and 100,000 samples respectively. This is interesting in itself since such a small training set is presented relative to the final test set. The 137 training samples provide actual revenue while the test set does not and expects the user to submit their predictions on the 100,000 firms. The 43 data fields provided are listed below:

- **ID**: Restaurant ID
- **Open Date**: Date that the restaurant opened in the format M/D/Y
- **City**: The city name that the restaurant resides in
- **City Group**: The type of city can be either *big cities* or *other*
- **Type**: The type of the restaurant where *FC* - Food Court, *IL* - Inline, *DT* - Drive through and *MB* - Mobile
- **P-Variables (P1, P2, ... ,P37)**: Obfuscated variables within three categories: demographic data, e.g population, age, gender; real estate data e.g car park availability and front facade; commercial data e.g points of interest, other vendors, etc. It is unknown if each variable contains a combination of the three categories or are mutually exclusive.
- **Revenue**: Annual (transformed) revenue of a restaurant in a given year and is the target to be predicted

As shown, the majority of the data fields are obfuscated variables without giving the statistican any prior knowledge of each one. One important rule overlooked in this competition is that no additional third-party data may be added to the prediction. We believe this is done to ensure fairness in access to information as well as to reward technical knowledge rather than business acumen.

3 Data Engineering

In this section, we provide eight interesting observations and problems with the data set and potential solutions to them. While these solutions are valid in a theoretical sense, there is no way for us to show certain treatments that they actually improve performance on the test set due to a constraint Kaggle imposed such that only three submissions can be made every 24 hours. Furthermore, the problem of submitting two sets of predictions to contrast effectiveness of a change is that the errors from model training vary as random weights are initalized for each iteration. Therefore, it is impossible to distinguish if improvements are a result of pure luck or actual improvements in the process.

3.1 Revenue Distribution

Plotting a simple histogram of the revenue field show that it is not normally distributed but rather has a long tail toward the right. One simple solution to this problem is to take the ln of the revenue field and obtain something that is more similar to a normal distribution. See figure 1 for a histogram comparison between the original revenue and logged revenue.

```
revenue <- train[,43]/1000000
par(mfrow=c(1,2))
pnorm.rev <- format(shapiro.test(revenue)$p.value,digits=2) #Normality
   Test using Shapiro-Wilks Test
pnorm.logrev <- format(shapiro.test(log(revenue))$p.value,digits=2)
hist(revenue, breaks=40, main=paste("Histogram of Revenue \n P-Value:
   ",pnorm.rev), xlab="Revenue (Millions)", ylab="Frequency")
hist(log(revenue), breaks=40, main=paste("Histogram of ln(Revenue) \n
   P-Value: ",pnorm.logrev), xlab="Revenue (Millions)",
   ylab="Frequency")
```

If this distribution for revenue holds in the test set, transforming the variable before training models will improve performance vastly, which holds in the training set. However, we cannot be completely certain that this distribution will hold in the test set.

3.2 Parsing *Open Date*

Since the opening date cannot simply be assumed to be factors, we propose two treatments in place of *Open.Date* for more meaningful interpretations. The first treatment is to calculate the number of days open by taking the difference, in days, between the opening date and an arbitrary constant such that it is later than the latest opening date of all samples. The date we chosen is January 1, 2015 and the difference in days can be treated as a continuous feature.

```
train$Open.Date <- as.numeric(as.POSIXlt("01/01/2015",
   format="%m/%d/%Y") - as.POSIXlt(train$Open.Date, format="%m/%d/%Y"))
test$Open.Date <- as.numeric(as.POSIXlt("01/01/2015",
   format="%m/%d/%Y") - as.POSIXlt(test$Open.Date, format="%m/%d/%Y"))
```

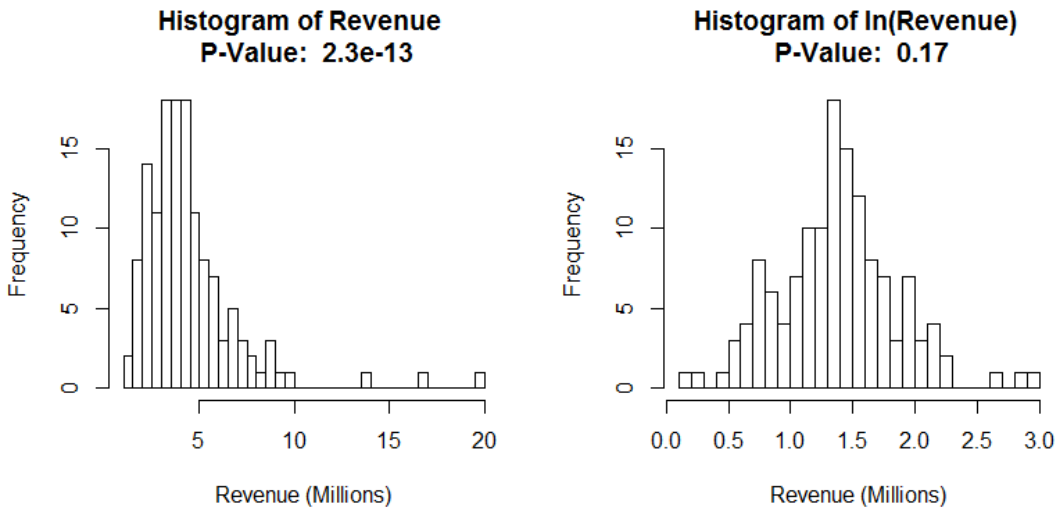

Figure 1: Histogram of both revenue and natural log of revenue. P-Value reported is the result of Shapiro-Wilks Test for normality. Using a 5% significance cut-off, we can reject the null of normality for revenue and fail to reject null of normality for ln(revenue)

The second treatment we propose is to create two additional features where one is the month that they opened and the other is the year that they opened. These two features can potentially help proxy seasonality differences since restaurant revenues are highly cylical.

```
train$Open.Month <- as.factor(format(as.POSIXlt(train$Open.Date,
    format="%m/%d/%Y"), "%m"))
train$Open.Year <- as.numeric(format(as.POSIXlt(train$Open.Date,
    format="%m/%d/%Y"), "%Y"))
test$Open.Month <- as.factor(format(as.POSIXlt(test$Open.Date,
    format="%m/%d/%Y"), "%m"))
test$Open.Year <- as.numeric(format(as.POSIXlt(test$Open.Date,
    format="%m/%d/%Y"), "%Y"))
```

3.3 The Unaccounted Problem

The unaccounted problem is simply the disparity between the features for the training set and test set where the test set actually contains more information than the training set. This problem directly affects the categorical versus continuous problem which is discussed in the next section. Within the training set, features that are treated as categorical can pose potential problems when being applied to the test set. These features are *Type*, *City* and most of the P-Variables. Type spans FC, IL and DT within the training set but is missing MB which is present within the test set. Likewise, there are 34 cities in the training set but 57 in the test set. Most supervised learning models fitted through the training set will encounter errors due to missing coefficients for the additional classes. We propose two treatments for the *Type* and *City* issue below. The p-variable problem is discussed in the next section.

3.3.1 Type

The unaccounted problem persists in *Type* where mobile type restaurants are in the test set but not in the training set, making it difficult to predict when no coefficients or weights are available. In the

test set of 100,000 samples, 290 are Mobile which accounts for a small proportion and we would ideally expect for it to yield a small percentage change in error. We propose a k nearest neighbour treatment for the restaurant type. We first construct a query matrix within the test set where each row is a *mobile* type restaurant. The search matrix consists of a test set where each row is not a *mobile* type restaurant. Any factor variables are removed from the query since the KNN algorithm within our implementation can only deal with continuous variables. The objective is to match each *mobile* type restaurant with a non-mobile type restaurant through finding the most similar features, as measured by euclidean distance.

More formally, let X be the test set matrix with only continuous features, the query matrix is $Q \subset \{X, X_{type} = MB\}$ and search matrix is $S \subset \{X, X_{type} \neq MB\}$. The KNN index $\Re$, a $N_Q \times K$ matrix, is computed as such

$$\min_{I_S, I_S \notin \Re} D = \sqrt{\sum_{i=1}^{M} (Q_i - S_i)^2}$$

where K is the number of nearest neighbour desired, N_Q is equal to the number of rows within Q, I_S is equal to the row index of S, M is the total number of continuous feature columns. The algorithm is repeated over each possible pair of Q and S. After obtaining K nearest neighbours for each row of Q, the mode is taken for the restaurant type and the *mobile* type is replaced with the mode. It is clear that the implementation of kNN requires computational power[1] and for maximum efficiency we use the `FNN` package in R. Code for the test set is below.

```
# Assuming all categorical factors have been removed.
Q <- test[test$Type == "MB",]
S <- test[test$Type != "MB",]
re <- knnx.index(S, Q)
# Convert all indices to Actual Type for that row index
re.type <- t(apply(re, 1, function(x){S$Type[x]}))
# Get the most popular
MB.transform <- as.vector(apply(re.type, 1,
   function(x){names(sort(table(x),decreasing=TRUE)[1])}))

test$Type[test$Type == "MB"] <- MB.transform
test$Type <- factor(test$Type) #Refactor
```

3.3.2 City

This problem, much like *type*, can also be treated with the kNN approach. We instead introduce a K-Means clustering methodology in imputing City values which is similar in nature to the kNN treatment. The unsupervised algorithm looks for tight clusters of data in features to and assigns them to an arbitrary cluster. Since we know that the P-Variables are suppose to represent three categories including geographical data, we assume each P-Variable to be mutually exclusive in category and attempt to identify variables that best represent *City* which should contain many geographical attributes. See figure 2 below for a box plot of the mean of the 37 p-variables foreach city. More specifically, we collect the average P-Variable for each subset of cities and plot the deviation over all cities. The intuition behind this idea is simple. Since we do not know exactly what each p-variable represents, under the assumption of mutually exclusive categories, a change

[1]This issue is evident in the Zero Problem which we briefly show later

in city should illict a change in certain p-variables. Plotting the mean over each city should help us identify that through examining the range of deviation a variable can have. One caveat to this method is related to the continuous vs. categorical problem where p-variables are discrete in its values and large deviations are simply a result of its categorical values.

```
# Assuming the training set only contains P-Variables
cities <- unique(train$City)
# Find the mean of all subset of cities
p.bar <- lapply(cities, function(c){ apply(train[which(train$City ==
    c),], 2, mean)})
p.bar <- matrix(unlist(p.bar), byrow=TRUE, ncol=37)
boxplot(p.bar)
```

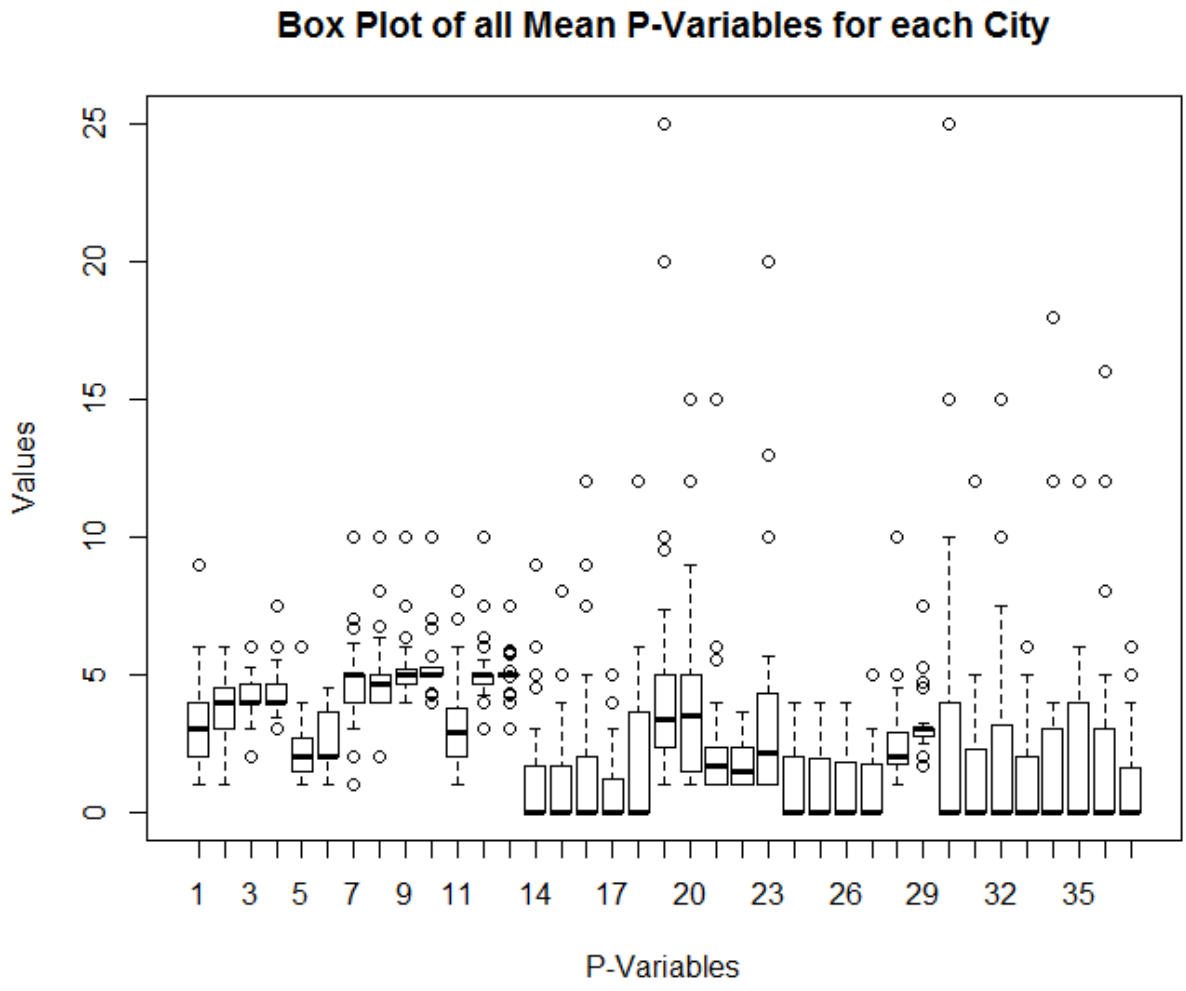

Figure 2: Box plot of the mean p-variables over each city. Each point in the sample is equal to the mean p-variable for that city

From figure 2, we identified P1, P2, P11, P19, P20, P23, and P30 to be approximately a good proxy for geographical location. Using k-means methodology (MacQueen, 1967), we can reclassify each 137 restaurant cities into arbitrary clusters based on these variables. If we let K be the number of clusters (centers), μ_k is a $M \times 1$ vector of randomly initialized centroid, often randomly assigned to one data point. Each centroid is then fitted through an iterative process to minimize the cost

function J which is simply the mean within-cluster distance

$$J = \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1}^{C}||X_j^{(i)} - \mu_i||$$

where X is the training set matrix with above P-Variables and $X^{(i)}$ is a set of firms within the ith cluster. C is equal to the number of row entries in $X^{(i)}$ and $||\quad||$ denotes euclidean distance. To select an optimal K, we use the Davies and Bouldin (1979) index which is a measure of the ratio of within-cluster distance $S_i = \frac{1}{C}\sum_{i=1}^{C}||X_i - \mu_i||$ and between-cluster distance $M_{i,j} = ||\mu_i - \mu_j||$. The DB measure can be defined as

$$R = \max(\frac{S_i + S_j}{M_{i,j}}) \qquad \forall i, j, i \neq j$$

Under optimality, the DB measure should be extremely low as the spread within-cluster is small and the distance between-clusters is large. Implementation for K-means clustering is performed using the `cclust` package in R. Code for the Davies-Bouldin Index is retrieved from Mills (2015). See figure 3 for a plot of the Davies-Bouldin Index as K increases. It is clear that the DB Index converges and selecting K between the range of 20 to 25 would be optimal. In our use case, we set $K = 20$ and show in later sections that clustering leads to significantly improved performance.

```
# Subset only the relevant P-Variables
cityMatrix <- train[,c(2,5:6,15, 23:24,27,34)]
dbind <- c()
for(i in 2:30){
  km <- cclust(as.matrix(cityMatrix), i)
  dbind <- c(dbind, DB(km$centers, km$withinss, i))
}

plot(dbind, main="DB Index of P-Variables Clustering", xlab="K",
   ylab="DB Measure")
text(dbind, labels=c(2:30),cex=0.7, pos=3)
```

We then apply the clustering centroids to both the test set and training set.

```
k <- 20
km <- cclust(cityMatrix, k)
train$City <- km$cluster[1:nrow(train)]
test.clusters <- predict(km, as.matrix(test))$cluster
test$City <- test.clusters
```

It is worth noting that there are differences for the P-Variables between the test set and training sets. It is necessary to apply the training-fitted centroids to the test set so that results are consistent when training our random forest and SVM models and likewise in predicting the test set.

3.4 The Categorical vs. Continuous Problem

While it is clear that certain variables such as *City* and *Type* can be treated as categorical, it is unclear whether the obsfuscated P-Variables should be treated categorical as they are discrete in nature. The unaccounted issue persists if they are treated as categorical. For example, if we plot the difference between the number of unique values in the test set and the training set for each P-Variables, there are almost always more values in the test set than the training set. See figure 4. Re-factoring features as categorical will then fail in test set predictions. We believe that the

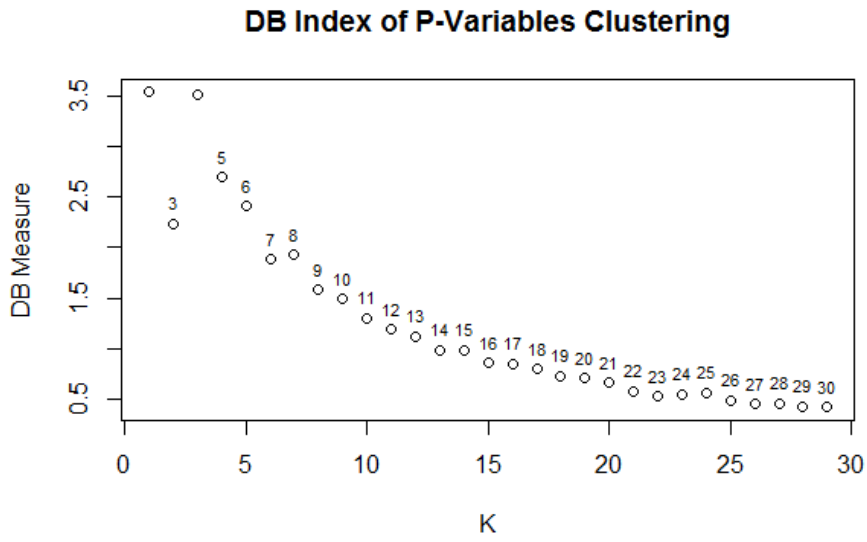

Figure 3: Davies-Bouldin index for K-Means clustering on variables: P1,P2,P11,P19,P20,P23,P30. The x-axis is the number of clusters used and y-axis is the DB Index measure.

benefits of assuming continuous values far exceed the cost. Using previous proposed treatments, like kNN or K-Means, on unaccounted for variables may actually increase errors due to model misspecification when applying on a large number of data points.

```
buf <- c()
for (i in 6:42){
  buf <- c(buf, length(unique(test[,i]))-length(unique(train[,i])))
}

barplot(buf,names.arg=colnames(train)[6:42], main="Additional Discrete
   Data point", xlab="Variables", ylab="Test less Train")
```

3.5 The Zero Problem

For certain P-Variables, a large number of samples contain zero values and are dependent among each other such that if one p-variable has zero on a certain row, the probability that other p-variables take on a zero value is high. This can be shown by getting a count of zeros for the training data set for all P-variables. See below. Although not shown, the test set exhibits similar characteristics as the training set p-variables.

```
a <- apply(train[,-c(1:5,43)],2,function(x){sum(x == 0)})
a
```

The number of zeros for these p-variables are quite frequent and consistent in which it is safe to assume that these zeros occur across the row rather than appearing randomly. We investigated this issue further by examining a parallel coordinates plot for a subset of zero-containing p-variable (P14-P18) and revenue. See figure . It is clear from the pink cluster (the zeros) that there's a subsequent large variation in revenue while the orange cluster shows a much more consistent grouping of low revenue restaurants. These two observations has led us to create the assumption that the zeros are placeholders for missing value. We used the same treatment of kNN proposed before

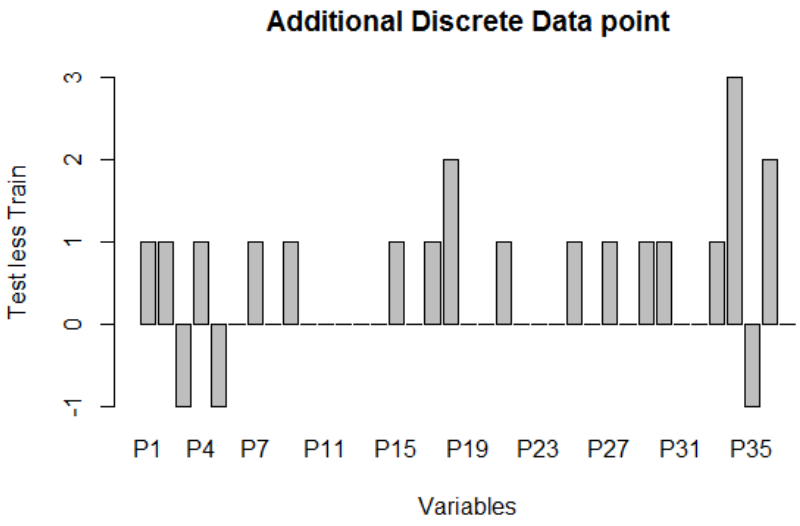

Figure 4: Plotted number of unique values in the test set less number of unique values in the training set for variables: P1, P2, ..., P37

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
0	0	1	0	0	0	0	0	0	0	0	0	0	88	88
P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30
88	88	88	0	0	0	0	0	88	88	88	88	0	2	88
P31	P32	P33	P34	P35	P36	P37								
88	88	88	88	88	88	88								

where $K = 10$ and we take the mean of the nearest neighbours. The computational time took approximately 10 hours for the test set and we feel that this type of treatment with kNN is a delicate process that needs to be carefully considered before continuing. We also show that results do not actually improve after applying the kNN imputations.

```
# For each column of P-variables with zeros
q.col <- c(18:22,28:31,34:41)
for (i in 1:length(q.col)) {
   x <- test[,q.col[i]] # Get the ith column

   ind <- which(x==0) # Find zeros
   #1:4 represents removing non-numeric columns
   parsed <- test[-ind, -c(1:4, q.col)]
   query <- test[ind,-c(1:4, q.col)]

   knn <- knnx.index(parsed, query, k=5)
   #Get the mean from the parsed KNN indices
   knn <- apply(knn, 1, function(x){mean(test[-ind,q.col][x,i])})

   test[ind,q.col[i]] <- knn
   print(i)
 }
```

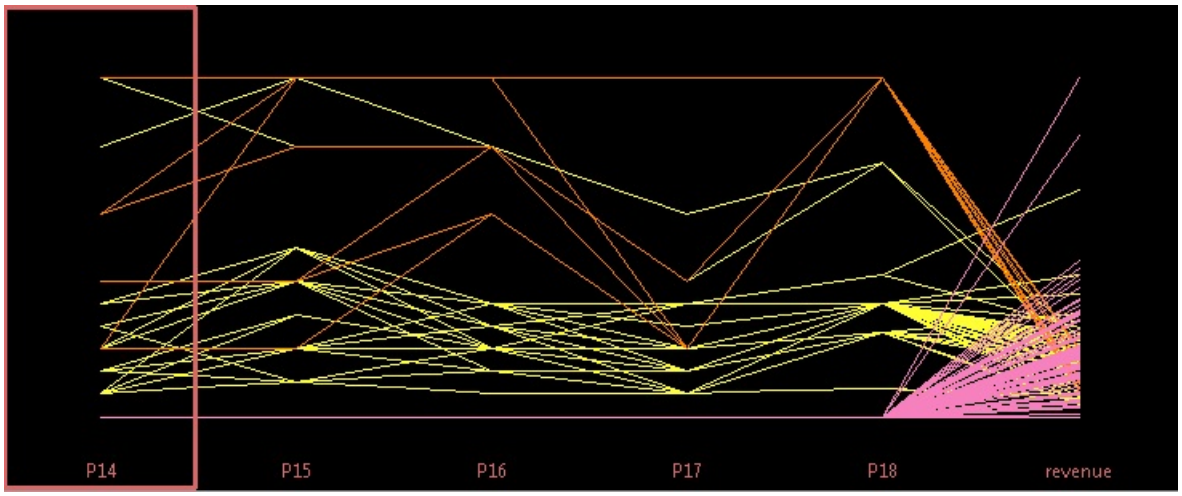

Figure 5: Parallel Coordinates plot using `ggobi` for variables P14-P18 and ending revenue. The orange cluster lines indicates a brushed cluster on P18 while the pink cluster lines indicate the zero value brushed cluster.

3.6 Dimension Reduction in P-Variables

Principal component analysis can be used to reduct the dimension of the data. This is especially useful for the p variables, because we are given no information about what they may represent, and so could themselves be linear combinationa of the "real" variables, or simply be highly dependent. There could be some issues with using PCA because the P variables are considered categorical variables, and PCA is designed for continous variables.

To compute the principal components, the eigenvalues and eigenvectors of the correlation matrix (C) are computed:

$$V^{-1}CV = D$$

, where D is the diagonal matrix of eigenvectors and V is the matrix of eigenaalues

These eigenvectors and eigenvalues are then sorted in decreasing order, with the eigenvector associated with the largest egienvalue being the first principal component, the second largest being the second principal component, and so on. The eigenvalue corresponds to the proportion of variance explaned by that principal component. A number of eigenvalues can be selected instead of the original variables, which causes a loss of information but allows the data to be explained in fewer variables. By doing prediction on these eigenvalues instead of the original variables, algorithms can run faster because of the smaller size of the dataset, and this can sometimes reduce noise in the data, leading to less overfitting.

A test was ran on the training set, using a random sample of 30% test and 70% training to get an idea how effective PCA would be. See figure 5. This test was highly simplified, using only the p-variables for prediction of revenue, but is useful to get an idea if PCA may be useful, and if so what would be a good number of principal components to keep. The results of the test show that even using .75 percent of the variance does not result in a significant decline in predictive power of the rf model.

```
#Do for 10 reps
for(j in 1:n){
  #partition the data into training and test
  trainIndex <- createDataPartition(y = x.train.weird$revenue, p = 0.7,
     list = FALSE)
  train <- x.train.weird[trainIndex, c(4:40,45) ]
  test <- x.train.weird[-trainIndex, c(4:40,45) ]
```

```
  #train the baseline random forest model
  model<-train(revenue~., method = 'rf', data = train)
  myvars <- names(test) %in% c("revenue")
  prediction<-predict(model, test[,!myvars])
  testRMSE<-sqrt( mean( (test$revenue-prediction)^2 ) )
  results[2,j]<-testRMSE

  #train the model using PCA
  pcaModel<-train(revenue~., method = 'rf', data = train, preProcess
     ='pca', thresh = 0.85)
  myvars <- names(test) %in% c("revenue")
  prediction<-predict(pcaModel, test[,!myvars])
  testRMSE<-sqrt( mean( (test$revenue-prediction)^2 ) )
  results[1,j]<-testRMSE
  }
#Compute the average RMSE for baseline and PCA
results[,n+1]<-apply((results[,]),1,sum)/n
```

P value cut off	Components	RMSE	baseline RMSE
.95	10	2695330	2593924
.90	5	2463386	2511265
.85	3	2777711	2730831
.80	2	2233936	2234563
.75	2	2805476	2794885

Figure 6: Random Forest results for training and test-training set. The training was partitioned into 70/30 and mean baseline RMSE along with mean Principal component RMSE for the test-train set were calculated. The P Value cutoff indicates the % of variance retained from the original dataset and components column indicate the number of principal components fed into the final model

4 Models

4.1 Random Forest

One of the models we are focusing on is random forest. Random forests are promising because they, have desirable characteristics, such as running efficiently on large datasets Breiman (2001), and flexibility, having been used effectively for a variety of applications, including multi class image recognitionGall et al. (2011), 3D face analysisFanelli et al. (2013), land classificationGhimire et al. (2012) and disease predictionGray et al. (2013).

A random forest is an ensemble of decision trees created using random variable selection and bootstrap aggregating (bagging). What this means is that first a group of decision trees are created. For each individual tree, a random sample with replacement of the training data is used for training. Also, at each node of the tree, the split is created by only looking at a random subset of the variables. A commonly used number for each split is the square root of the number of variables.The prediction is made by averaging the predictions of all the individual trees. Breiman (2001)

Random forests can also provide an error estimate, called the out-of-bag error. This is computed by feeding the individual trees cases that they havent seen. Because each tree is created with a bootstrap sample, we expect about 1/3 of the samples to not be included in the training for any

given tree. The training data that was not included in the bootstrap sample for a given tree is fed through that tree, and a prediction is made. The results of doing this for all trees are computed, and for each sample (row) an out of bag estimate is created by averaging the results of all the trees. The proportion of incorrect classification gives an estimate of the error rate.

Another important aspect of random forests is their ability to rank variable importance. The variable importance is computed by finding the out-of-bag accuracy, and then by using a permutation test on the variable, by randomly permuting the out-of-bag values for a given variable, and then measuring the accuracy again. The difference in accuracy gives an estimate of the importance of the variable. Breiman (2001)

4.2 SVM

Support vector machines are linear classification models that attempt to perform classification by separating the classes with a hyperplane, such that the division between classes as wide as possible Hastie et al. (2009). Given data in the form

$$(x_i, y_i), x_i \in R^p, y_i \in -1, 1$$

i.e. x_i is the feature vector, and y_i is the class the observation belongs to.

In the case where the data is linearly separable, the SVM wants to find two hyperplanes separating the data such that the distance between them is maximized. Because any given hyperplane can be denoted:

$$w \bullet x - b = 0$$

The two hyperplanes can be denoted:

$$w \bullet x - b = -1$$

$$w \bullet x - b = 1$$

The distance between these hyperplanes is $\frac{2}{||w||}$, which is maximized by minimizing $||w||$ subject to:

$$y_i(w \bullet x_i - b) - 1 \forall i$$

By substituting $||w||$ with $\frac{||w||^2}{2}$, this results in a problem that can be solved with quadratic optimization.

This above method can only find linear decision boundaries, but by using the Kernel method it is possible to do this with non-linear decision boundaries. This method involve changing the normal dot product to a kernel function. For example, in the case of a polynomial kernel of degree d, the dot product would change to $k(x_i, x_j) = (x_i, x_j)^d$

5 Ensemble Method

In machine learning problems a common way to improve accuracy is through ensembling multiple prediction algorithms. Ensemble learning involves creating base prediction algorithms, and then combining them in a way that leads to better accuracy.

Ensemble models can perform better than the models they are composed of if the models are independent, and if they get greater than 50% accuracy. This intuitively makes sense, as it shoud not be possible to increase accurracy by combining identical models, or by adding models that are not useful.When errors of classifiers are uncorrelated, majority voting using many classifiers reduces error rates, as long as classifiers have over 50

Because of the the increase in complexity associated with ensembling, the model becomes extremely hard to interpret, and also tends to run very slowly. However, the nature of Kaggle competitions do not factor speed and interpretability as a primary concern and thus, ensembling is a useful approach. Another ensemble model was created using a variety of machine learning algorithms. These were not chosen carefully like random forest and SVM were because they were known to perform well in this situation, but were chosen based on their ease of implementation and test-training set performance. This is a very black box style approach to prediction, but given the nature of the problem it was deemed likely to provide a resonably good prediction.

5.1 Dumb Model

A few different ways of ensembling models were tried. The most "dumb" approach was using a variety of models with all implemented with the caret package. These were trained using default settings, and ignoring all issues of missing values. All the data was converted to numeric, including variables known to be categorical such as city and type.

The models used were random forest, generalized linear model, generalized boosted regression model, boosted generalized linear model,k nearest neighbours, lasso, ridge regression, SVM with the radial basis function, and M5 rules. These models were trained on a subset of the training data, and then used to predict the remainder of the training set. The results of this prediction were then used to train the ensemble model. It is important that the ensemble model is trained on a different set than the original models were trained on in order to avoid overfitting. Next the models are trained on the full training set to increase the power of the model, and then test set predictions are made. These are then combined using the ensemble model, and the final prediction is made.

This is clearly not a good approach to the problem, but is interesting because it gives a baseline for how good you could expect a prediction algorithm to do with no preprocessing or intelligent feature selection. This model achieved a RMSE of 1752064, which was resonably good but nowhere near the top 10%.

```
#Model List
models<-c('rf', 'glm', 'gbm', 'glmboost', 'knn', 'lasso',
        'ridge', 'svmRadial', 'M5Rules')
#create data partition
trainIndex <- createDataPartition(y = trainDat$revenue, p = 0.7, list
   = FALSE)
train <- trainDat[trainIndex, ]
test <- trainDat[-trainIndex, ]
results<-as.data.frame(matrix(rep(0,(length(models)+1)*nrow(test),nrow(test)
   ,length(models)+1)))

#train the models on 70\% and predict on 30\%
for(j in 1:length(models)){
  model<-train(revenue~., method = models[j], data = train)

  myvars <- names(test) %in% c("revenue")
  prediction<-predict(model, test[,!myvars])

  results[,j]<-prediction
  print(j)
}

myvars <- names(test) %in% c("revenue")
```

```
results<-cbind(results, test[,myvars])

names(results)<-c(models,'revenue')

#Compute the ensemble model on the unused data to avoid overfitting
ensembleModel<- train(revenue~., method = 'lm', data = results,
   importance = TRUE, preProcess = "pca")

#### Now predict on the real set
train <- trainDat
test <- testBig
results<-as.data.frame(matrix(rep(0,(length(models)+1)*nrow(test),nrow(test)
   ,length(models)+1)))

for(j in 1:length(models)){
  assign(paste0(models[j],"model"), train(revenue~., method =
     models[j], data = train, preProcess = "pca"))

  prediction<-predict(get(paste0(models[j],"model")), test[,-1])

  results[,j]<-prediction
  print(j)
  }

names(results)<-c(models)

# Run the ensemble model on the results of the models
prediction<-predict(ensembleModel,results)
```

5.2 SVM and Random Forest Model

We focused on using two models, random forest and SVM. These models work well with ensembling because of their complexity leads to very high accuracy, and also because they are totally different types of models and so should have highly uncorrelated errors. In fact random forest is an ensemble model in and of itself because it works by combining many decision trees.

The SVM is implemented using the R `e1071` package, using the radial basis function. Random forest is implemented with the random forest package. The models are trained 100 times each, but with a different random sample of the data each time. This gives 100 different RF and SVM models, which are all tested on their test sets, and the RMSE of each model is recorded. The RF and SVM models with the lowest RMSE are used for the ensemble.

There are many different ways to combine models to make an ensemble. One of the most simple ways is using weighted averages of the predictions. Our initial model used this approach, by combing the RF and SVM output using an average weighted by the RMSE of the models:

$$R\alpha = \frac{RMSE_{SVM}}{RMSE_{SVM} + RMSE_{RF}}$$

$$Ensemble\ prediction = \alpha * prediction_{RF} + (1 - \alpha) * prediction_{SVM}$$

```
rmse.cv.rf <- c()
rmse.cv.svm <- c()
```

```
rmse.t.svm <- c()
vec.rf <- list()
vec.svm <- list()

for (i in 1:100){

  trainIndex <- createDataPartition(y = y.train, p = 0.7, list = FALSE)
  x.prac <- x.train.weird[trainIndex, ]
  y.prac <- y.train[trainIndex]
  x.cv <- x.train.weird[-trainIndex, ]
  y.cv <- y.train[-trainIndex]

  #Random Forest
  vec.rf[[i]] <- randomForest(y.prac~., data = x.prac, importance=T)
  rmse.cv.rf[i] <- sqrt( mean( (y.cv-predict(vec.rf[[i]], x.cv))^2 ) )

  #SVM
  x.prac <- model.matrix(y.prac~., data=x.prac)[,-1] # Remove Intercept
  x.cv <- model.matrix(y.cv~., data=x.cv)[,-1] # Remove Intercept
  scaling <- c(TRUE,TRUE,FALSE,FALSE,FALSE, rep(TRUE, 20),
     rep(FALSE,11) , TRUE)
  vec.svm[[i]] <- ksvm(x.prac, y.prac, type="eps-svr", kernel="rbfdot",
     scaled=scaling)
  rmse.cv.svm[i] <- (sqrt(mean((y.cv-predict(vec.svm[[i]] , x.cv))^2) )
     )
  rmse.t.svm[i] <- (sqrt(mean((y.cv-predict(vec.svm[[i]] , x.prac))^2)
     ) )
}

  min(rmse.cv.svm)
  min(rmse.cv.rf)
```

6 Results

Our initial model was using random forest in `caret` package with default settings and the only variable preprocessing was to code all *city* and *type* instances that were in the test set but not in the training set as missing. This was done through dummy variable encoding where if there are M numer of factors in variable X, then $M-1$ dummy variables were generated such that $D_m = 1 \quad iff \quad m \neq M-1 \quad else \quad D_m = 0$.

The starting random forest model was used to give a baseline for what we should expect performance of superior models to be, and achieved a RMSE of 1,851,510.63. Using a "dumb" ensemble model of 9 different models but no variable preprocessing surpassed this model, with a RMSE of 1752064.31. The results of these models were combined using linear regression fitted on the CV set to determine optimal weights. See figure 6 for full results table. We make the following important observations for our test sets:

- Over all iterations and combinations of treatments for both variables and model selecton, it seems that the best set was using all proposed treatments except trying to parse the zero problem.

- Using simple treatments for City and Type with only two models greatly improved performance even over the dumb baseline model. We attribute this mostly to the effectiveness of our treatments rather than pure luck.
- Log transformation did not improve real test set performance although test-training set results were very promising.
- Treating P-Variables with the Zero Problem with both PCA and kNN did not improve performance. We hypothesize that this is due to large misspecification errors of the treatment models that introduces more noise rather than clarity.

P-Variable Treatment	**City Problem Treatment**	**Type Treatment**	**Models Used**	**Ensemble Weighting**	**Submission RMSE**
Baseline Models					
None	Dummy Variable Encoding	Dummy Variable Encoding	Random Forest	None	1,851,510
None	Numeric	Dummy Variable Encoding	rf, glm, gbm, glmboost, knn, lasso, ridge, svmRadial, M5Rules	linear regression	1,752,064
Basic Transformations					
None	Dummy Variable Encoding	Dummy Variable Encoding	Random Forest (Log Revenue)	None	1,751,385
None	Dummy Variable Encoding	Dummy Variable Encoding	Random Forest w/CV	None	1,956,114
Proposed Treatment Models					
None	K Means (K=22)	Set to IL	Random Forest (Log Revenue)	None	1,742,138
None	K Means (K=20)	kNN Treatment (K=5)	Random Forest, SVM (100 reps), take best CV	Weighted	1,658,797
None	K Means (K=20)	kNN Treatment (K=5)	Random Forest, SVM (100 reps), take best CV, (Log of Revenue)	Weighted	1,713,659
KNN (K=3) for Zero Problem	K Means (K=20)	kNN Treatment (K=5)	Random Forest, SVM (100 reps), take best CV	Weighted average	1,721,388
KNN (K=1) for Zero Problem	K Means (K=20)	kNN Treatment (K=5)	Random Forest, SVM (100 reps), take best CV	Weighted average	1,827,636
PCA Processing with 20 Principal Components	K Means (K=20)	kNN Treatment (K=5)	Random Forest, SVM (100 reps), take best CV	Weighted average	1,934,188

Figure 7: Test set results for all Kaggle Submissions on TFI Food. Each row is a submission result and columns denote the methods used.

7 Conclusion

The TFI competition was a tough challenge to overcome in terms of both model selection and feature engineering. The lack of information contributed to the blackbox approach of creating assumptions and throwing in models until test performance was improved upon. In data preprocessing, we examined certain characteristics of the data set and provided potential fixes with PCA, K-Means clustering and kNN classification. Within Model selection, we focused primarily upon SVM and Random Forest implementations as they performed the best in the "dumb" model. We

then used simple ensemble techniques such as linear regression and weighted RMSE to craft our final test set of predictions. The lowest RMSE we were able to obtain was 1,658,797 which were able to push us from 500th place to 60th place, at this time of writing. We attribute this to a combination of both luck and good variable preprocessing.

7.1 Limitations and Personal Thoughts

When trying to solve a prediction problem of this type, we realize the importance of data preprocessing in comparision to the implementation of the machine learning algorithm itself. Through simple R package implementations, it is relatively easy to implement an ensemble of good models and get a RMSE above baseline. However, to further lower the RSME, preprocessing is necessary. For example, the key in changing the performance of our model from being near the middle 50% of competitors to the top 10% was realizing that both K-Means clustering and kNN treatments could be used to deal with the Unaccounted problem. In this competition, preprocessing was difficult because of the nature of the obfuscated variables. This made data visualization much more difficult, complicated the process of imputing variables, and made interpretation of the model quite dificult. The lack of information about the P-variables made it necessary to use trial and error based approaches to categorizing and preprocessing these variables, by comparing the approaches based only on how it helps the predictive power of the model.

Another important consideration was that the approach used for a Kaggle competion may not be suitable for many real applications. In reality lowering the RMSE is not the only consideration when creating a model, interpretability and speed also matter. Furthermore, the inability to gather more information from different sources hindered our development progress. We believe the constraints that we faced in this competition, albeit allowed us to understand our models better, would not be something that appears regularly in real life use-cases. Real life cases would allow us to use more tractable models on more meaningful features such that the results are interpretable with supporting economic rationale.

8 References

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

David L. Davies and Donald W. Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1(2):224–227, April 1979. ISSN 0162-8828. doi: 10.1109/TPAMI.1979.4766909.

Gabriele Fanelli, Matthias Dantone, Juergen Gall, Andrea Fossati, and Luc Van Gool. Random forests for real time 3d face analysis. *International Journal of Computer Vision*, 101(3):437–458, 2013.

Juergen Gall, Angela Yao, Nima Razavi, Luc Van Gool, and Victor Lempitsky. Hough forests for object detection, tracking, and action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(11):2188–2202, 2011.

Bardan Ghimire, John Rogan, Víctor Rodríguez Galiano, Prajjwal Panday, and Neeti Neeti. An evaluation of bagging, boosting, and random forests for land-cover classification in cape cod, massachusetts, usa. *GIScience & Remote Sensing*, 49(5):623–643, 2012.

Katherine R Gray, Paul Aljabar, Rolf A Heckemann, Alexander Hammers, Daniel Rueckert, Alzheimer's Disease Neuroimaging Initiative, et al. Random forest-based similarity measures for multi-modal classification of alzheimer's disease. *NeuroImage*, 65:167–175, 2013.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

J. MacQueen. Some methods for classification and analysis of multivariate observations, 1967. URL `http://projecteuclid.org/euclid.bsmsp/1200512992`.

S. Mills. Davies-bouldin r code, 2015. URL `http://people.math.carleton.ca/~smills/`.