STAT430 Final Project: Why did you have to leave me? - Using Machine Learning to Predict and Explain Employee Turnover

Shangting Li, Shuyu Wang, Ruojie Zeng, Yang Yuan

December 23, 2017

Abstract

This report illustrates methodologies to investigate factors contributing to employee turnover. We first build and compare predictive models using various statistical learning techniques. Outputs of the selected model based on test data accuracy are discussed to provide actionable recommendations. Using the predictive model, we are able to make predictions on current employees decision given information of significant factors.

Introduction

The ability of an organization to attract new employees demonstrates the value of its branding. However, to evaluate how an organization is really performing from the inside, it is more important to know the employee retention rate, which refers to the ability of an organization to retain the current employees. Moreover, since an organization usually invests a fairly huge amount of money on training the new employees, a higher employee retention rate can also help reduce this cost. Therefore, in order to improve the employee retention, it is necessary for human resources departments to investigate potential elements that may influence employees' decisions of whether or not to leave the company.

Although there are external factors that may lead to emplyees leaving, such as political or technological change of the industry, it is not easy, if at all possible, to control these factors. Here, we would like to study the internal factors, such as working hours, salary and promotions, and see how these factors influence employees' decisions. We hope that through the analysis of these factors, companies would be able to respond to the major internal factors by changing their policies or design new rewards systems accordingly.

Our analysis seeks to answer the following questions:

- What are the most important factors contributing to staff turnover?
- Who is leaving?
- Why do valuable employees leave?
- Can we build a predictive model and use it to identify elite employees who are likely to leave?

Materials and Methods

Data Exploration

Description of Dataset

For this project, the dataset we are going to use is `HR_comma_sep.csv`. This dataset is simulated by `Ludovic benistant`, which can be downloaded from kaggle. In this dataset, there are in total of 15,000 observations and 10 variables. Among these variables, we would like to set the left variable, which is a factor variable

indicating whether the worker left the company (1 for left and 0 for stay), to be the response variable. We would use the rest of the variables as predictors.The description of the variables are as follows:

- *Satisfaction_level*: An index ranging from 0 to 1. The higher the value, the higher the satisfaction level of the worker in the company. [Numeric]
- *Last_evaluation*: An index ranging from 0 to 1.The higher the value, the higher the evaluation of the worker by the company is. [Numeric]
- *Num_project*: Number of projects the workers have participated in. [Numeric]
- *Average_monthly_hours*: Average working hours per month.[Numeric]
- *Time_spend_company*: Number of years spent in this company. [Numeric]
- *Work_accident*: Whether the worker has ever encountered work accident (1 for true and 0 for false). [Factor]
- *Promotion_last_5years*: Whether the worker has ever received promotion in the last 5 years (1 for true and 0 for false). [Factor]
- *Sales*: The working position in the company. [Factor]
- *Salary*: The Salary levels of the workers (Three levels: low, medium, high). [Factor]

Data Preparation:

We would split the data in half. The first half of the data is used for training of potentially useful models, and the second one is used for testing the performance of models. We will randomly select half of the data to fit a model, and use the rest to evaluate the performance of the fitted model.

```
> data_HR = read.csv('HR_comma_sep.csv')
> factor_cols = c('Work_accident', 'left', 'promotion_last_5years', 'sales', 'salary')
> data_HR[,factor_cols] = data.frame(apply(data_HR[factor_cols], 2, as.factor))
> colnames(data_HR)= c("satisfaction","evaluation","projects","hours","years","accident",
+                      "left","promotion","sales","salary")
> train_index = sample(1:nrow(data_HR), size = round(0.5 * nrow(data_HR)))
> train_HR = data_HR[train_index, ]
> test_HR = data_HR[-train_index, ]
```

Here we change some variable names for tidiness and clarification. In addition, we coerce some variables into factor variable.

Methodology

Models Considered

Below are the potentially useful classification models we looked into:

- Logistic Regressions
- K-Nearest-Neighbors(KNN)
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Naive Bayes
- Tree-Based Methods
- Support Vector Machines (SVM)

Figure 1 shows all the details of the models we would look into.

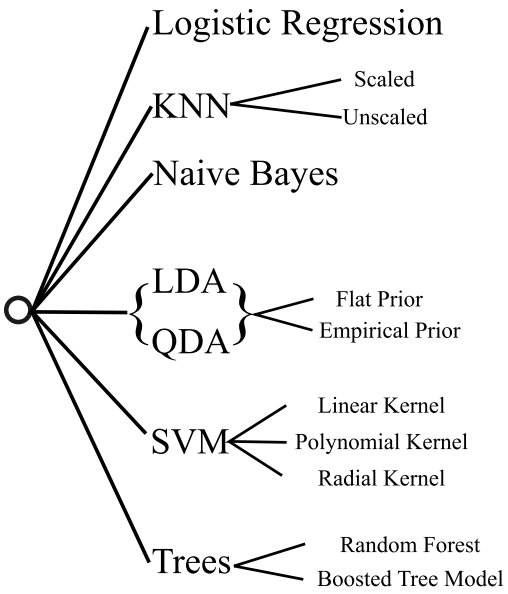

Figure 1: Models used for our analysis

Cross Validation

For models with tuning parameters, we use the 5-fold cross validation to tune these parameters which maximizes the performances of the models. For example, for random forest model we would discuss in the chapter, we tune the number of parameters randomly selected in developing the trees.

Model Training and Model Performances

Logistic Regression:

First we tried the binomial logistic regression model. The binomial logistic regression approach estimates the probability of a binary response based on the values of the predictors. If the probability is larger than 0.5, the model predicts that $left = 1$ and $left = 0$ otherwise.

```
> glm = glm(left ~ ., train_HR, family = "binomial")
> prob = predict(glm,newdata = test_HR,type = 'response')
> pred = ifelse(prob > 0.5, 1, 0)
> acc_glm = ClassAcc(pred,test_HR$left)
> acc_glm
```

```
## [1] 0.7931724
```

Here, the testing accuracy is 0.79. This is a reasonably well accuracy. However, for a simulated data the accuracy could be potentially higher.

K-Nearest-Neighbors

Next, we tried the k-nearest neighbors, a non-parametric method. In this approach, the classification of an observation is determined by the most common class of its k nearest neighbors. Here, we fit two KNN models. In the first model, we scaled the predictors in order to have mean of zero and unit variance. In the second model, we left it unscaled.

```
> #Scaled Result for KNN
> set.seed(seed)
> knn_scale = train(
+   left ~ .,
+   data = train_HR,
+   method = "knn",
+   trControl = trainControl(method = "cv", number = 5),
+   preProcess = c("center", "scale")
+ )
> cv_knn_scale = get_best_result(knn_scale)$Accuracy
>
> acc_knn_scale = ClassAcc(
+   predict(knn_scale, test_HR),
+   test_HR$left
+ )
> #Unscaled Result for KNN
> set.seed(seed)
> knn_unscale = train(
+   left ~ .,
+   data = train_HR,
+   method = "knn",
+   trControl = trainControl(method = "cv", number = 5)
+ )
> cv_knn_unscale = get_best_result(knn_unscale)$Accuracy
>
> acc_knn_unscale = ClassAcc(
+   predict(knn_unscale, test_HR),
+   test_HR$left
+ )
```

Training Accuracy and Testing Accuracy of KNN Model for scaled data and unscaled data:

```
> c(cv_knn_scale,cv_knn_unscale)
```

```
## [1] 0.9322683 0.9329354
```

```
> c(acc_knn_scale,acc_knn_unscale)
```

```
## [1] 0.9362582 0.9371916
```

We could see KNN model has a much better testing accuracy compared with logistic regression model. In addition, we could see both testing and training accuracy for the KNN model increases if the data is scaled. So in this case, we are justified to use scaled data over unscaled data to train the KNN model.

Linear Discriminant Analysis

LDA (Linear Discriminant Analysis) approach for two classes assumes that the conditional probability density functions for both classes are both normally distributed with respective parameters. LDA also assumes that the covariance matrix of each class is identical.

Here, we fit two LDA models, one with priors estimated from data and the other onw with flat prior, where prior refers to the proportion of the two classes in the dataset.

```
> flat= c(1,1)/2
> lda_norm = lda(left ~ ., train_HR)
> acc_lda_norm = ClassAcc(predict(lda_norm, test_HR)$class, test_HR$left)
```

```
> #LDA (with Priors estimated from data)
> lda_flat = lda(left ~ ., train_HR,prior=flat)
> acc_lda_flat = ClassAcc(predict(lda_flat, test_HR)$class, test_HR$left)
> #LDA with Flat Prior
> c(acc_lda_norm,acc_lda_flat)
```

```
## [1] 0.7814375 0.7455661
```

No matter which prior we use, the accuracy of the linear discriminant model seems inferior compared with the KNN model above. One possible reason is that the assumption held by LDA is not correct in this case. We could imagine more often than not, the group of people leaving the company should have different covariance matrix compared with the group of people staying in the company. As a result, in this case a QDA model might be more appropriate.

Quadratic Discriminant Analysis

Different from LDA, QDA (Quadratic Discriminant Analysis) approach does not assume identical covariance for each class. Instead, different covariance is allowed for each class.

We fit two models for QDA, one with priors estimated from data and another one with flat prior.

```
> qda_norm = qda(left ~ ., train_HR)
> acc_qda_norm = ClassAcc(predict(qda_norm, test_HR)$class, test_HR$left)
> #QDA (with Priors estimated from data)
> qda_flat = qda(left ~ ., train_HR, prior=flat)
> acc_qda_flat = ClassAcc(predict(qda_flat, test_HR)$class, test_HR$left)
> #QDA (with Flat Prior)
> c(acc_qda_norm,acc_qda_flat)
```

```
## [1] 0.8771836 0.8442459
```

Here we see that QDA performs much better than LDA, as we expected. k appear to be very different for different factors of *left*. In addition, we notice that the empirical prior performs better, so it is highly possible that the proportion of factors for *left* in test data is not uniform.

Support Vector Machines

Support vector machines are supervised learning models. It strives to find a hyperplane that could most efficiently separate two classes in given data. In support vector machines, the feature spaces of the data, or the dimensions the data are projected, are decided by the kernel functions used. Here we would like to explore three types of kernels: Linear, Polynomial and Radial.

Support Vector Machines with Linear Kernel

```
> set.seed(seed)
> lin_grid = expand.grid(C = c(2 ^ (-5:5)))
> if (FALSE){
+ svmLinear_tuned = train(
+     left~.,
+     data = train_HR,
+     method = 'svmLinear',
+     trControl = trainControl(method = "cv", number = 5)
+ )
```

```
+ }
> #Large Cross Validation Model.
> #Takes A Long Time to Build So we Saved it and Call it whenever Needed.
> #You are welcomed to see if the model trained
> #is the same as the model we loaded.
> svmLinear_tuned= readRDS("svmLinear.rds")
> pred = predict(svmLinear_tuned,test_HR)
> acc_svmLinear = ClassAcc(pred,test_HR$left)
> acc_svmLinear
```

```
## [1] 0.7786372
```

We could see the support vector machine with linear kernel performs not very well. This happens possibly because that the two classes are comparably linear-inseparable in feature space. To confirm our hypothesis, we also try polynomial kernel and radial kernel for support vector machines.

Support Vector Machines with Polynomial Kernel

```
> set.seed(seed)
> if (FALSE){
+ svmPoly_tuned = train(
+     left~.,
+     data = train_HR,
+     method = 'svmPoly',
+     trControl = trainControl(method = "cv", number = 5)
+ )
+ }
> svmPoly_tuned = readRDS("svmPoly.rds")
> #Large Cross Validation Model.
> #Takes A Long Time to Build So we Saved it and Call it whenever Needed.
> #You are welcomed to see if the model trained
> #is the same as the model we loaded.
> pred = predict(svmPoly_tuned,test_HR)
> acc_svmPoly = ClassAcc(pred,test_HR$left)
> acc_svmPoly
```

```
## [1] 0.9611948
```

Support Vector Machines with Radial Kernel

```
> set.seed(seed)
> if (FALSE){
+ svmRadial_tuned = train(
+     left~.,
+     data = train_HR,
+     method = 'svmRadial',
+     trControl = trainControl(method = "cv", number = 5)
+ )
+ }
> svmRadial_tuned = readRDS("svmRadial.rds")
> #Large Cross Validation Model.
> #Takes A Long Time to Build So we Saved it and Call it whenever Needed.
```

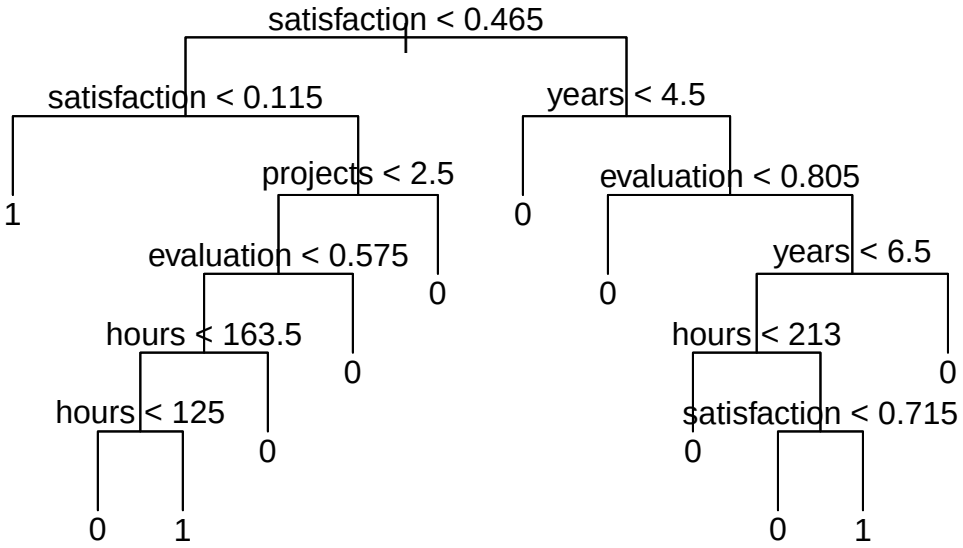

Figure 2: Unpruned Classification Tree

```
> #You are welcomed to see if the model trained
> #is the same as the model we loaded.
> pred = predict(svmRadial_tuned,test_HR)
> acc_svmRadial = ClassAcc(pred,test_HR$left)
> acc_svmRadial
```

```
## [1] 0.9477264
```

We could see this two kernels perform way better than the linear kernel, which justifies our assumption. By projecting the data into higher dimensions, the two classes of people could be almost separated and this gives a testing accuracy better than even KNN models.

Trees

It is also very intuitive for us to fit tree-based models because in our dataset there are many factor variables. These models are very easy to interpret. For example, the following code produced a sample decision tree of classification. We could see the nearer the variable condition is to the root, the more important is the variable in influencing classification.

Tree Example

```
> exp_tree = tree(left ~ ., data = train_HR)
> plot(exp_tree,type= c("uniform"))
> text(exp_tree, pretty = 4)
```

However, as we discussed in class, a single tree suffers from high prediction variance since sometimes slight changes in the variables' values could significantly change the results. Therefore, we would explore the tree methods improved by ensemble methods: Bagging and Boosting.

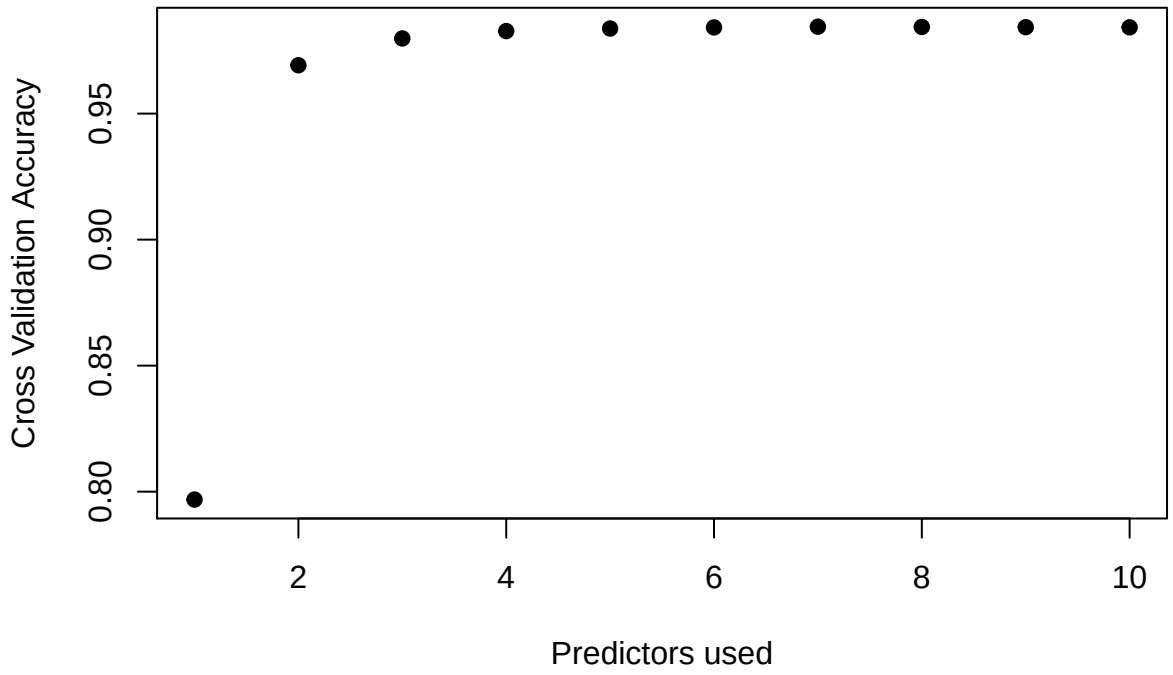

Figure 3: Cross Validation Accuracy vs Number of Predictors used

Random Forest

Random forest constructs multiple decision trees for training and outputs the mode of the classes for the purpose of classification.

```
> set.seed(seed)
> tunegrid = expand.grid(.mtry = c(1:10))
> if (FALSE){
+ rf = train(left~., data=train_HR, method = 'rf', tuneGrid=tunegrid,
+            trConrol = trainControl(method = 'oob')
+ )
+ }
> rf = readRDS("caretRF.rds")
> #Large Cross Validation Model.
> #Takes A Long Time to Build So we Saved it and Call it whenever Needed.
> #You are welcomed to see if the model trained
> #is the same as the model we loaded.
> best_mtry = get_best_result(rf)$mtry
> best_rf = randomForest(left ~ ., train_HR,mtry = best_mtry)
> plot(rf$results$mtry,rf$results$Accuracy,pch = 19,xlab = 'Predictors used',
+      ylab = 'Cross Validation Accuracy'
+ )
```

After cross validation, we could see the best selection for number of randomly selected predictors to use is 9 to optimize model performance. As a result, we would fit a model using $mtry = 9$

```
> best_rf = randomForest(left ~ ., train_HR,mtry = best_mtry)
> pred_rf = predict(rf, newdata = test_HR)
> acc_rf = ClassAcc(pred_rf,test_HR$left)
> acc_rf
```

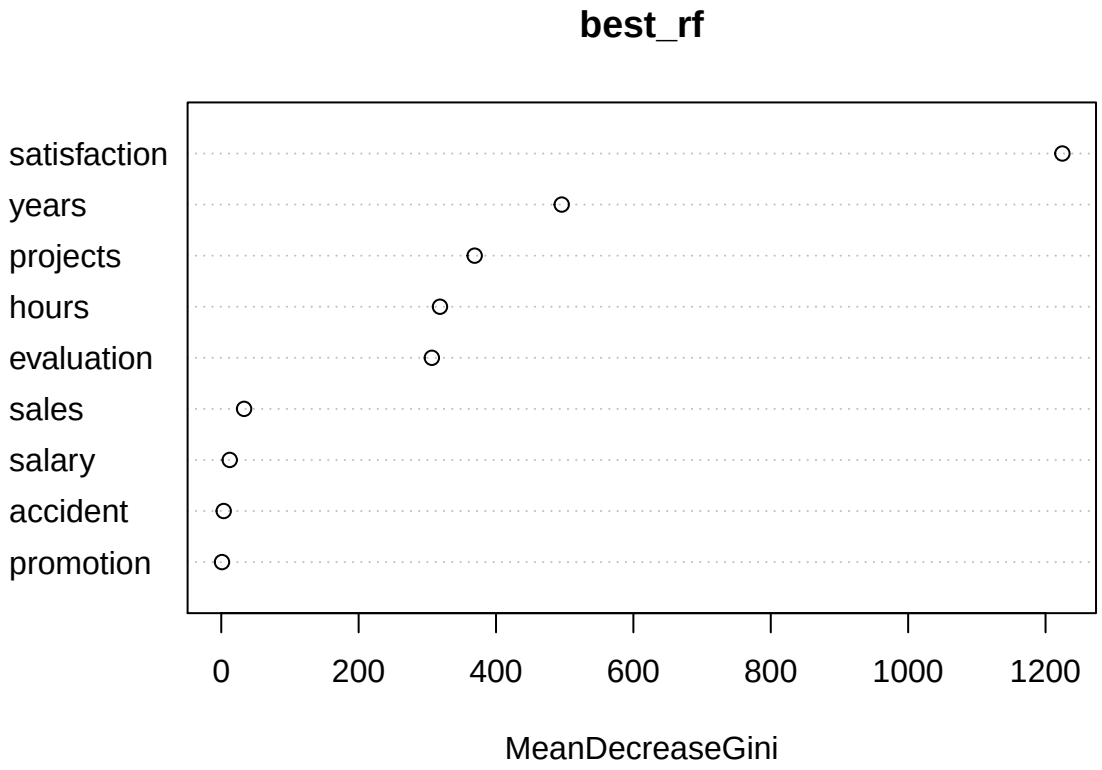

Figure 4: Ranking of factors by importance

```
## [1] 0.9933324
```

We could see that the random forest method performs outstandingly well, and it beats all the methods above. Also, from the random forest we could get a collective ranking for variable importance.

```
> varImpPlot(best_rf)
```

From Figure 4 we could see that the top three most important factors that is influencing the workers' decisions are their satisfaction levels, number of years in the company, and the evaluation they have received from the employers.

Boosted Tree Model

Similar with bagging, boosting also creates multiple trees. However, in the boosting approach, each tree is grown based on information of the previously grown trees.

```
> set.seed(seed)
> gbm_grid = expand.grid(interaction.depth = c(1, 2),
+                        n.trees = c(500, 1000, 1500),
+                        shrinkage = c(0.001, 0.01, 0.1),
+                        n.minobsinnode = 10)
> if (FALSE){
+ gbm = train(left ~ ., data = train_HR,
+              method = "gbm",
+              trControl = trainControl(method = "cv", number = 5),
+              verbose = FALSE,
+              tuneGrid = gbm_grid)
+ }
```

```
> gbm = readRDS('gbm.rds')
> #Large Cross Validation Model.
> #Takes A Long Time to Build So we Saved it and Call it whenever Needed.
> #You are welcomed to see if the model trained
> #is the same as the model we loaded.
> pred_gbm = predict(gbm, newdata = test_HR)
> acc_gbm = ClassAcc(pred_gbm,test_HR$left)
> acc_gbm
```

```
## [1] 0.9778637
```

Boosted tree model also performs very well compared to other models.

Naive Bayes

Finally, Naive bayes approach assumes that the predictors are independent. The strength of Naive bayes is its ability to deal with a large number of predictors.

```
> nb_mod = train(
+  left ~ .,
+  data = train_HR,
+  trControl = trainControl(method = "cv", number = 5),
+  method = "nb"
+ )
> acc_nb_tuned =get_best_result(nb_mod)$Accuracy
> acc_nb_tuned
```

```
## [1] 0.9089344
```

Here NB does not perform compared to tree methods and Support vector machines. One possible reason is that there are some variables with significant dependencies. For example, satisfaction level, the most important variable decided in random forest model, could be related to salary, years, and evaluation received.

Results and Discussion

KNN Testing Accuracies

```
> accuracies = cbind(
+   Methods = c('KNN_scaled','KNN_unscaled'),
+   Accuracy = round(c(acc_knn_scale,acc_knn_unscale),4)
+ )
> colnames(accuracies) = c("Method", "Test Accuracy")
> knitr::kable(accuracies,
+       caption = "KNN Testing Accuracies",
+       booktabs = TRUE)
```

Table 1: KNN Testing Accuracies

Method	Test Accuracy
KNN_scaled	0.9363
KNN_unscaled	0.9372

So scaled knn performs much better than unscaled knn.

LDA and QDA Testing Accuracies

```
> accuracies = cbind(
+   Methods = c('LDA_Normal','LDA_Flat','QDA_Normal','QDA_Flat'),
+ Accuracy = round(c(acc_lda_norm,acc_lda_flat,acc_qda_norm,acc_qda_flat),4)
+ )
> colnames(accuracies) = c("Method", "Test Accuracy")
> knitr::kable(accuracies,
+       caption = "LDA and QDA Testing Accuracies",
+       booktabs = TRUE)
```

Table 2: LDA and QDA Testing Accuracies

Method	Test Accuracy
LDA_Normal	0.7814
LDA_Flat	0.7456
QDA_Normal	0.8772
QDA_Flat	0.8442

So QDA_normal performs the best among the four models.

Logistic Regression and Naive Bayes Testing Accuracies

```
> accuracies = cbind(
+   Methods = c('Logistic_Reg','Naive_Bayes'),
+   Accuracy = round(c(acc_glm,acc_nb_tuned),4)
+ )
> colnames(accuracies) = c("Method", "Test Accuracy")
> knitr::kable(accuracies,
+       caption = "Logistic Regression and Naive Bayes Testing Accuracies",
+       booktabs = TRUE)
```

Table 3: Logistic Regression and Naive Bayes Testing Accuracies

Method	Test Accuracy
Logistic_Reg	0.7932
Naive_Bayes	0.9089

SVM Testing Accuracies

```
> accuracies = cbind(
+   Methods = c('SVM_Linear','SVM_Poly','SVM_Radial'),
+   Accuracy = round(c(acc_svmLinear,acc_svmPoly,acc_svmRadial),4)
+ )
> colnames(accuracies) = c("Method", "Test Accuracy")
```

```
> knitr::kable(accuracies,
+       caption = "SVM Testing Accuracies",
+       booktabs = TRUE)
```

Table 4: SVM Testing Accuracies

Method	Test Accuracy
SVM_Linear	0.7786
SVM_Poly	0.9612
SVM_Radial	0.9477

Random forest and boosted tree Testing Accuracies

```
> accuracies = cbind(
+   Methods = c('random forest','boosted tree'),
+   Accuracy = round(c(acc_rf,acc_gbm),4)
+ )
> colnames(accuracies) = c("Method", "Test Accuracy")
> knitr::kable(accuracies,
+       caption = "SVM Testing Accuracies",
+       booktabs = TRUE)
```

Table 5: SVM Testing Accuracies

Method	Test Accuracy
random forest	0.9933
boosted tree	0.9779
Test Accuracy Su	mmery table

```
accuracies = cbind(
  Methods = c(
  'Logistic_Reg',
  'KNN_scaled',
  'KNN_unscaled',
  'Naive_Bayes',
  'LDA_Normal',
  'LDA_Flat',
  'QDA_Normal',
  'QDA_Flat',
  'SVM_Linear',
  'SVM_Poly',
  'SVM_Radial',
  'random forest',
  'boosted tree'
  ),
  Accuracy = round(
  c(
  acc_glm,
  acc_knn_scale,
  acc_knn_unscale,
```

```
  acc_nb_tuned,
  acc_lda_norm,
  acc_lda_flat,
  acc_qda_norm,
  acc_qda_flat,
  acc_svmLinear,
  acc_svmPoly,
  acc_svmRadial,
  acc_rf,
  acc_gbm
  ),4))
  colnames(accuracies) = c("Method", "Test Accuracy")
  knitr::kable(accuracies,
  caption = "Testing Accuracies",
  booktabs = TRUE)
```

Table 6: Testing Accuracies

Method	Test Accuracy
Logistic_Reg	0.7932
KNN_scaled	0.9363
KNN_unscaled	0.9372
Naive_Bayes	0.9089
LDA_Normal	0.7814
LDA_Flat	0.7456
QDA_Normal	0.8772
QDA_Flat	0.8442
SVM_Linear	0.7786
SVM_Poly	0.9612
SVM_Radial	0.9477
random forest	0.9933
boosted tree	0.9779

Here we see that random forest perform much better than boosted tree and all of the other models used with an accuracy equals to 0.9926657. So it is the best model to help companies predict if an employee would leave. Note that random forest is a non-parametric and discriminant method, since it doesn't make strong assumptions about the form of the mapping function and learns explicit boundaries between classes.

We know that GINI importance measures the average gain of purity by splits of a given variable. If the variable is useful, it tends to split mixed labeled nodes into pure single class nodes. Permuting a useful variable, tend to give relatively large decrease in mean gini-gain. Here based on the plot of variables VS mean decrease gini in Figure 4 , we can see that for the random forest model, the most important variable to influence employees' decisions is satisfaction. In addition, years, projects, evaluation and hours are also important to influence their decisions.

Density plot

To further investigate how different variables are influncing employees' decisions, we create density plots of the five most influential factors as discussed in the previous section.

```
> featurePlot(
+   x = train_HR[, c("satisfaction", "years", "projects", "hours", "evaluation")],
```

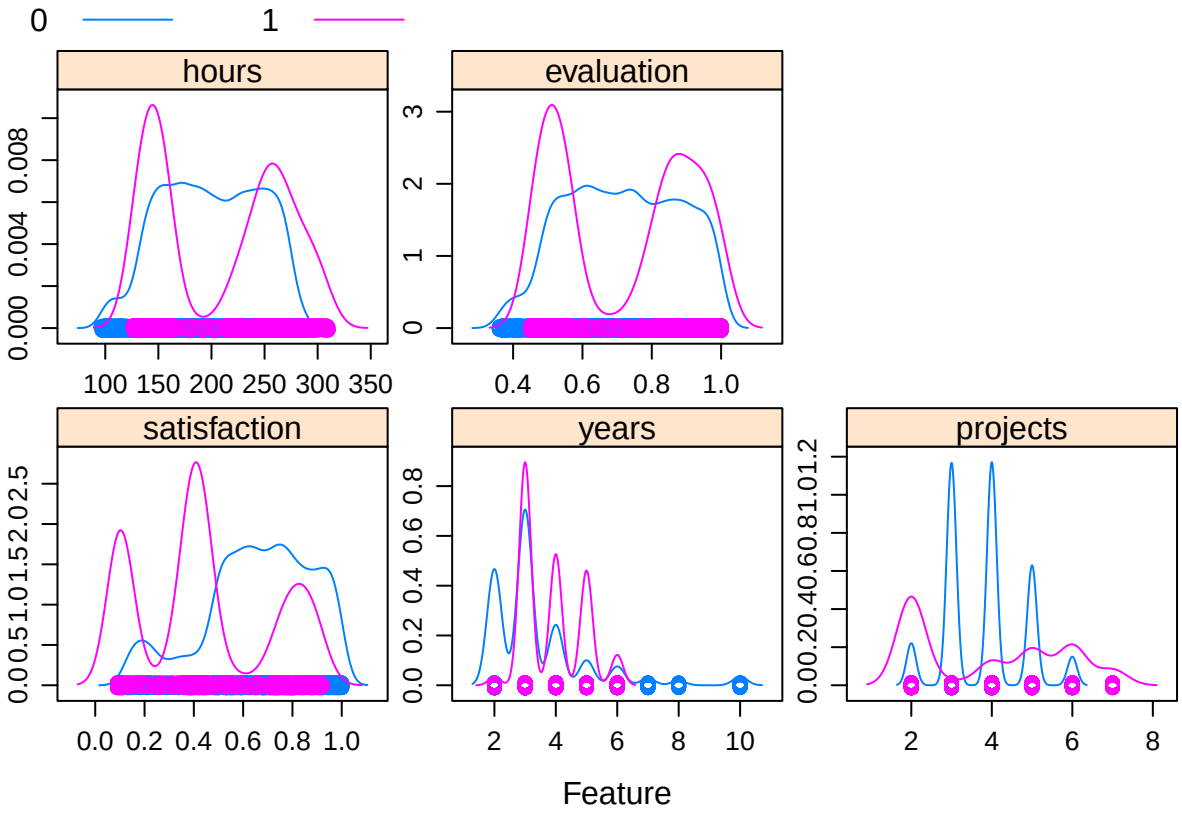

Figure 5: Density plot of five most important factors

```
+   train_HR$left,
+   plot = "density",
+   scales = list(
+   x = list(relation = "free"),
+   y = list(relation = "free")
+   ),
+   auto.key = list(columns = 5)
+   )
```

As shown in figure 5, those density plots visulize the distribution of each feature for people who have left(lines in purple) and those who stay (lines in blue).

- From `hours` plot, we can see people who left tend to have either much less working hours(an average of 150 hrs) or too many working hours (an average of 250 hrs.). The company should ensure a relative equal amount of workload among employees.
- From the `evaluation` distribution, we can see why it is not necessarily to retain every employee. Those high frequecies on the left are employees who don't work well. Whereas the bell shape on the right side of the plot indicates many good employees left as well. `They are the employees the company should have retained`.
- `satisfaction` plot shows that people who stay tend to have higher satisfaction, which is in line with the common sense. Note that there are some people with high satisfaction rate still left, which may be due to `external factors`. Further investigation and more information is needed to fully explain this result.
- `years` plot shows that most of the people who left had spent 3 to 5 years in the company. This is

problematic since the company had invested a lot on training and developing those employees. The company should profile those employees and figure out whether it is external or internal factors causing the turnover.

- From `projects` plot, the most noticable obervation is that more people left than stay when either they had few projects or too many projects. Similar to `hours` results, we would suggest the company to ensure a fair assignment of project among employees.

Conlusion and Recommendation

In this project, we focus on analyzing factors contributing to employees turnover and building predictive models for future turnover estimation. We first create and test classification models using various classification techniques, then choose the random forest model as the best to use since it has the highest test accuracy (99.2%) of all model we build. We also create density plots for the five most influntial variables to compare typical features of empolyees who left with those who stay.

Based on the results of analysis, we recommend the company to focus more on the employees' satisfaction levels, most recent evaluations and number of projects they have worked. In addition, companies should pay more attention to employees who have stayed for 3 to 5 years and try to keep the number of projects and workload in balance among employees.

References

David Dalpiaz. (2017). R for Statistical Learning

Venables, W. N.; Ripley, B. D. (2002). Modern Applied Statistics with S (4th ed.). Springer Verlag. ISBN 0-387-95457-0.