Machine Learning Engineer Nanodegree

Capstone Project

Bruno Xavier
June 27, 2018

I. Definition

Project Overview

The main domain of this Capstone project will be focusing on applying Supervised Learning algorithms to a real-world application. Supervised Learning mainly concerns of mapping a function Y=f(x) of input variable (x) to the output variable (Y) in that the model learns it so well that it could predict new oncoming variable (x) to predict (Y). Therefore, in Supervised Learning, the algorithm learns from a dataset that already has correct answers and learning goes iteratively until the algorithm reaches the acceptable performance level. Common industries using these methods are pharmaceutical, retail and cybersecurity companies.

With the exponential growth of data, companies cannot rely just on intuition on important financial decisions or to identify the root cause of ongoing issues or even identifying new consumer trends. Quite often these issues are co-related with others that a decision maker might never suspect are part of the problem. To stay afloat, companies need to invest in their employees in a balance of updated skills and employee wellbeing to retain them within the company. And these investments can be very costly to maintain, especially if the employee decides to leave the company. Employees are key for company growth and learning the reasons for employees to leave, also known as churning, is necessary to understand their reasons for leaving and remediate that or predict when they are about to leave. Kaggle's HR-Analytics dataset will be used in this Capstone.

Dataset: https://www.kaggle.com/colara/hr-analytics

Problem Statement

A common problem across businesses in many industries is that of employee churn. Businesses often must invest substantial amounts retaining and training employees, so every time an employee leaves it represents a significant investment lost. Both time and effort then need to be channeled into replacing them. Being able to predict when an employee is likely to leave and offer them incentives to stay can offer huge savings to a business. This is the essence of employee churn prediction; how can we quantify how and

when an employee is likely to churn? Groups initially will be divided into two categories, those employees who've left and those who've stayed and be treated as a binary classification Supervised Learning problem. As for the problem of why employees leave, the decision feature in which employees either have stayed or left will be put aside and the remaining features will be grouped into nodes and analyzed as to find out which factors increase churn risk. Logistic Regression will be used as an initial model for benchmarking and Random Forest, Decision Trees and SVM will be used to determine the best algorithm solution to this problem and compared to the benchmark model. The best algorithm should have higher recall , F1-Score and precision than all others.

Metrics

The dataset to be used is of Kaggle's HR Analytics challenge. It contains a total of 14999 rows and 10 columns. The main features for evaluation and its type are:

Variable	**Scaling**	**Description**
Satisfaction level	Numerical	Level of employee satisfaction
Last evaluation	Numerical	Last performance evaluation review score
Number of Projects	Numerical	Number of projects completed while at work
Avg. Monthly Hours	Numerical	Average monthly hours at workplace
Time Spend Company	Numerical	Number of years spent in the company
Work Accident	Numerical	Has the employee had a workplace accident
Promotion last 5 years	Numerical	Was employee promoted in the last 5 years
Sales	Categorical	Which department does employee work
Salary	Categorical	Relative level of salary (low, med, high)
Left	Categorical	Did the employee churn or not

On the numerical scaling side, satisfaction level is an important measurement as very likely correlate with churning, the more dissatisfied the employee is the more likely they'll leave. Last evaluation indicates when the employee's last feedback, positive or not, from the company which depending on the result the employee might be discouraged to work and leave. A similar concept would also apply to when the employee was last promoted and, in this dataset's case, if it happened in the last 5 years. Number of projects completed is somewhat abstract as it does not inform the quality or length of work but might indicate how engaged the employee is with the company. Similar concept is with average monthly hours worked, which might also show engagement with the company, but it might also reflect that the employee is not satisfied with current work hours and leave. Work accident is also a factor for the employee leaving but it does not inform us how serious the accident was.

On the categorical scaling side, which department the employee works might also be a factor of churning, albeit not necessarily a strong one, as some departments have a traditionally have a high turnover rate such as retail, accountancy and customer service. Employee salary level is also an important predictor such as that if the employee is not

satisfied with salary they'll likely leave. Lower salaries will likely have a higher turnover rate.
The feature "left" is what this model wants to predict. In this imbalanced dataset, there are currently 3751 employees who have left the company against 11428 who have stayed. This model will analyze the employee data and extract information on that will pinpoint those employees who are likely to leave and inform decision makers, so they can better decide which actions and policies to take and hopefully retain their employees or avoid similar cases in the future.

The models used will be measured by their accuracy, F1-score, precision and recall scores and compared against each other for the highest scores. Since this dataset is imbalanced, the accuracy score suffers from the accuracy paradox: Predictive Models with a given level of Accuracy may have greater Predictive Power than Models with higher Accuracy. (Afonja, 2017). Therefore, the accuracy score should be taken with a grain of salt.

Given that TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives:

Accuracy is defined as $A(M) = \frac{TN+TP}{TN+FP+FN+TP}$, which measures how often the model predicts correctly.

Recall = $\frac{TP}{TP+FN}$, which states how well the model identifies those who churn.

Precision = $\frac{TP}{TP+FP}$, which states how believable is the model.

F1 Score = $\frac{2*(Precision * Recall)}{Precision + Recall}$, which is the balance between precision and recall.

Another classification metrics considered was AUC-ROC (Area under the curve- Receiver Operating Characteristics) however it is not a good case to use as the dataset is imbalanced and it is not a good visual illustration as the false positive rates don't drop drastically when the total real negatives rate is very high.

Dataset: https://www.kaggle.com/colara/hr-analytics

II. Analysis

Data Exploration

In this section the dataset is imported, and a data sample is presented.

Initial checks on how much data there is, if there are any missing values, what type of data is within each column

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years
0.38	0.53	2	157	3	0	1	0
0.80	0.86	5	262	6	0	1	0
0.11	0.88	7	272	4	0	1	0
0.72	0.87	5	223	5	0	1	0
0.37	0.52	2	159	3	0	1	0

Figure 1

And using df.info() to extract basic dataset information such as number of entries and columns as well as each feature and their cell type:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
satisfaction_level       14999 non-null float64
last_evaluation          14999 non-null float64
number_project           14999 non-null int64
average_montly_hours     14999 non-null int64
time_spend_company       14999 non-null int64
Work_accident            14999 non-null int64
left                     14999 non-null int64
promotion_last_5years    14999 non-null int64
sales                    14999 non-null object
salary                   14999 non-null object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

14999 Entries with 10 columns were found along with sales and salary data types are categorical while the others are numerical. Categorical data types need to be transformed to numerical with the use of one hot encoding which will be discussed in a later section.

The next step would be finding if there are any null cells which there weren't. Then we proceed on to acquire basic information regarding the target variable 'left' and we get:

# of people who have left = 3571
# of people who have stayed = 11428
% of people who have left = 24%

And finally, for extracting basic statistical data from the dataset we execute df.describe() and obtain:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000

Figure 2

Which provides us with a count, mean, std, min and max.

Exploratory Visualization

This section analyzes the correlation between the different features and how it relates to churning by targeting the 'left' feature. Various graphs are presented for further interpretation.

Initially the most important graph needed will be the Correlation Coefficients Heatmap in which determines the correlation between features and how strong they are. It turns out somewhat unsurprisingly that Employee satisfaction level plays a greater role in determining employee churn than any other coefficient.

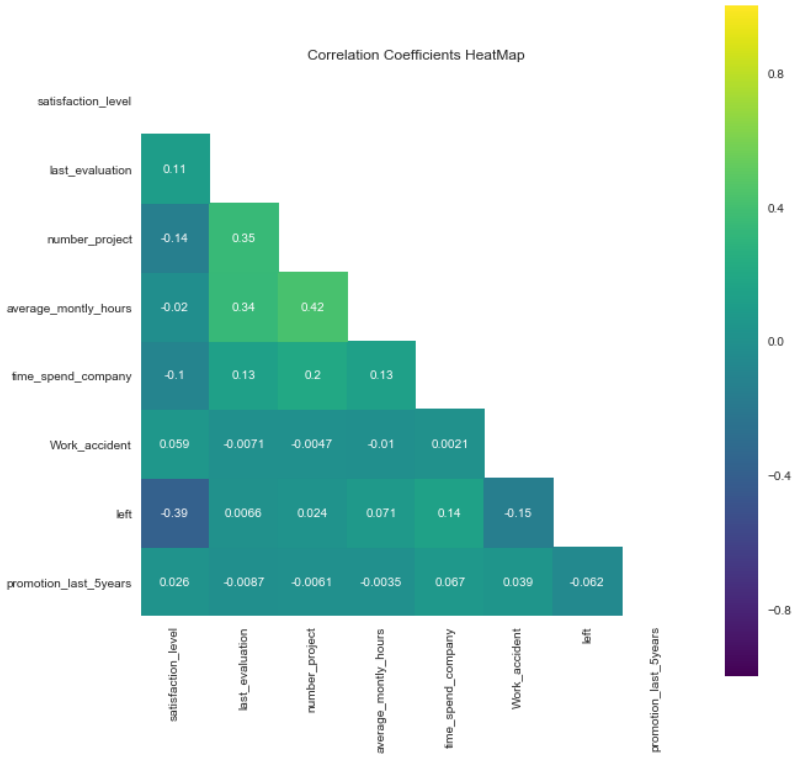


Figure 3

Then proceeding to graph how features relate to churn, it was possible to observe interesting characteristics from the dataset and even point some trends. Below are some of the more interesting finds to keep brevity:

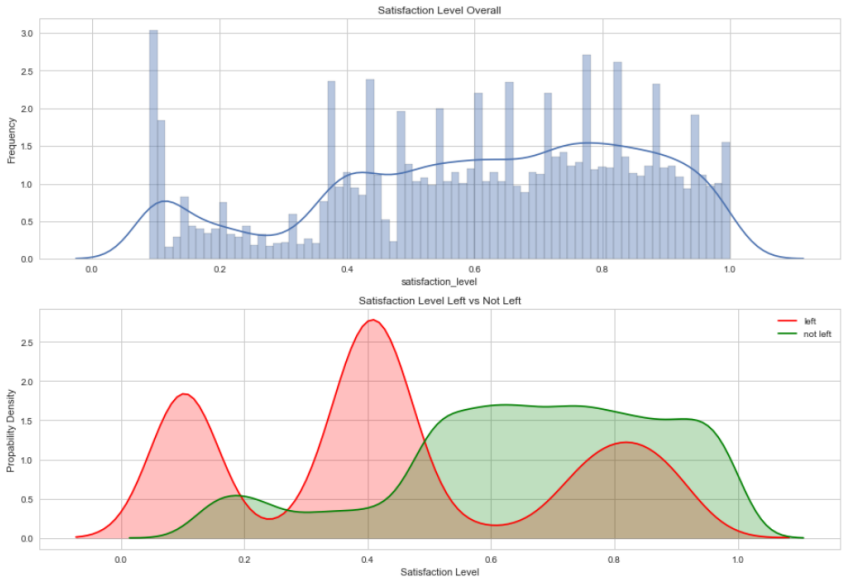


Figure 4

At medium and lower satisfaction levels are the highest rate of churn.

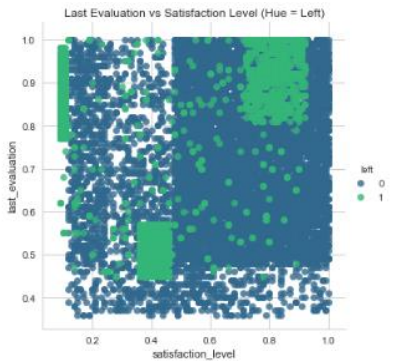


Figure 5

It seems those highly praised in their evaluations and have a greater satisfaction level tend to leave the company. The opposite seems to work the same way, which probably would be contrary to traditional intuition.

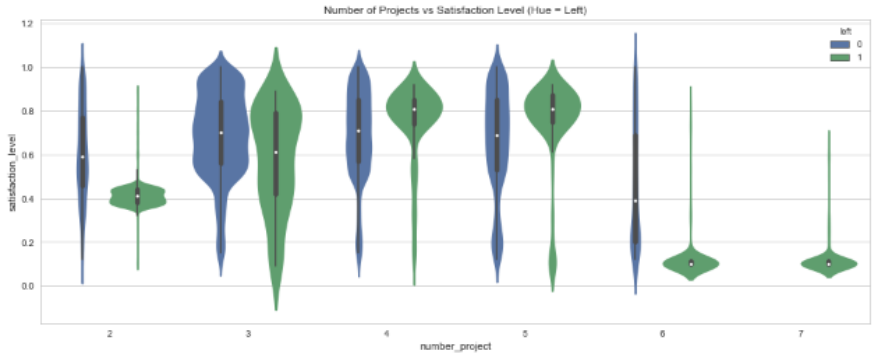


Figure 6

This graph clearly shows that the more projects an employee may have at a company, the less likely it will stay on this job.

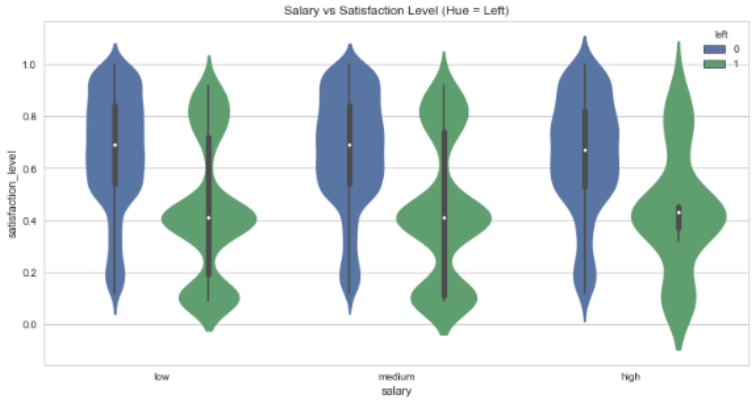


Figure 7

This graph shows that although satisfaction levels are key to employee not leaving, employees will still leave at medium and lower salaries no matter what level of satisfaction they have.

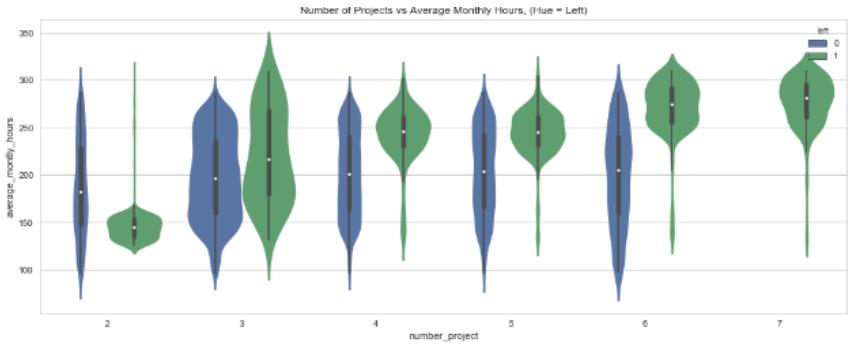


Figure 8

This graph indicates that the higher number of hours worked in conjunction with a higher number of projects is a recipe for churning.

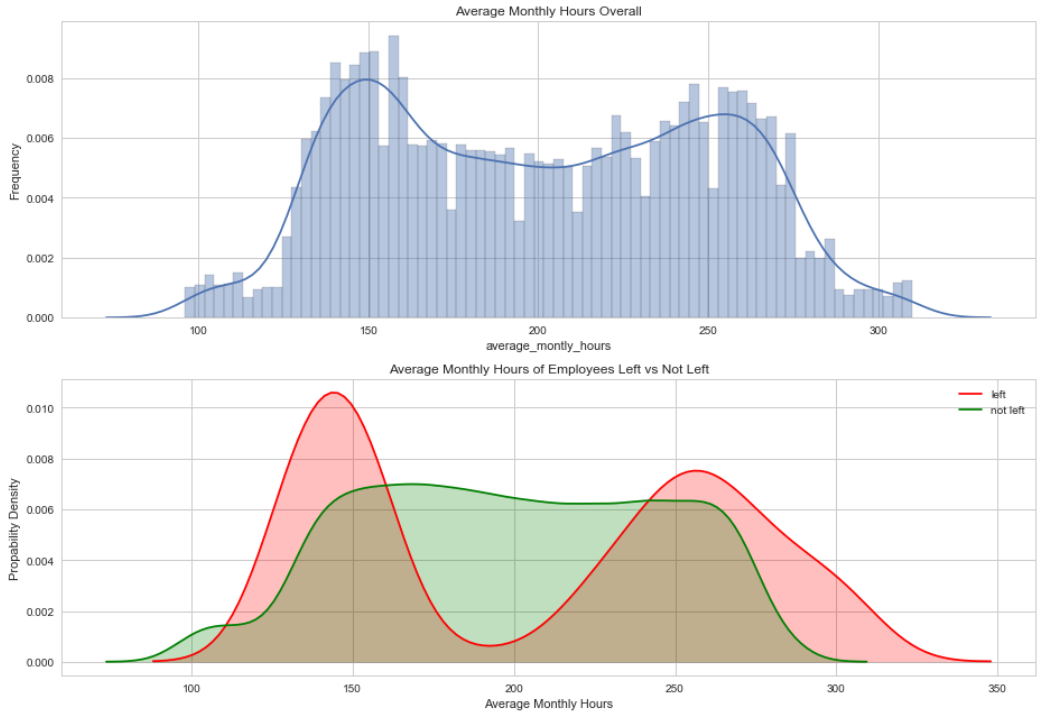


Figure 9

This graph suggests that ideal workers monthly hours should be around 200 as it has most employees staying. with the company with minimum churning.

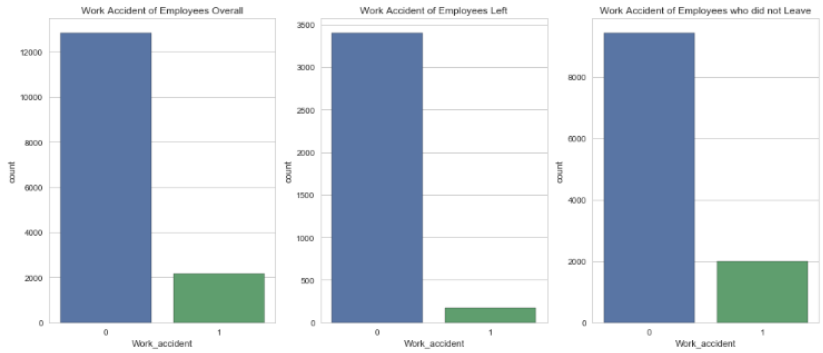


Figure 10

Work accidents don't seem to be that big of factor for churning. Without knowledge of what type of accident this feature might not be relevant in the overall learning process

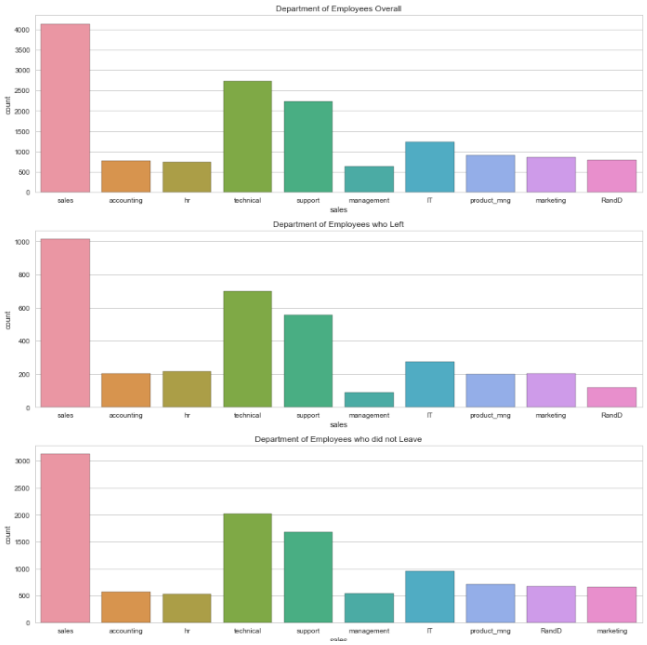


Figure 11

It seems that job position doesn't really affect churning as they seem to stay almost constant.

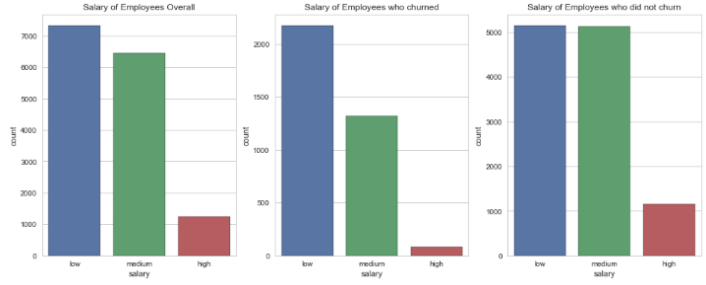


Figure 12

The main point in this graph is that the higher the salary, the better the company will retain their employees based on the salary feature.

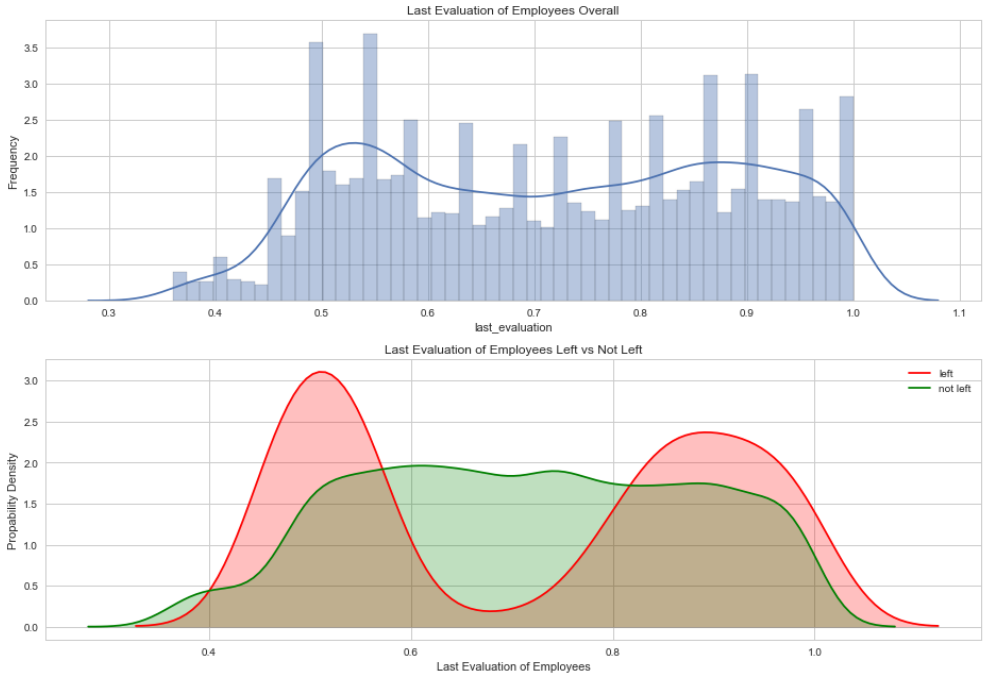


Figure 13

Key takeaway is that having evaluations around 0.7 score seems to be key to hold on to their employees on the last_evaluation feature.

Figure 14

It seems that independent of salary, having a last evaluation of near 0.7 is key to reducing churn.

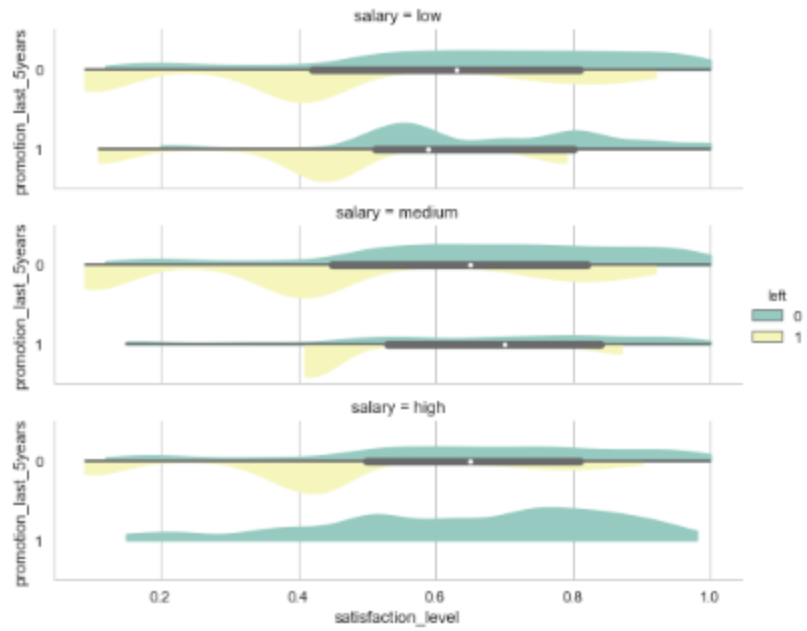


Figure 15

On lower and medium salaries there is still a significant amount of employee churn even at higher satisfaction levels indicating that salary is another possible contender for most significant features.

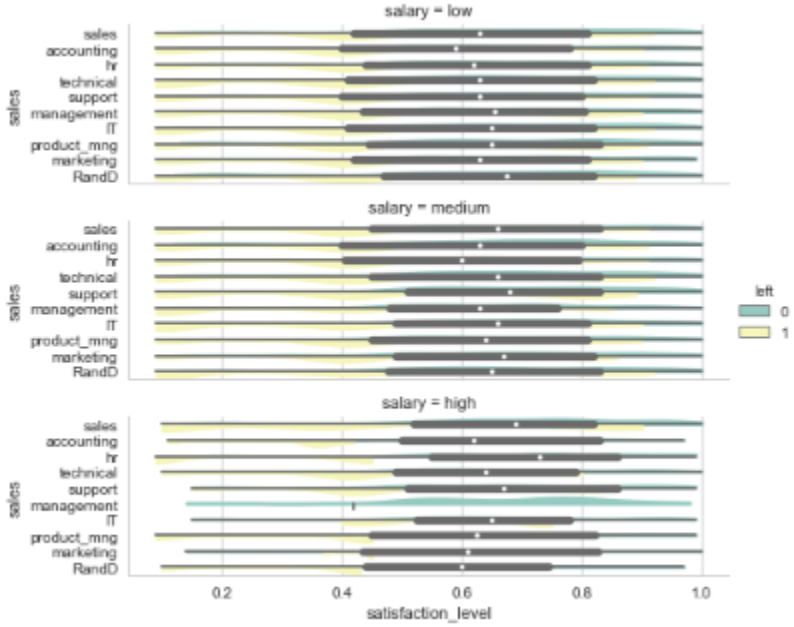


Figure 16

On this factorplot it doesn't really have any significant indicators that really stand out. It just shows that the lower the employee satisfaction the more churn it will have in all job positions.

Algorithms and Techniques

With a company with that many employees and around 15,000 samples on which to train for churning purposes and with all additional personnel that come and go, it is necessary that the chosen algorithm must scale well, in both number of samples and on number of features. 10 features were identified and is a good start on identifying employee churn behavior and will very likely add more features as the understanding of employee behavior improves such as finding new correlations that affect the employee's decision to leave the company. The number of features will never be greater than the number of samples.

The final model was used to predict churning on demand. A quick prediction time is important and will only become more important as the number of employees grows and new policies need to be made. On the other hand, training time is not a significant factor. As new data becomes available, the model can be retrained asynchronously and pushed to live when ready. Online learners would be ideal in this situation, as the true results of each churn are constantly available, and the model could then be updated with the new information.

Based on the above, and by consulting the scikit-learn documentation on choosing an estimator, the following algorithms were trained and tested:

Logistic Regression

Is commonly used to estimate a probability that an instance belongs to a class or feature, in this project's case it refers to churn ('left' feature). If the estimated probability is greater than 50% it assigns a 1. Thus, Logistic Regression is a binary classifier. A good score was not expected without further fine tuning. This is a very basic model that will be used as a benchmark to the other models. (Geron, 2018)

Decision Trees

A decision tree classifier is a worthy candidate for this situation. They are used in data mining as they are performant, scalable, and suited to both complex numerical and categorical data while requiring little data preparation. This model can also be applied in a non-technical environment as decision trees can be easily visualized and explained to non-technical individuals. An employee churn detection system would likely be more effective if it can be intuitively understood by programmers and administrative decision makers for its implementation. (Geron, 2018)

Random Forest

Like Decision Trees, Random Forest introduces extra randomness when growing decision trees and instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This makes it easy to measure the relative importance of each feature deciding which ones really matter. This algorithm is a candidate due to its ease of use, feature picking behavior which will likely outperform the other models. (Geron, 2018)

SVM

It is a very powerful and versatile model that handles performing linear and non-linear classification, regression and even outlier detection. SVM was chosen because it has been particularly well suited for classification of complex but small to medium datasets, so it should be a worthy model candidate as it would also pick up any outliers. (Geron, 2018)

Benchmark

The benchmark model will be the logistic regression model's accuracy, F1-score, precision and recall scores. Given that TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives:

Accuracy is defined as $A(M) = \frac{TN+TP}{TN+FP+FN+TP}$, which measures how often the model predicts correctly.

Recall = $\frac{TP}{TP+FN}$, which states how well the model identifies those who churn.

Precision = $\frac{TP}{TP+FP}$, which states how believable is the model.

F1 Score = $\frac{2*(Precision * Recall)}{Precision + Recall}$, which is the balance between precision and recall.

III. Methodology

Data Preprocessing

For preprocessing the dataset, it was needed to check if there were any null cells by applying the following command and result which indicated a clean set with no empty cells:

>>> df.apply(lambda x: sum(x.isnull()), axis=0)

>>> Output [ ]:

```
satisfaction_level        0
last_evaluation           0
number_project            0
average_montly_hours      0
time_spend_company        0
Work_accident             0
left                      0
promotion_last_5years     0
sales                     0
salary                    0
dtype: int64
```

Another data preprocessing was the use of One-hot encode categorical features which converts a single categorical feature into multiple binary features, where each binary feature represents a possible value of the original categorical feature. This is necessary as not all algorithms work well (or at all) with categorical features. One-hot encoding allows these categorical features to be represented simply as vectors which are much easier to work with. Both salary and sales features were hot encoded.

sales_accounting	sales_hr	sales_management	sales_marketing	sales_product_mng	sales_sales	sales_support	sales_technical
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0

Sales example

$_high	$_medium
0	0
0	1
0	1
0	0
0	0

Salary example1

Implementation

The main algorithm and techniques used were:

Correlation:

```
# Correlation Coefficients Heatmap
correlation = df.corr()
plt.figure(figsize=(10, 10))
mask = np.zeros_like(correlation)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(correlation, vmax=1, square=True, annot=True, cmap='viridis', mask = mask)
plt.title('Correlation Coefficients HeatMap')
```

One Hot Encoding

```
# Appends string prior to one hot encoding
df['salary'] = '$_' + df['salary'].astype(str)
df['sales'] = 'sales_' + df['sales'].astype(str)

# Create 'salary' dummies and join
onehotenc_salary = pd.get_dummies(df['salary'])
df = df.join(onehotenc_salary)

# Create 'sales' dummies and join
onehotenc_sales = pd.get_dummies(df['sales'])
df = df.join(onehotenc_sales)
```

```
# Drop these features to avoid dummy variable trap
df = df.drop(['salary', 'sales', '$_low', 'sales_IT'], axis=1)
display(df.head())
```

Test, Training and Validating Datasets

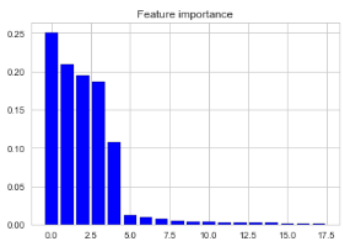
```
# Randomly, split the data into test/training/validation sets
train, test, validate = np.split(df.sample(frac=1), [int(.6*len(df)), int(.8*len(df))])
print (train.shape, test.shape, validate.shape)


# Separate target and predictors
y_train = train['left']
x_train = train.drop(['left'], axis=1)
y_test = test['left']
x_test = test.drop(['left'], axis=1)
y_validate = validate['left']
x_validate = validate.drop(['left'], axis=1)
```

Feature Importance

```
#Plotting the feature importance
importances = rf.feature_importances_
std = np.std([rf.feature_importances_ for tree in rf.estimators_], axis=0)
indices = np.argsort(importances)[::-1]

plt.figure()
plt.title("Feature importance")
plt.bar(range(x_train.shape[1]), importances[indices], color="b", yerr=std[indices], align="center")
```



This graph indicates that only the first 5 features are relevant. Therefore, the other features were dropped.

Decision Tree, Random Forest and SVM model implementation:

```
# DECISION TREE (depth of 3)
tree_model = tree.DecisionTreeClassifier(max_depth=3)
# Fit a decision tree
tree_model = tree_model.fit(x_train, y_train)
# Training accuracy
tree_model.score(x_train, y_train)

# Predictions/probs on the test dataset
predicted = pd.DataFrame(tree_model.predict(x_test))
probs = pd.DataFrame(tree_model.predict_proba(x_test))

# Store metrics
tree_accuracy = metrics.accuracy_score(y_test, predicted)
tree_roc_auc = metrics.roc_auc_score(y_test, probs[1])
tree_confus_matrix = metrics.confusion_matrix(y_test, predicted)
tree_classification_report = metrics.classification_report(y_test, predicted)
tree_precision = metrics.precision_score(y_test, predicted, pos_label=1)
tree_recall = metrics.recall_score(y_test, predicted, pos_label=1)
tree_f1 = metrics.f1_score(y_test, predicted, pos_label=1)

# Evaluate the model using 10-fold cross-validation
tree_cv_scores = cross_val_score(tree.DecisionTreeClassifier(max_depth=3),
                                 x_test, y_test, scoring='precision', cv=10)
tree_cv_mean = np.mean(tree_cv_scores)
```

```
# RAMDOM FOREST
rf = RandomForestClassifier()
# Fit
rf_model = rf.fit(x_train, y_train)
# Training accuracy
rf_model.score(x_train, y_train)

# Predictions/probs on the test dataset
predicted = pd.DataFrame(rf_model.predict(x_test))
probs = pd.DataFrame(rf_model.predict_proba(x_test))

# Store metrics
rf_accuracy = metrics.accuracy_score(y_test, predicted)
rf_roc_auc = metrics.roc_auc_score(y_test, probs[1])
rf_confus_matrix = metrics.confusion_matrix(y_test, predicted)
rf_classification_report = metrics.classification_report(y_test, predicted)
rf_precision = metrics.precision_score(y_test, predicted, pos_label=1)
rf_recall = metrics.recall_score(y_test, predicted, pos_label=1)
rf_f1 = metrics.f1_score(y_test, predicted, pos_label=1)

# Evaluate the model using 10-fold cross-validation
rf_cv_scores = cross_val_score(RandomForestClassifier(), x_test, y_test, scoring='precision', cv=10)
rf_cv_mean = np.mean(rf_cv_scores)
```

```
# SVM
svm_model = SVC(probability=True)
# Fit
svm_model = svm_model.fit(x_train, y_train)
# Accuracy
svm_model.score(x_train, y_train)

# Predictions/probs on the test dataset
predicted = pd.DataFrame(svm_model.predict(x_test))
probs = pd.DataFrame(svm_model.predict_proba(x_test))

# Store metrics
svm_accuracy = metrics.accuracy_score(y_test, predicted)
svm_roc_auc = metrics.roc_auc_score(y_test, probs[1])
svm_confus_matrix = metrics.confusion_matrix(y_test, predicted)
svm_classification_report = metrics.classification_report(y_test, predicted)
svm_precision = metrics.precision_score(y_test, predicted, pos_label=1)
svm_recall = metrics.recall_score(y_test, predicted, pos_label=1)
svm_f1 = metrics.f1_score(y_test, predicted, pos_label=1)

# Evaluate the model using 10-fold cross-validation
svm_cv_scores = cross_val_score(SVC(probability=True), x_test, y_test, scoring='precision', cv=10)
svm_cv_mean = np.mean(svm_cv_scores)
```

And finally, the model comparison:

```
# Model comparison
models = pd.DataFrame({
    ' Model'    : [("Logistic Regression"), ('Decision Tree'), ('Random Forest'), ('SVM')],
    'Accuracy' : [logit_accuracy, tree_accuracy, rf_accuracy, svm_accuracy],
    'Precision': [logit_precision, tree_precision, rf_precision, svm_precision],
    'recall'   : [logit_recall, tree_recall, rf_recall, svm_recall],
    'F1'       : [logit_f1, tree_f1, rf_f1, svm_f1],
    'cv_precision' : [logit_cv_mean, tree_cv_mean, rf_cv_mean, svm_cv_mean]
    })

models.sort_values(by='Precision', ascending=False)
```

One of the challenges in implementation was determining which parameters to use for tuning for each model and how to model it out for comparison. Although Decision Trees and Random Forest models had similar parameters, they differed quite a bit for Logistic Regression and SVM models. The use of GridSearchCV greatly helped in this process of trial and error, but it was time consuming nevertheless.

Refinement

Each algorithm takes several parameters as inputs which can drastically affect the performance of the algorithm. However, the highest-performing combination of parameters cannot be determined at first without being tested and somewhat guessed in a process called parameter tuning.

Each possible combination of parameters is a n-dimensional grid, where n corresponds to the number of parameters being tuned that exhaustively searches for the highest performing parameter values, where the performance of each cell is tested using cross-validation. With the use of GridSearchCV the highest-performing parameter values are reported, along with their score.

For the decision tree and random forest classifiers, the following parameters can be tuned:

- criterion: the metric of information gain used at each step in the decision tree
- max_features: the number of features considered at each step in the tree when evaluating which feature best partitions the data, according the metric defined by the criterion parameter.
- max_depth: the maximum depth of the tuned tree. Decision trees tend to overfit, so deeper is not always better.
- min_samples_split: the minimum number of samples needed to consider adding another split to the tree.
- n_estimators: the number of decision trees to train, the combination of which compose the random forest.
- Setting C: Penalty parameter of the error term. If there is a lot of noisy observations decrease the value. It corresponds to regularize more the estimation.

Below are the original model parameters.

These are the Logistic Regression Unoptimized Parameters:

{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'max_iter': 100, 'multi_class': 'ovr', 'n_jobs': 1, 'penalty': 'l2', 'random_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}

-GridSearchCV didn't suggest any change as the scores didn't change either.

These are the Decision Trees Unoptimized Parameters:
{'class_weight': None, 'criterion': 'gini', 'max_depth': 3, 'max_features': None, max_leaf_nodes': None, 'min_impurity_split': 1e-07, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': None, 'splitter': 'best'}

-GridSearchCV changed {'criterion': 'entropy', 'max_depth': 10}.

These are the Random Forest Unoptimized Parameters:
{'bootstrap': True, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'min_impurity_split': 1e-07, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': 1, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}

-GridSearchCV changed {'bootstrap': False,}

These are the SVM Unoptimized Parameters:
{'C': 1.0, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': None, 'degree': 3, 'gamma': 'auto', 'kernel': 'rbf', 'max_iter': -1, 'probability': True, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}

-GridSearchCV changed {'C': 10}

The original model scores pre-tuning:

	Model	Accuracy	F1	Precision	cv_precision	recall
2	Random Forest	0.987333	0.973050	0.985632	0.987848	0.960784
3	SVM	0.956000	0.909091	0.894309	0.877560	0.924370
1	Decision Tree	0.952333	0.901853	0.884253	0.880133	0.920168
0	Logistic Regression	0.760333	0.345769	0.493506	0.531342	0.266106

And after tuning:

	Model	Accuracy	F1	Precision	cv_precision	recall
2	Random Forest	0.989333	0.977305	0.989943	0.987973	0.964986
1	Decision Tree	0.979667	0.956272	0.979442	0.880133	0.934174
3	SVM	0.965333	0.926864	0.930791	0.877560	0.922969
0	Logistic Regression	0.760333	0.345769	0.493506	0.531342	0.266106

As can be seen, except for the Logistic Regression model, all other models showed up with a great accuracy, recall and precision scores after tuning with a big jump in score for the Decision Tree Model.

The Random Forest algorithm is still the best model, even though it only had a slight increase in score after tuning.

IV. Results

Model Evaluation and Validation

Upon performing a model comparison with all other models, it was clear that Random Forest is the best model based on Accuracy, Precision and Recall metrics. This performance would raise a red flag about overfitting and a lack of generalization to future periods of data. However, the classifier was evaluated with K-folds cross-validation (K=10) and evaluated on the test dataset.

```
# RANDOM FOREST
rf = RandomForestClassifier()
# Fit
rf_model = rf.fit(x_train, y_train)
# Training accuracy
rf_model.score(x_train, y_train)

# Predictions/probs on the test dataset
predicted = pd.DataFrame(rf_model.predict(x_test))
probs = pd.DataFrame(rf_model.predict_proba(x_test))

# Store metrics
rf_accuracy = metrics.accuracy_score(y_test, predicted)
rf_roc_auc = metrics.roc_auc_score(y_test, probs[1])
rf_confus_matrix = metrics.confusion_matrix(y_test, predicted)
rf_classification_report = metrics.classification_report(y_test, predicted)
rf_precision = metrics.precision_score(y_test, predicted, pos_label=1)
rf_recall = metrics.recall_score(y_test, predicted, pos_label=1)
rf_f1 = metrics.f1_score(y_test, predicted, pos_label=1)

# Evaluate the model using 10-fold cross-validation
rf_cv_scores = cross_val_score(RandomForestClassifier(), x_test, y_test,
                               scoring='precision', cv=10)
rf_cv_mean = np.mean(rf_cv_scores)
```

Model	Accuracy	F1	Precision	cv_precision	recall
Random Forest	0.989333	0.977305	0.989943	0.987973	0.964986

Printing out the 10-fold cross validation results shows the model is robust.

```
[ 1.          1.          1.          0.98550725  0.97058824  1.
  0.97014925  0.97014925  0.98333333  1.        ]
```

The final Random Forest model Parameters are:
{'bootstrap': False, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, ‘max_features': 'auto', 'max_leaf_nodes': None, 'min_impurity_split': 1e-07, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': 1, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}

Overall this model generalizes well to unseen data due to its random nature and that it performs well on all evaluated metrics. However, as more and more features are added it would be wise to perform feature importance and remove features not heavy correlated with others in the model. Also, there will always be the issue of employees leaving at random (maybe an employee just doesn't want to work anymore), or for causes not evaluated by the model. But for the current scenario presented, this model should save the company using a lot of headache and hopefully incentivize management to act.

Justification

	Model	Accuracy	F1	Precision	cv_precision	recall
2	Random Forest	0.989333	0.977305	0.989943	0.987973	0.964986
1	Decision Tree	0.979667	0.956272	0.979442	0.880133	0.934174
3	SVM	0.965333	0.926864	0.930791	0.877560	0.922969
0	Logistic Regression	0.760333	0.345769	0.493506	0.531342	0.266106

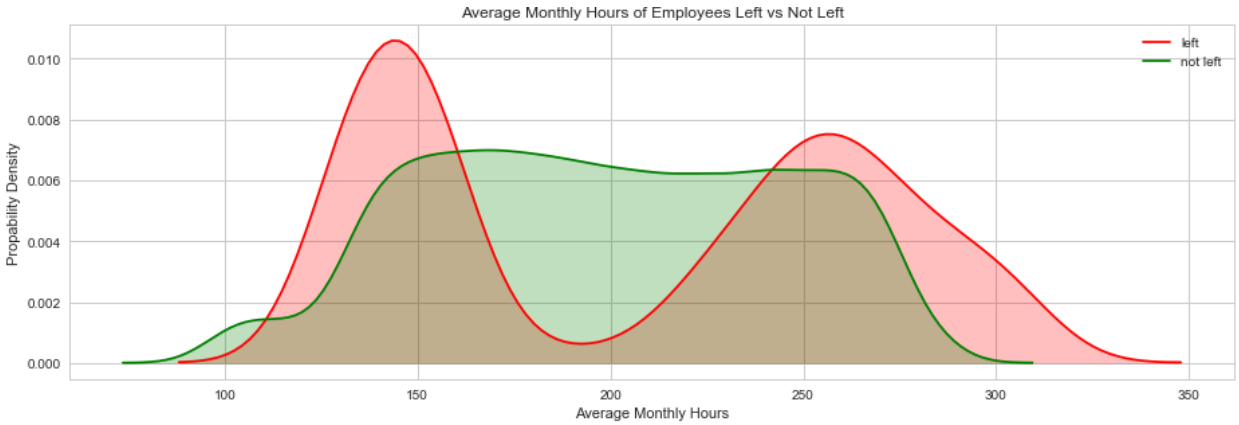
Comparing to the Logistic Regression model which has been used as a benchmark, the Random Forest model performs better on all metrics presented.

With a high cross validation precision, recall and F1 scores, the company will be able to identify with confidence if an employee will churn based on the features given.

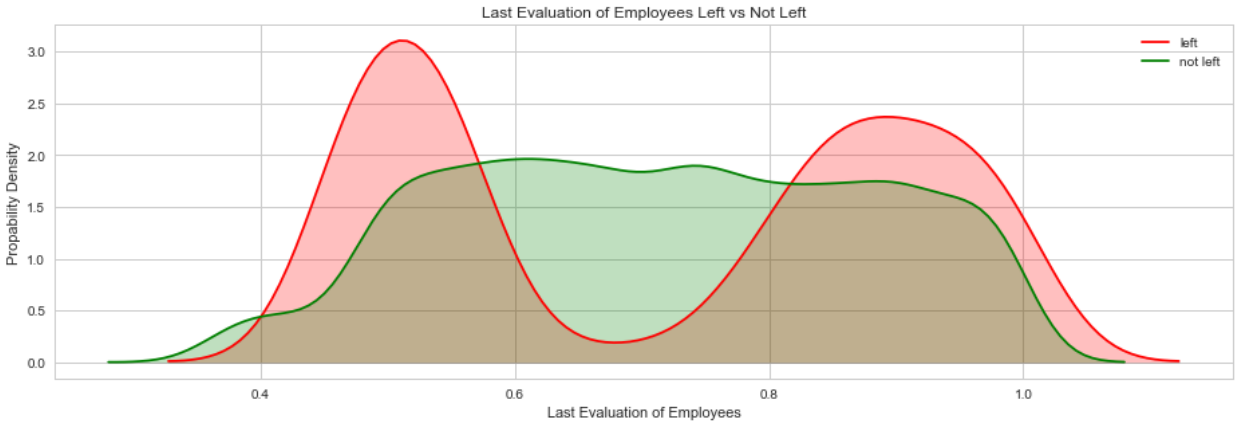
V. Conclusion

Free-Form Visualization

When analyzing graphs just before performing one hot encoding and Feature Importance tuning there were a couple of graphs that had interesting information regarding employee churning:



This graph calls attention to the valley in there which states that if the monthly work hour is set just around 180, the company can both maximize employee retention and reduce churning simultaneously achieving 2 of the company's goal in just one feature change.



Similarly, and albeit contradictory, in this case if the employer gives just a 0.7 evaluation score, it would maximize employee retention and churning reduction. One would expect that the higher the evaluation the less likely to leave the company.

This just opens an analyst's mind of the intricacies hidden between features that would seem logical at first, but in reality, is farther from the truth.

So, it is crucial for HR and Administrative managers to understand these concepts before applying a hiring or firing policy.

Reflection

The main steps taken for this capstone were:

1. Data preparation: identifying potentially relevant features, data types, churn percentage and statistical information retrieving.
2. Data visualization: exploring correlation of the data, both to form a stronger understanding of the relationship between the features and to identify any preprocessing necessary before training.
3. Algorithm selection: considering the details of the problem at hand to choose potential candidate algorithms for training.
4. Benchmarking: Outlining and testing a benchmark against which the candidate algorithms can be measured against.
5. Feature preprocessing: carrying out the preprocessing as identified in the data visualization and exploration stage, including inputting potential missing values and one-hot encoding categorical features.
6. Algorithm parameter tuning: train/test/validate splitting the data, then performance parameter grid searches with cross-validation to identify the highest-performing model of the candidates.

The main challenge for this project was trying to understand what the data meant and how each feature related to each other. Albeit with random forest it might make perfect mathematical sense to remove some features it feels are not pertinent, but it is not always easy to rationalize the root cause of this.

Overall, for a first Capstone project, the model performance is far greater than any initial educated guess, exceeding the author's expectation. This model definitely feels ready to be packaged and put into real-world application production.

In this section, you will summarize the entire end-to-end problem solution and discuss one or two aspects of the project you found interesting or difficult. You are expected to reflect on the project to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

Improvement

The main improvement should be to add more data and more features to further enrich this model. There are certainly more issues that employees face that cause them to churn that aren't fully recorded. In a time where employee *ghosting* (the act of an employee leaving the company/interview/ facility without telling their employer and never coming back) is so common, more companies struggle to understand what is going on.

On a second note, SVM's performance was sub-par than expected when compared to decision trees and random forest even though it is considered one the most powerful

Machine Learning models. Further parameter tuning and adding more data might bring its performance up.

On a third and final note, a model that was put some thought to be used was the XGBoost which is an implementation of gradient boosted decision trees designed for speed and performance, commonly used in applied machine learning and Kaggle competitions for structured or tabular data. (Brownlee, 2016)

References

Afonja, T. (2017, December). *Towards Data Science*. Retrieved from Accuracy Paradox: https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b

Brownlee, J. (2016, August). *Machine Learning Mastery*. Retrieved from A Gentle Introduction to XGBoost for Applied Machine Learning: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Geron, A. (2018). *Hands-On Machine Learning with Scikit-Learn & TensorFlow.* Sebastopol: O'Reilly.

Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python.* Cambridge: The MIT Press.

Ho, T. K. (1995). Random Decision Forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 278-282 vol.1.

Ng, A. (2015). *Stanford University*. Retrieved from CS229: https://cs229.stanford.edu/materials.html

Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Pearson Education.

scikit-learn. (2018). Retrieved from 1.4. Support Vector Machines: http://scikit-learn.org/stable/modules/svm.html#svm-classification

scikit-learn. (2018). Retrieved from 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Walter, W. (2018). *Kaggle*. Retrieved from HR-Analytics: https://www.kaggle.com/colara/hr-analytics