HR Employee Attrition Use Case

Maziyar Hamdi, Feb 04, 2018

Overview

Problem

Use Case: A firm wants to understand why some of best employees are leaving the company. The firm also wants to predict which employees will leave next, why and what should they do to retain them.

Attached is the fictional data set that provides some information on ~1470 employees.

Ask:
1. Identify reasons for churn 2. Models predicting which employees will leave next 3. Recommendation on which employees from point 2. above should be retained?

Output: Model/Code and analysis on the problem.

Method

In order to figure out the reasons for churn, after data-preprocessing and train-validation split, I am going to build a classifier on the data using Attrition feature as the class label. By looking at the most informative features associated with Class representing churn (i.e. Attrition=Yes), we can find the attributes that are possibly the reason for churn.

Then, in order to predict who are going to leave next, I look at the people that have not already left and run my classifier on those candidates. By looking at the output of classifier (probability that person is going to leave) I can identify the employees that are most likely to leave.

Finally, by performing sensitivity analysis on the attributes that Firm has control over (for example MonthlyIncome, Stock, Rate) I pick the ones that company could/should retain by improving those attributes. More specifically, I increase those attributes by a fixed percentage and re-run the classifier to see with the new value for that attribute, are they in "risk zone" (classified as likely to leave) or not. The ones that--if their attributes improve--get out of the "risk zone" are potential candidates for retention.

Setup

You need to have Python 2.7.

The requirements can be installed by running the following in the root directory:

```
pip install -r requirements.txt
```

To install xgboost you need to run these:

```
brew install gcc@5
pip install setuptools
git clone --recursive https://github.com/dmlc/xgboost
cd xgboost
./build.sh
cd python-package; python setup.py install
```

Data Overview

Data sample includes 1470 instances and for each instance we get 34 attributes.

Attributes include a class label, namely Attrition, integer valued, ordinal and categorical features. Also there are missing values in some of the columns. I used Data Framework in order to read and manipulate the csv values.

```
import pandas as pd
df = pd.read_csv('tw.csv')
df.isnull().sum()
```

Output:

```
Age                       0
BusinessTravel            3
Department                0
DistanceFromHome          0
Education                 0
EducationField            0
EmployeeCount             0
EmployeeNumber            0
EnvironmentSatisfaction   4
Gender                    4
HourlyRate                0
JobInvolvement            0
JobLevel                  0
JobRole                   0
JobSatisfaction           0
MaritalStatus             0
MonthlyIncome             0
MonthlyRate               0
NumCompaniesWorked        0
Over18                    0
OverTime                  0
PercentSalaryHike         0
PerformanceRating         0
RelationshipSatisfaction  0
StandardHours             0
```

```
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
Attrition                   0
dtype: int64
```

To get a better sense of information carried in each attribute, it's good to take a look at mean and variance of integer valued attributes.

```
df.var()
```

Output:

```
Age                         8.345505e+01
DistanceFromHome            6.572125e+01
Education                   1.048914e+00
EmployeeCount               0.000000e+00
EmployeeNumber              3.624333e+05
EnvironmentSatisfaction     1.192874e+00
HourlyRate                  4.132856e+02
JobInvolvement              5.063193e-01
JobLevel                    1.225316e+00
JobSatisfaction             1.216270e+00
MonthlyIncome               2.216486e+07
MonthlyRate                 5.066288e+07
NumCompaniesWorked          6.240049e+00
PercentSalaryHike           1.339514e+01
PerformanceRating           1.301936e-01
RelationshipSatisfaction    1.169013e+00
StandardHours               0.000000e+00
StockOptionLevel            7.260346e-01
TotalWorkingYears           6.054056e+01
TrainingTimesLastYear       1.662219e+00
WorkLifeBalance             4.991081e-01
YearsAtCompany              3.753431e+01
YearsInCurrentRole          1.312712e+01
YearsSinceLastPromotion     1.038406e+01
YearsWithCurrManager        1.273160e+01
dtype: float64
```

```
df.mean()
```

Output:

```
Age                            36.923810
```

```
DistanceFromHome               9.192517
Education                      2.912925
EmployeeCount                  1.000000
EmployeeNumber              1024.865306
EnvironmentSatisfaction        2.723056
HourlyRate                    65.891156
JobInvolvement                 2.729932
JobLevel                       2.063946
JobSatisfaction                2.728571
MonthlyIncome               6502.931293
MonthlyRate                14313.103401
NumCompaniesWorked             2.693197
PercentSalaryHike             15.209524
PerformanceRating              3.153741
RelationshipSatisfaction       2.712245
StandardHours                 80.000000
StockOptionLevel               0.793878
TotalWorkingYears             11.279592
TrainingTimesLastYear          2.799320
WorkLifeBalance                2.761224
YearsAtCompany                 7.008163
YearsInCurrentRole             4.229252
YearsSinceLastPromotion        2.187755
YearsWithCurrManager           4.123129
dtype: float64
```

We can easily see that there are some attributes with very low variance, therefore we need to perfrome a feature selection as well.

Preprocessing

I am going to use scikit-learn which is developed for working with NumPy arrays and pandas' DataFrame for machine learning tasks. Now that I get a better sense of data, I want to descibe the preprocessing tactics that are used in order to prepare data for classification / prediction problem.

1. Handling Ordinal attributes
2. Handling Categorical attributes
3. Handling missing values
4. Normalizing features

Handling Ordinal attributes

The only ordinal attribute in the dataset that has information (high variance) is 'BusinessTravel', Non-Travel < Travel_Rarely < Travel_Frequently.

Therefore, I use a feature-mapping as follows to deal with this ordinal value:

```
travel_mapping = {'Non-Travel':0, 'Travel_Rarely':1, 'Travel_Frequently':2}
df['BusinessTravel'] = df['BusinessTravel'].map(travel_mapping)
```

Handling Categorical attributes

To deal with categorical attributes, I use one-hot encoding techinque, that is, I determine all possible values for a categorical attribute and create one seperate feature for each of them. For example, if Attribute X has 3 possible values: a,b, c. I create three binary columns instead: X_a, X_b, X_c. This can be done easily using Pandas Framework

```
df_cat = pd.get_dummies(df)
```

Here is the output which now has 56 attributes:

```
df_cat.columns
Out:
Index([u'Age', u'DistanceFromHome', u'Education', u'EmployeeCount',
       u'EmployeeNumber', u'EnvironmentSatisfaction', u'HourlyRate',
       u'JobInvolvement', u'JobLevel', u'JobSatisfaction', u'MonthlyIncome',
       u'MonthlyRate', u'NumCompaniesWorked', u'PercentSalaryHike',
       u'PerformanceRating', u'RelationshipSatisfaction', u'StandardHours',
       u'StockOptionLevel', u'TotalWorkingYears', u'TrainingTimesLastYear',
       u'WorkLifeBalance', u'YearsAtCompany', u'YearsInCurrentRole',
       u'YearsSinceLastPromotion', u'YearsWithCurrManager',
       u'BusinessTravel_Non-Travel', u'BusinessTravel_Travel_Frequently',
       u'BusinessTravel_Travel_Rarely', u'Department_Human Resources',
       u'Department_Research & Development', u'Department_Sales',
       u'EducationField_Human Resources', u'EducationField_Life Sciences',
       u'EducationField_Marketing', u'EducationField_Medical',
       u'EducationField_Other', u'EducationField_Technical Degree',
       u'Gender_Female', u'Gender_Male', u'JobRole_Healthcare Representative',
       u'JobRole_Human Resources', u'JobRole_Laboratory Technician',
       u'JobRole_Manager', u'JobRole_Manufacturing Director',
       u'JobRole_Research Director', u'JobRole_Research Scientist',
       u'JobRole_Sales Executive', u'JobRole_Sales Representative',
       u'MaritalStatus_Divorced', u'MaritalStatus_Married',
       u'MaritalStatus_Single', u'Over18_Y', u'OverTime_No', u'OverTime_Yes',
       u'Attrition_No', u'Attrition_Yes'],
      dtype='object')
```

Handling missing values

The easiest way to handle missing values is to drop rows that contain missing values, but there is a disadvantage of loosing valueable information. Here I am using interpolation technique in order to prepare data for classification. Since the data has lots of boolean (binary) features I am going to replace missing values with median of that column.

```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values='NaN', strategy='median', axis=0)
imputed_data = imputer.fit_transform(df_cat)
```

Normalizing features

Finally, I am using a feature scaling method in order to make sure all the features have the same energy and reduce the risk of over-fitting. This can be done by normalizing each column by range of that column as follwos:

```
from sklearn.preprocessing import Imputer, MinMaxScaler
mms = MinMaxScaler()
normalized_data = mms.fit_transform(imputed_data)
df_norm = pd.DataFrame(imputed_data, columns=df_cat.columns)
```

Training and Validating Model

I used sklearn pipelines in order to perform the feature selection and the training model. I use a ensemble-learning like algorithm and use a set of weak classifiers, each of whoch produces a probability. Then, using an averaging function -- which can be tuned by exhaustive search and some metric like precision-recall -- I compute the final probablity that an instance belongs to a specific class or not.

The first thing to do is to split the data into train and validation set.

```
# training, validation split
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size = .1,
                                                  random_state=12)
```

Since here the training data is imbalanced, and labels are not 100 percent accurate (basically if an employee has not left it does not gurantee that he/she does not belong to our desired class), I over sampled the rare class:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=12, ratio = 1)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

Here is the list of classifiers that I used:

```
def pipelines(X, y, n_feat=30):
    pipelines = [Pipeline([
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier', LogisticRegression(penalty='l2') )
        ])]
    pipelines.append(Pipeline([
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier', MultinomialNB(alpha=1, fit_prior=True))]))

    pipelines.append(Pipeline([
```

```
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier', RandomForestClassifier(n_estimators=100, n_jobs=3))]))

    pipelines.append(Pipeline([
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier',  SVC(probability=True, random_state=1, kernel='rbf'))]))

    pipelines.append(Pipeline([
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier',  SVC(probability=True, random_state=1, kernel='linear'))]))

    pipelines.append(Pipeline([
        ('selector', SelectKBest(chi2, k=n_feat)),
        ('classifier', XGBClassifier())]))

    pipelines.append(Pipeline([
        ('selector', SelectKBest(chi2, k=10)),
        ('classifier', KNeighborsClassifier())]))

    return [pip.fit(X, y) for pip in pipelines]
```

Because of the correlation of features (as a result of handling categorical data, and original correlation in the data) and low-variance features, I used Chi^2 feature selection method with the parameter tuned by exhaustive search. I also used Linear SVC and simple logistic regression for the sake of feature selection (in order to answer the question 1 since their output is easier to interprete). I used KNN in order to handle outliers.

For model validation I used cross-validaton score and also I used the precision-recall-f1 score on the validations test

```
classifiers = pipelines(X_train_res, y_train_res)

# model validation
for clf in classifiers:
    y_t = clf.predict(X_val)
    print precision_recall_fscore_support(y_val, y_t)
    print extra_cv(clf, X_train, y_train)
```

I used these metrics for the parameter tuning and also to find the optimal weights using exhaustive research. Output of Validation tests:

```
Classifier: LogisticRegression
Performance of Validation Test:
(array([0.92708333, 0.37254902]), array([0.73553719, 0.73076923]), array([0.8202765
, 0.49350649]), array([121,  26]))
Cross Validation Score: 0.869957798159

Classifier: MultinomialNB
Performance of Validation Test:
```

```
(array([0.91208791, 0.32142857]), array([0.68595041, 0.69230769]),
array([0.78301887, 0.43902439]), array([121,  26]))
Cross Validation Score: 0.847298453053

Classifier: RandomForestClassifier
Performance of Validation Test:
(array([0.87591241, 0.9       ]), array([0.99173554, 0.34615385]),
array([0.93023256, 0.5       ]), array([121,  26]))
Cross Validation Score: 0.858626994896

Classifier: SVC
Performance of Validation Test:
(array([0.92156863, 0.4       ]), array([0.7768595 , 0.69230769]),
array([0.84304933, 0.50704225]), array([121,  26]))
Cross Validation Score: 0.840519888595

Classifier: SVC
Performance of Validation Test:
(array([0.91919192, 0.375     ]), array([0.75206612, 0.69230769]),
array([0.82727273, 0.48648649]), array([121,  26]))
Cross Validation Score: 0.859368740712
```

Here is the simple weighted average function that has been used to combine score of different algorithms

```
weights = [
1, # LogisticRegressionCV
1, # MultinmialNB
5, # RandomForest
5, # SVC
0, # SVC linear
8, # XGB
5  # KNN
]

def scores(classifiers, X):
    y_proba = zip(*[map(lambda x:x[0], pipeline.predict_proba(X).tolist())
          for pipeline in classifiers])

    return [list(scores) for scores in y_proba]

def prediction(scores):
    return np.inner(weights, scores) / np.sum(weights)
```

Informative features

I used the output of classifiers to intereprete the significance of the features for different classifiers. Here is the list of informative features for each classifiers.

```
CLassifier: LogisticRegression
---  YearsInCurrentRole 2.878
---  YearsWithCurrManager 1.875
```

```
---  JobRole_Research Director 1.612
---  JobInvolvement 1.530
---  JobSatisfaction 1.241
---  EnvironmentSatisfaction 1.217
---  Age 1.149
---  JobRole_Manager 0.885
---  JobRole_Healthcare Representative 0.820
---  Department_Research & Development 0.602
---  PerformanceRating 0.419
---  JobRole_Manufacturing Director 0.386
---  JobLevel 0.333
---  TotalWorkingYears 0.310
---  MaritalStatus_Divorced 0.236
---  OverTime_No 0.213
---  Department_Sales 0.149
---  StockOptionLevel 0.030
---  JobRole_Sales Representative -0.092
---  EducationField_Medical -0.120
---  MonthlyIncome -0.205
---  MaritalStatus_Married -0.490
---  EducationField_Marketing -0.521
---  JobRole_Laboratory Technician -0.822
---  EducationField_Technical Degree -0.973
---  MaritalStatus_Single -1.102
---  YearsAtCompany -1.522
---  OverTime_Yes -1.571
---  BusinessTravel -1.601
---  YearsSinceLastPromotion -2.257

CLassifier: MultinomialNB
---  OverTime_Yes 0.019
---  MaritalStatus_Single 0.011
---  JobRole_Research Director 0.009
---  OverTime_No 0.006
---  JobRole_Sales Representative 0.006
---  JobRole_Manager 0.005
---  JobRole_Healthcare Representative 0.005
---  JobRole_Manufacturing Director 0.005
---  JobRole_Laboratory Technician 0.005
---  JobLevel 0.004
---  Department_Sales 0.004
---  MaritalStatus_Divorced 0.003
---  BusinessTravel 0.003
---  EducationField_Marketing 0.003
---  EducationField_Technical Degree 0.002
---  YearsInCurrentRole 0.002
---  YearsWithCurrManager 0.002
---  MonthlyIncome 0.002
---  TotalWorkingYears 0.001
---  StockOptionLevel 0.001
---  YearsAtCompany 0.001
---  Age 0.000
---  MaritalStatus_Married 0.000
---  PerformanceRating 0.000
---  EducationField_Medical 0.000
---  YearsSinceLastPromotion 0.000
---  JobSatisfaction 0.000
```

```
---  EnvironmentSatisfaction 0.000
---  Department_Research & Development 0.000
---  JobInvolvement 0.000

CLassifier: RandomForestClassifier
---  JobSatisfaction 0.092
---  StockOptionLevel 0.063
---  Age 0.062
---  YearsInCurrentRole 0.062
---  OverTime_Yes 0.056
---  TotalWorkingYears 0.055
---  OverTime_No 0.055
---  MonthlyIncome 0.054
---  EnvironmentSatisfaction 0.053
---  YearsWithCurrManager 0.052
---  JobInvolvement 0.052
---  JobLevel 0.050
---  YearsAtCompany 0.050
---  BusinessTravel 0.048
---  YearsSinceLastPromotion 0.040
---  MaritalStatus_Single 0.022
---  EducationField_Medical 0.017
---  MaritalStatus_Married 0.014
---  JobRole_Laboratory Technician 0.012
---  Department_Research & Development 0.012
---  Department_Sales 0.011
---  EducationField_Technical Degree 0.011
---  PerformanceRating 0.010
---  MaritalStatus_Divorced 0.010
---  EducationField_Marketing 0.008
---  JobRole_Healthcare Representative 0.008
---  JobRole_Manufacturing Director 0.008
---  JobRole_Sales Representative 0.006
---  JobRole_Manager 0.004
---  JobRole_Research Director 0.003

CLassifier: SVC
---  YearsInCurrentRole 2.654
---  JobRole_Research Director 1.357
---  YearsWithCurrManager 1.272
---  JobInvolvement 1.115
---  EnvironmentSatisfaction 1.030
---  JobSatisfaction 1.010
---  Age 0.968
---  JobRole_Healthcare Representative 0.857
---  OverTime_No 0.771
---  JobRole_Manager 0.685
---  Department_Research & Development 0.503
---  PerformanceRating 0.498
---  MaritalStatus_Divorced 0.486
---  JobRole_Manufacturing Director 0.458
---  JobLevel 0.293
---  Department_Sales 0.169
---  StockOptionLevel 0.112
---  TotalWorkingYears 0.078
---  JobRole_Sales Representative 0.005
---  MaritalStatus_Married -0.013
```

```
---  MonthlyIncome -0.094
---  EducationField_Medical -0.154
---  EducationField_Marketing -0.173
---  MaritalStatus_Single -0.473
---  JobRole_Laboratory Technician -0.621
---  EducationField_Technical Degree -0.733
---  OverTime_Yes -0.771
---  YearsAtCompany -0.907
---  BusinessTravel -1.263
---  YearsSinceLastPromotion -1.866

CLassifier: XGBClassifier
---  BusinessTravel 0.089
---  JobInvolvement 0.083
---  EnvironmentSatisfaction 0.079
---  JobSatisfaction 0.076
---  YearsWithCurrManager 0.071
---  StockOptionLevel 0.068
---  Age 0.059
---  YearsSinceLastPromotion 0.059
---  MonthlyIncome 0.054
---  YearsInCurrentRole 0.052
---  OverTime_No 0.043
---  JobLevel 0.041
---  TotalWorkingYears 0.032
---  YearsAtCompany 0.024
---  EducationField_Medical 0.019
---  JobRole_Laboratory Technician 0.019
---  MaritalStatus_Single 0.017
---  Department_Research & Development 0.016
---  EducationField_Technical Degree 0.016
---  MaritalStatus_Divorced 0.013
---  EducationField_Marketing 0.013
---  PerformanceRating 0.013
---  JobRole_Research Director 0.010
---  JobRole_Healthcare Representative 0.010
---  Department_Sales 0.008
---  JobRole_Manager 0.005
---  MaritalStatus_Married 0.005
---  JobRole_Manufacturing Director 0.005
---  JobRole_Sales Representative 0.003
---  OverTime_Yes 0.000
```

Looking at the informative features, here is the list of features that are correlated to Attrition==Yes class

```
---  BusinessTravel
---  JobInvolvement
---  EnvironmentSatisfaction
---  JobSatisfaction
---  YearsWithCurrManager
---  StockOptionLevel
---  Age
---  YearsSinceLastPromotion
---  MonthlyIncome
```

```
---  YearsInCurrentRole
---  MonthlyIncome
---  OverTime_Yes
```

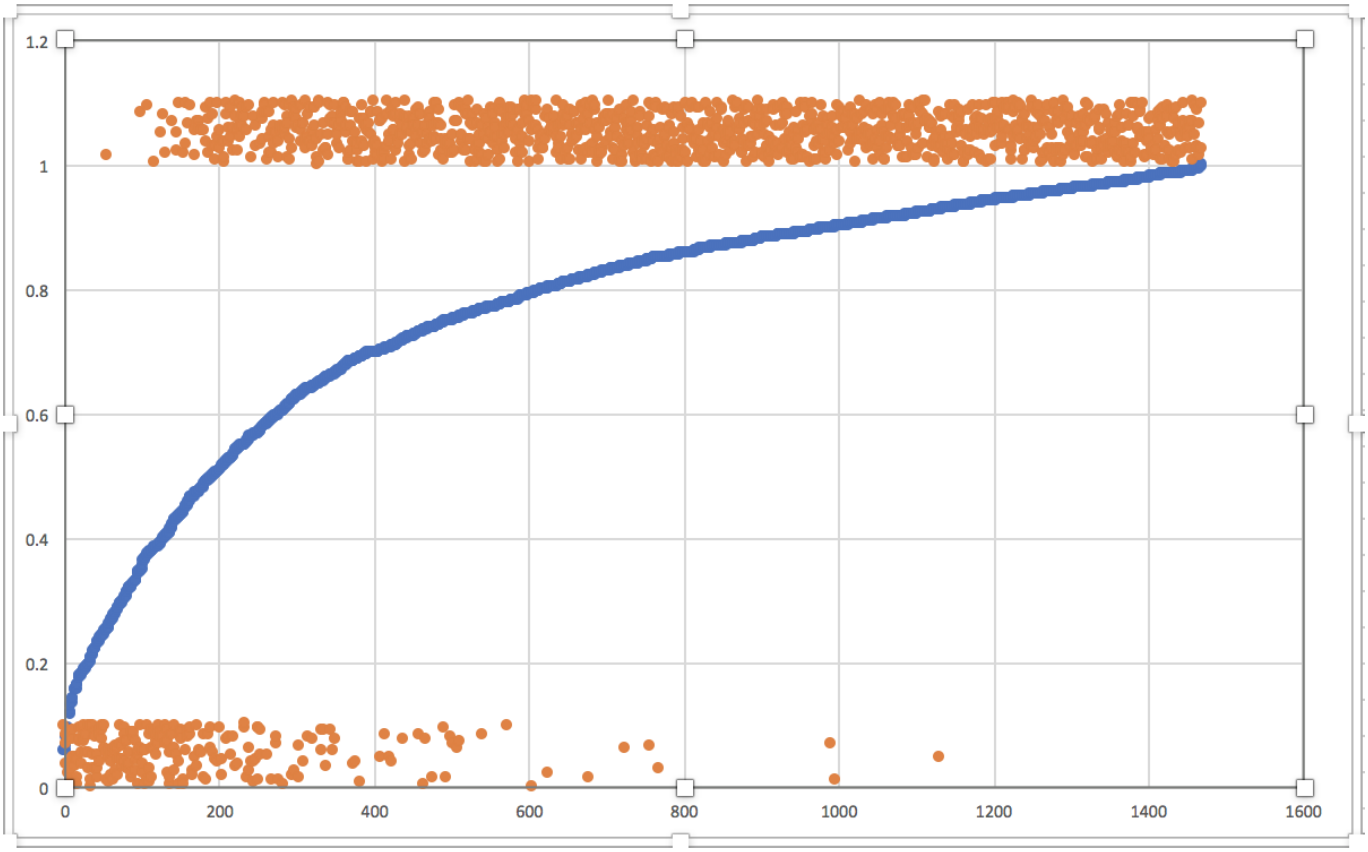
We can summarize this as General Satisfaction (Job Satisfaction, Travel Required for Job, OverTime) and Compensation (Monthly Income, StockOptionLevel) are the two main reasons for churns.

Prediction

Now I passed all the input to the trained model and added another column to the data frame, called 'Probability'

```
raw_scores = scores(classifiers, X)
classifier_output = map(aggregate, raw_scores)
df = df.assign(Probability=pd.Series(classifier_output).values)
df = df.sort_values(by='Probability')
df.to_csv('output.csv')
```

Here is the probability and the actual class (1=Attrion No, 0=Attrition Yes) plotted vs index



By looking into probabilities and the classes, we can figure out that probability greater that 0.8 is associated with Attrition=No class and probabilities less than 0.4 is associated with class=Yes. In between are people that are likely to leave.

```
pred_df = df[df['Attrition']=='No']
```

```
indexes = pred_df[pred_df['Probability']<0.8].index
indexes.tolist()
```

Here is the index of employees that are most probabl to leave sorted based on the likelihood of leaving:

```
[1436, 369, 1011, 652, 1122, 1196, 917, 673, 827, 961, 1142, 921, 318, 665, 397,
763, 1087, 149, 1137, 1180, 301, 48, 1102, 815, 634, 1168, 170, 494, 691, 320, 601,
747, 1391, 546, 889, 924, 416, 347, 612, 658, 91, 536, 1215, 57, 1070, 1402, 496,
350, 157, 1349, 909, 450, 564, 686, 853, 995, 1426, 632, 152, 1450, 38, 11, 164,
859, 1197, 1082, 1343, 1422, 1345, 856, 830, 294, 1084, 620, 986, 1308, 948, 1317,
319, 1128, 912, 202, 54, 284, 929, 1001, 142, 991, 1097, 1048, 839, 989, 930, 964,
3, 1095, 1464, 253, 1017, 734, 996, 685, 172, 703, 262, 876, 437, 1328, 505, 571,
402, 824, 1178, 854, 430, 697, 86, 543, 974, 1386, 1459, 1192, 335, 764, 1400, 1193,
1086, 203, 951, 870, 631, 1435, 1310, 881, 807, 483, 719, 1238, 1120, 880, 1306, 19,
1062, 389, 1460, 1261, 1229, 196, 1172, 1285, 328, 138, 1145, 17, 633, 887, 283,
427, 618, 1188, 1005, 1144, 1340, 845, 825, 291, 1413, 362, 1368, 101, 1244, 1108,
237, 1270, 63, 109, 841, 901, 486, 70, 72, 553, 1319, 670, 1037, 1382, 1028, 895,
346, 1364, 312, 759, 120, 433, 1004, 16, 310, 569, 933, 1165, 23, 1126, 290, 1066,
309, 337, 481, 1245, 1433, 758, 754, 559, 476, 1321, 1403, 655, 263, 256, 717, 314,
487, 624, 195, 672, 1467, 862, 659, 354, 671, 1378, 781, 381, 1067, 1041, 1449, 60,
512, 6, 1094, 1003, 1302, 925, 470, 76, 786, 180, 461, 94, 675, 490, 1329, 46, 441,
66, 143, 4, 128, 331, 835, 1018, 1121, 642, 200, 577, 1334, 1325, 493, 175, 550,
579, 638, 474, 1456, 532, 207, 745, 1347, 460, 644, 603, 32, 260, 1230, 226, 168,
1025, 47, 533, 1406, 212, 80, 1363, 1256, 885, 265, 451, 1342, 1344, 651, 508, 819,
500, 1311, 606, 1050, 130, 897, 1218, 139, 478, 1359, 998, 61, 292, 1034, 383, 1123,
926, 1109, 146, 1207, 811, 793, 221, 277, 1100, 1115, 206, 1226, 274, 79, 1000, 684,
154, 868, 578, 565, 30, 1274, 498, 511, 886, 1224, 52, 742, 1428, 53, 1118, 1389,
753, 818, 1309, 406, 475, 491, 906, 68, 1337, 523, 900, 1074, 1241, 1455, 563, 722,
1069, 1417, 768, 220, 43, 625, 131, 941, 1220, 431, 1022, 1143, 445, 987, 1384, 518,
501, 282, 1454, 718, 311, 374, 55, 423, 270, 148, 125, 1096, 1151, 1419]
```

Candidates for Retaining

In order to figure out which employees Firm can retain, I start by picking few attributes that Firm has control over:

1. Amount of BusinessTravel
2. MonthlyRate
3. MonthyIncome
4. StockOptionLevel
5. Promotion
6. OverTime

I assume that Firm is willing to increase stock option level (after normalization) by 50% MonthlyIncome by 10%, NomthlyRate by 10%, to decrease travel required (after normalization) by 20%, and to remove all over time obligations. Any of these can be commented out (if Firm cannot accomodate). So I create a new data set with these changes to see if Firm offers those, the likelihood of leaving decrease or not.

```
rows = X.values.tolist()
selected_rows = []
for ind in indexes.tolist():
    selected_rows.append(rows[ind])

X_new = pd.DataFrame(selected_rows, columns=X.columns)

X_new['BusinessTravel'] = X_new['BusinessTravel']*.8
X_new['MonthlyRate'] = X_new['MonthlyRate']*1.1
X_new['StockOptionLevel'] = X_new['StockOptionLevel']*1.5
X_new['MonthlyIncome'] = X_new['MonthlyIncome']*1.1
X_new['YearsSinceLastPromotion'] = 0
X_new['OverTime_No'] = 1.0
X_new['OverTime_Yes'] = 0.0
```

Now I run the classifier on the new "imporved" attributes of employees that are in risk zone. The goal is to find out, with the new improvemnets, which employees are willing to stay, i.e. the score is greater than 0.8.

```
raw_scores_new = scores(classifiers, X_new)
classifier_output_new = map(aggregate, raw_scores_new)
df_with_prob_new = X_new.assign(Probability=pd.Series(classifier_output_new).values)
df_with_prob_new =
df_with_prob_new.assign(OriginalIndex=pd.Series(indexes.tolist()).values)
df_with_prob_new = df_with_prob_new.sort_values(by='Probability', ascending=False)
df_with_prob_new.to_csv('output_new.csv')


new_indexes =
df_with_prob_new[df_with_prob_new['Probability']>0.8]['OriginalIndex'].tolist()
print new_indexes
```

Output sorted based on score:

```
[206, 597, 1096, 60, 577, 80, 655, 139, 625, 314, 312, 270, 1467, 1025, 406, 16,
331, 825, 1126, 1382, 237, 1344, 256, 79, 758, 651, 1145, 1074, 486, 718, 1359, 17,
343, 886, 52, 277, 1229, 1413, 1433, 474, 354, 925, 500, 1067, 511, 175, 559, 644,
523, 203, 481, 991, 164, 451, 1345, 1123, 1095, 926, 1342, 350, 989, 553, 996, 1192,
948, 1435, 1028, 19, 807, 870, 1378, 335, 180, 309, 912, 195, 1459, 346, 612, 505,
1422, 53, 6, 631, 283, 854, 602, 1037, 924, 1261, 423, 1340, 986, 719, 882, 1188,
1218, 895, 1450, 263, 624, 142, 768, 319, 1084, 1384, 157, 1462, 1402, 320, 868,
634, 389, 1368, 1034, 687, 1137, 830, 703, 433, 1196, 450, 1220]
```

In order to see how we can improve each of these candidates we can run the algorithm with only one line selected, for example to find which employees can be retained by increasing monthly income, we just need to increase 'MonthlyIncome' and then re-run the model, the output denotes the candidates that can be retained by improving the compensation:\

```
[1384, 1200, 179, 1022, 718, 423, 113, 374, 282, 1417, 888, 684, 987, 625, 148,
```

```
 1260, 1096, 1454, 1118, 406, 125, 117, 1274, 53, 43, 220, 722]
```

Future Works

1. I could divide features into different groups (compensation based, location based, satisfaction based, personal based, etc) and train different models for each of them. This helps us to improve the quality of prediction of which employees could be retained and how.
2. There is room for improvements in model selection, parameter tuning. I tried to be in the 2-4 hours scope.
3. More complex models such as NNs can be used.
4. More advanced aggregation algorithm can be used, e.g., the output of weak classifiers can be passed to another classifier to learn the optimal aggregation schema using the labelled set.