CompSci 671 In-Class Kaggle Competition Report

Yutong Shao
UID: Claireyt

December 9, 2022
Kaggle Competition Link

1 Exploratory Data Analysis

1.1 Data Description

This training data set contains 34 features describing a worker's attributes, and one label concerning attrition status of this worker. We were to train models on training set and predict the attrition status of test set.

To check these statistical properties, I first drew a table to check whether the data needs preprocessing, and observe each feature's statistical properties. It turns out that this data set is clean and tidy - no missing values, data type of each columns is consistent and well-defined. So, it saves me from many data-cleaning work - I only need to encode categorical features to numerical variables, then do feature engineering, which will be mentioned later.

1.2 Data visualization

However, the methods above are not enough to get the most out of the data. I still need to do some visualization to check the distributions of each feature in case there are some 'redundant' feature that does not contribute much, or contains outliers that need to be eliminated. Therefore, for categorical variable, I drew bar plots. For numerical features, I drew histograms and box plots, because histograms are good at showing distributions, and box plots add things by showing 0.5 and 0.75 percentile, meanwhile identify outliers. Diagrams are show below. (only include the graphs of one feature for brevity)

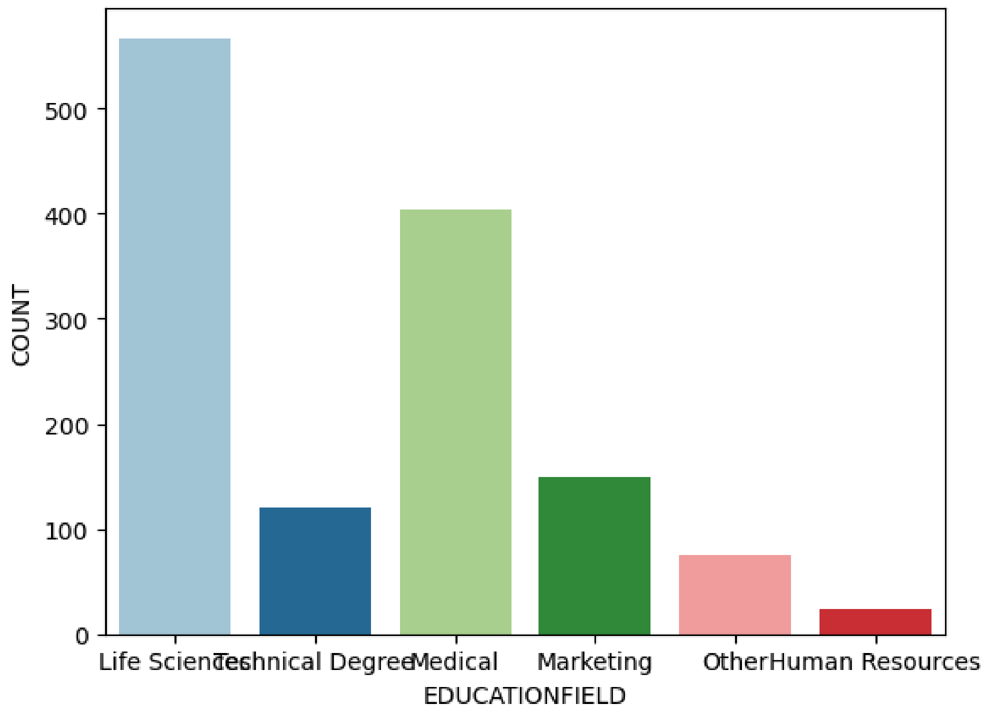

Figure 1: Bar plot for EducationField

The bar plot of "Attrition" shows there are 88% of people did not attrit, only around 11% of people attritted. So, we can see the ratio of attrition is not high, and this can serve as a criterion for checking whether our prediction is reasonable. Other bar plots are quite usual except for one - "Over18", which has only one bar on the plot. It means all of the people in the training set is over 18, and therefore this feature does not contribute much to our model. So, we can safely drop this column before model training. Another thing worth noticing is that we also need to drop "EmployeeID", which also has no actual meaning to our algorithms.

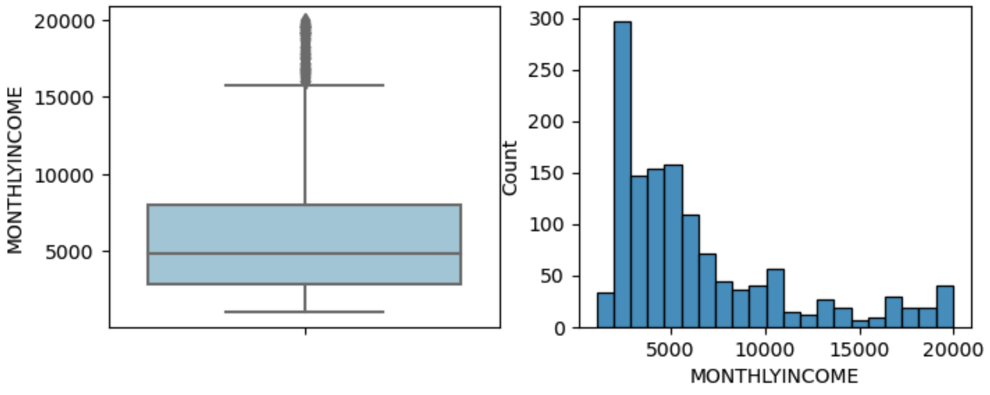

Figure 2: Box plot and histogram for MonthlyIncome

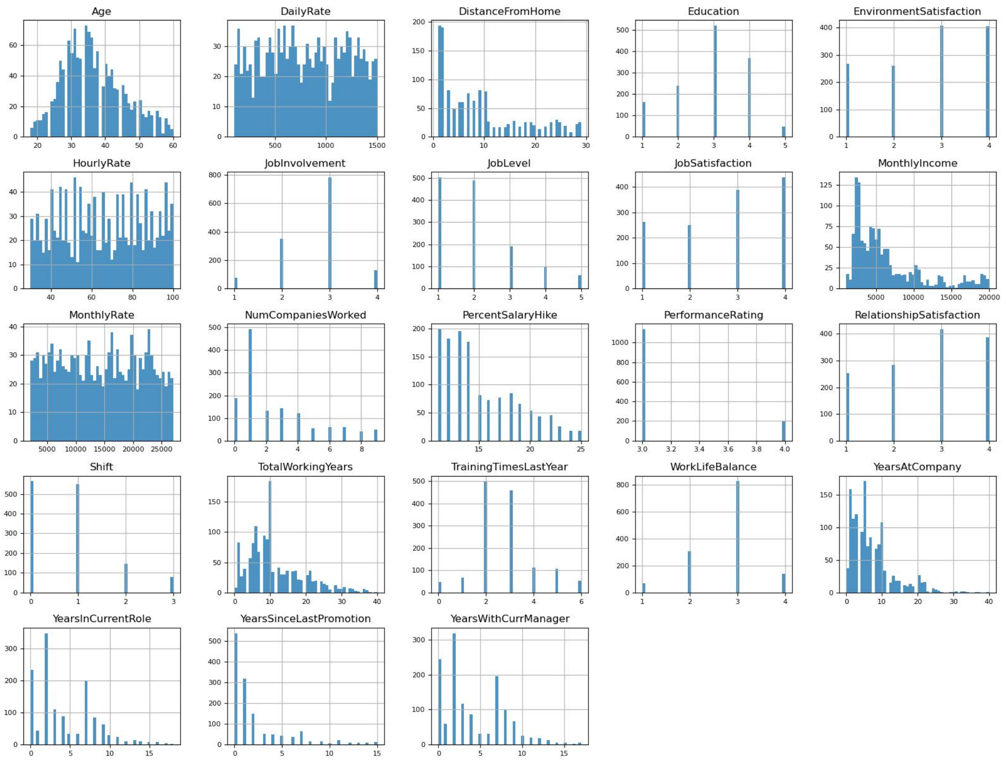

Figure 3: Histogram

For numerical data, I draw both box plots and histograms. After browsing all the plots we can see there are a few abnormal plots. One is "EmployeeCount", because every data entry counts one, so we can safely drop this column. And "StandardHours" of all individuals are the same. So, the features we need to drop are "EmployeeID","StandardHours", "EmployeeCount","Over18".

Moreover, the box plot also show another valuable information - some features have outliers that may have bad impact for our model. Since outliers have many sources, including error input, extreme values, they will influence the overall distribution of the data set and cause inaccurate model estimation. Thus, we need to remove outliers of these features.

2 Feature Engineering

In this part, I need to do some feature engineering to select the most useful features. Because data and features define the upper limit of machine learning models, while we are just trying to converge to this upper limit using models.

2.1 Correlation Analysis

Correlation analysis is an important step of feature engineering. To ensure each feature is valuable and contribute to models, we need to remove the highly-correlated ones. And the most straightforward approach is using correlation matrix, setting a threshold and delete features that has correlation above that threshold. I draw a heat map to show the correlations, and the closer the color is to blue, the higher the correlations of the two features are.

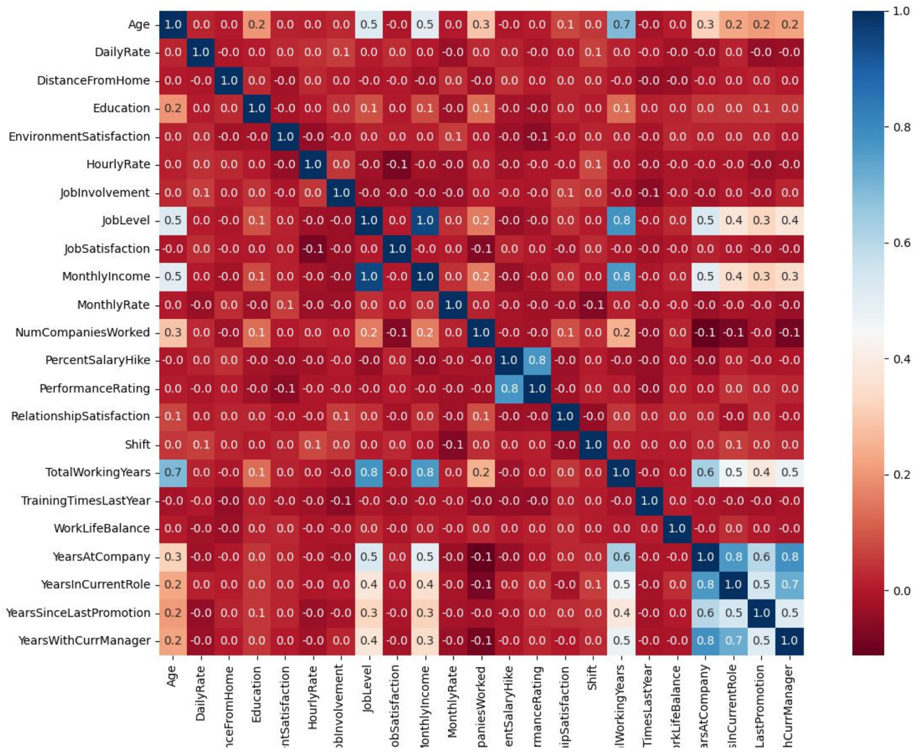

Figure 4: Correlation heat map

We can see a few squares have very blueish color with correlation over 0.6. Because I want to maintain as much information as possible while getting rid of redundant features, I set the threshold to be 0.7 - any features has a correlation above 0.7 with another one will be removed. Finally, we have a list of features to drop: ["PerformanceRating", "TotalWorkingYears", "YearsInCurrentRole", "YearsWithCurrManager"] Dropping these are also reasonable from intuition. Because the total working years should be highly correlated with the years spent with current role or manager, and employee's monthly income could also be hooked with his or her performance.

2.2 Encoding Categorical data

After removing some highly correlated features, the next step I need to do is converting categorical data to numerical data. And I chose `Sklearn.LabelEncoder` to encode these features.

2.3 Outlier Detection

As I mentioned before, we need to remove outliers from numerical data. Therefore, I wrote a function that can detect outliers and return there indices. I chose the quantile method - first compute interquantile range (IQR), i.e. the difference between 0.75 and 0.25 quantile, then set the upper bounds to be 0.75 quantile + 1.5 * IQR, and lower bounds are similarly defined, so that any data lies outside of the interval - [lower bound, upper bound] - will be considered as outliers.

2.4 Feature Selection

I use `Sklearn.SelectKbest` to select the best features and set k to be 20. This method is basically computing the dependence of each feature to the label and retain those that have higher dependence.

2.5 Scaling Numerical Data

Since we have over 20 features and each numerical feature has different meaning and unit. For example, the 'MonthlyIncome' should be in dollars, while 'DistanceFromHome' should be in miles. The primary reason for scaling these data is that we want the algorithm to treat the *ranking* or *relative magnitude* of each entry seriously instead of the *absolute values*. Because in the latter case, the algorithm would think a monthly income of 16183 is more important than 10 total working years. Fortunately, rescale features will largely solve this problem.

Therefore, I use `Sklearn.Preprocessing.StandardScaler` to transform data to be in the same range for better training and prediction.

3 Models

I chose tree-based model because they are non-parameterized and are easier to visualize, I used `AdaBoost` and `RandomForest`.

3.1 Model 1

Reasons why I chose `AdaBoost` are as follows.

1. robustness - it is relatively robust to overfitting in low noise datasets, because it ensembles many weak tree classifers that underfit (high bias) and combine those predictions into a stronger model, and reduce the overfitting (high variance);
2. ease of hyperparameter tuning - it has only a few hyperparameters that need to be tuned;

3.2 Model 2

Random forests are bagged decision tree models that split on randomly selected subset of features on each split. Reasons why I chose `RandomForest` are as follows.

1. it can handle all kinds of data, including numerical, categorical, binary features, little preprocessing need to be done before model training;
2. we also do not need to rescale data if using Random Forest;
3. efficiency of training - it is faster than other boosted decision trees because we are only training on subsets of features, which greatly improves training speed and efficiency.

4 Training

4.1 Data splits

I used `Sklearn.model_selection.train_test_split` to split training set and validation set - 80% are train set and 20% are validation set. When doing cross validation, I chose k to be 5.

4.2 Algorithms

4.2.1 AdaBoost

1. Algorithm

   (a) Assign each data point the same weight;

   (b) Train weak classifiers, and identify the classification result;

   (c) Update the weights of each data points. increase the weights of wrongly classified data and decrease the weights of those that are correctly classified;

   (d) Iterate the above steps for T times;

   (e) Finally, output the final classifier, which is a strong classifer.

2. Run time

   Adaboost does not have very apparent computation efficiency improvement, so the run time is slower than random forest, around 0.3 seconds.

4.2.2 Random Forest

1. Algorithm

   (a) Bootstrapping a sample of size n from the training set.

   (b) Draw a single decision tree follow the steps below:

   - Randomly choose m features;
   - Evaluate the splitting criteria on these features, and split on the best feature;
   - Repeat until the observations of this node reach a certain minimum threshold;

   (c) Iterate the above steps for T times.

(d) Output all trees, then use the majority-vote rule to decide the best tree and use this tree to make predictions.

2. Run time

   Since random forest only trains on a subset of features every time, I assume the training time won't be longer than that of Adaboost. Around 0.2 seconds shall be reasonable.

5 Hyperparameter Tuning

In each of the tuning process, I used `sklearn.model_selection.GridSearchCV`.

5.1 Hyperparameter Tuning for AdaBoost

The tuning parameters of Adaboost is quite lightweighted - there are only two hyperparameters need to be tuned: `n_estimators` and `learning_rate`. I set the parameter grid to be

```
params = {'n_estimators': [10, 25, 50, 75, 100, 200, 300, 500],
'learning_rate': [0.001, 0.01, 0.1, 0.5, 1., 5, 10, 20, 50, 100]}
```

The first parameter is `n_estimators`, for which I have tried a wider range of numbers, from 10 to 500. The second parameter is `learning_rate`, and I have tried 10 different values ranging from 0.01 to 100. And the functional relationship between predictive accuracy and learning rate is

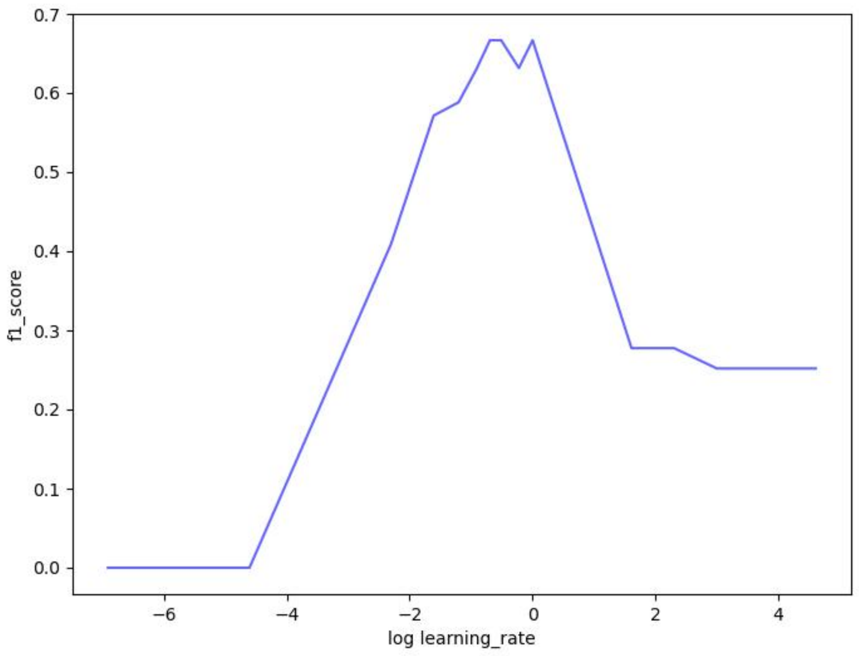

Figure 5: Learning rate VS f1 score

We can see that when `learning_rate` is around 0.5, the f1_score is the highest.

The best parameters are

```
{'learning_rate': 0.5, 'n_estimators': 200}
```

5.2 Hyperparameter Tuning for Random Forest

Hyperparameter tuning for random forest involves more parameter.

```
paramgrid_rf = {'max_depth': [5, 10, 15, 20, 25],
'min_samples_leaf': [i for i in np.linspace(1, 10, num = 5)],
'min_samples_split': [i for i in np.linspace(1, 10, num = 5)],
'n_estimators': [10,50,100],
'criterion': ['gini', 'entropy', 'log_loss']}
```

Note that random forest algorithm has more parameters than what I have tuned, but the change of other parameters did not yield significant improvement on the prediction result and accuracy of the model.

And the functional relationship between predictive accuracy and `min_samples_leaf` is

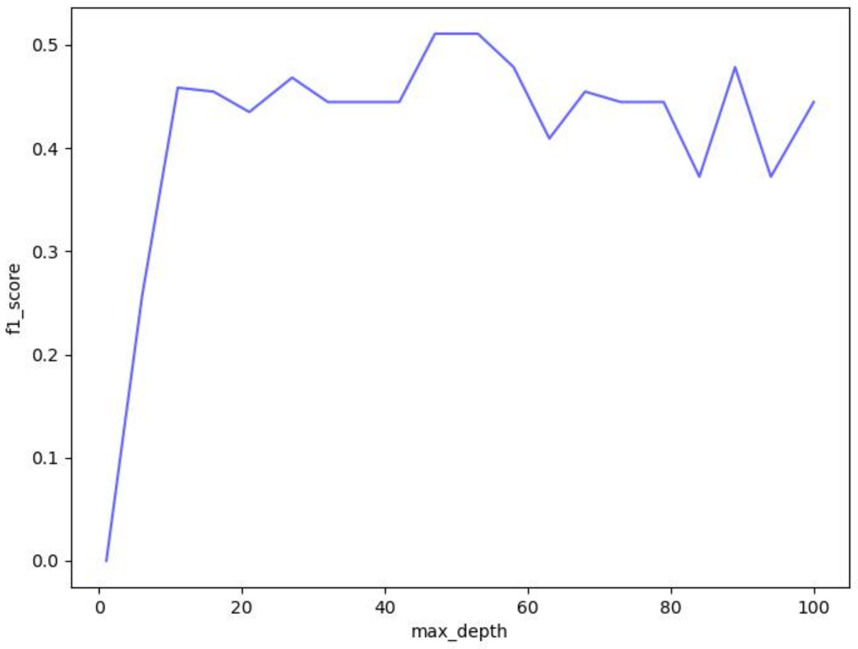

Figure 6: max depths VS f1 score

6 Predictive Accuracy

I finally got a f1 score of 0.606. The graphs below show the AUC curve and confusion matrix.

7 Errors and Mistakes

I think the most tricky part is data preprocessing and feature engineering. The first thing I ignored was that I should remove the outliers before model training. Because outliers will cause high bias. And I also did not rescale the data to the same range, which is very important because data with different magnitudes will be treated differently while in fact only the relative ranking matters. After implementing these two steps, f1_score improved from 0.5045 to 0.54.

And another thing is I should select 20 features instead of using all of the 26 features to train the model. Because some of the features are not relevant to the label, so applying `SelectKbest`, we can pick out the most relavant features.

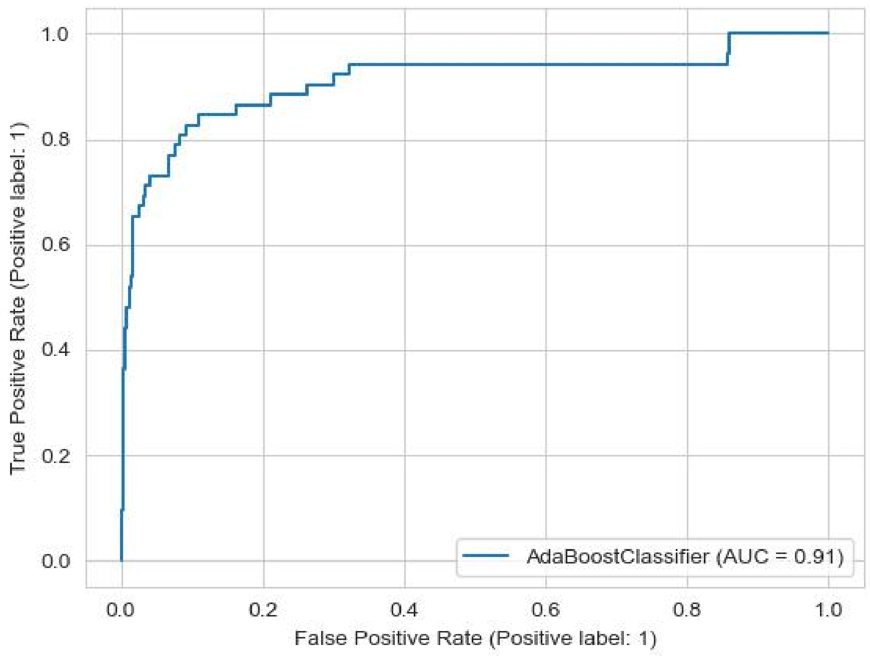

Figure 7: ROC curve

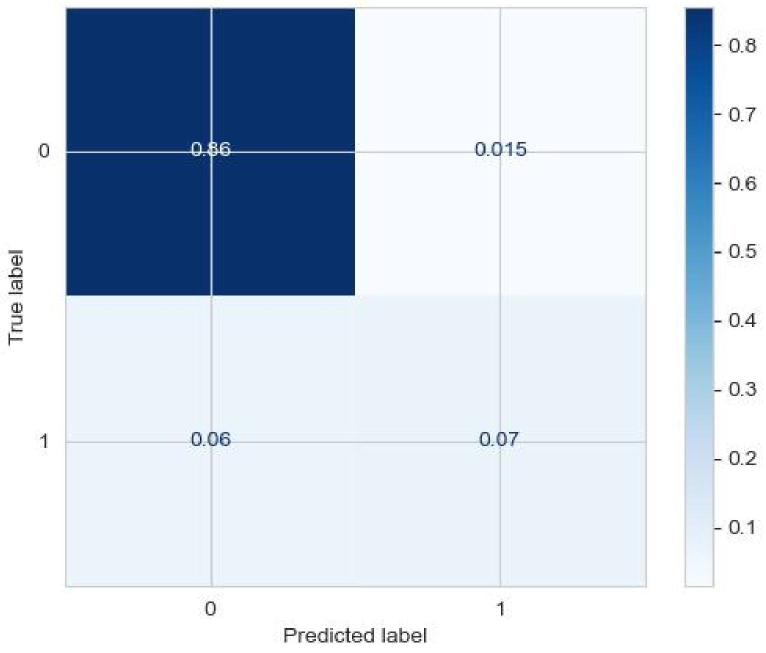

Figure 8: Confusion matrix

Code

Copy and paste your code into your write-up document. Also, attach all the code needed for your competition. The code should be commented so that the grader can understand what is going on. Points will be taken off if if the code or the comments do not explain what is taking place. Your code should be executable with the installed libraries and only minor modifications.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import seaborn as sns
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import plot_roc_curve, roc_auc_score, f1_score,
    confusion_matrix
import time

import warnings
warnings.filterwarnings('ignore')

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

Data decription

```python
def DescribeData(dataframe):
    print('\n', '-' * 20, 'shape'.upper(), 20 * '-')
    print(dataframe.shape)
    print('\n', '-' * 20, 'dtypes'.upper(), 20 * '-')
    print(dataframe.dtypes)
    print('\n', '-' * 20, 'columns'.upper(), 20 * '-')
    print(dataframe.columns)
    print('\n', '-' * 20, 'info'.upper(), 20 * '-')
    print(dataframe.info())
    print('\n', '-' * 20, 'any null values'.upper(), 20 * '-')
    print(dataframe.isnull().values.any())
    print('\n', '-' * 20, 'null values'.upper(), 20 * '-')
    print(dataframe.isnull().sum().sort_values(ascending=False))
    print('\n', '-' * 20, 'descriptive statistics'.upper(), 20 * '-')
    print(dataframe.describe())
DescribeData(train)

# find categorical and numerical features
def FindNumOrCategFeature(dataframe, dtype):
    '''''
    Report the numerical or categorical features
    for the dataframe.
    dtype = 'numerical' or 'categorical' or 'both'
    '''''
    # categorical features
    cate = (dataframe.dtypes == 'object')
    categorical = list(cate[cate].index)

    # numerical features
    num = (dataframe.dtypes == 'int64')
    nums = list(num[num].index)

    if dtype == 'numerical':
        print('Categorical featrues:')
        print(categorical)

    elif dtype == 'categorical':
        print('Numerical featrues:')
        print(nums)

    elif dtype == 'both':
        print('Categorical featrues:')
        print(categorical)
        print('Numerical featrues:')
        print(nums)

    return categorical, nums
cates, nums = FindNumOrCategFeature(train, 'both')

def DescribeCateFeature(dataframe, categorical, plot=False):

    print('\n', '-' * 10, categorical.upper(), 10 * '-')
    print(pd.DataFrame({
        categorical: dataframe[categorical].value_counts(),
        'RATIO (%)': round(100 * (dataframe[categorical].value_counts() /
   len(dataframe)), 2)
    }))

    if plot == True:
        sns.countplot(x=dataframe[categorical], palette='Paired')
        plt.ylabel('COUNT')
        plt.xlabel(categorical.upper())
        plt.show()

for c in cates:
    DescribeCateFeature(dataframe=train, categorical=c, plot=True)
```

drop useless columns

```
train1 = train.copy()
test1 = test.copy()
# drop 'EmployeeID','STANDARDHOURS', 'EmployeeCount','Over18'
# all are over 18
# employee count is always 1
# standard hours is always 80
# employee ID does not conribute to model estimation
train2 = train1.drop(columns=['EmployeeID','StandardHours', 'EmployeeCount
    ','Over18'])

train2.hist(figsize=(20, 15), bins=50, xlabelsize=8, ylabelsize=8, alpha
    =0.8);
plt.savefig('histogram.jpg')
```

Feature engineering for training set

```
corr = train2.corr()
corr

def CorrAnalysis(dataframe, corr_threshold=0.7, plot=False):
    corr = dataframe.corr()
    cor_matrix = corr.abs()
    upper_triangle_matrix = cor_matrix.where(
        np.triu(np.ones(cor_matrix.shape), k=1).astype(np.bool_))
    drop_list = [col for col in upper_triangle_matrix.columns
                 if any(upper_triangle_matrix[col] > corr_threshold)]

    if plot == True:
        plt.figure(figsize=(15, 10))
        sns.heatmap(corr, cmap='RdBu', square=True, annot=True, fmt='.1f')
#          plt.show()
```

```
        plt.savefig('corr.jpg')

    return drop_list

CorrAnalysis(train2, plot=True)

# drop highly correlated features
features_to_drop = ['MonthlyIncome',
 'PerformanceRating',
 'TotalWorkingYears',
 'YearsInCurrentRole',
 'YearsWithCurrManager']
train3 = train2.drop(columns=features_to_drop)
train3.shape
train3.columns
```

Encoding cateorical data

```
train3c = train3.copy()
cates1 = list(set(cates).intersection(set(train3c.columns)))
cates1

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit_transform(train3c[cates1[-3]])

def LabelEncoder1(dataframe, feature_list):

    for f in feature_list:
        le = LabelEncoder()
        encoded = le.fit_transform(dataframe[f])
        dataframe[f] = encoded

    return dataframe

df = LabelEncoder1(dataframe=train3c, feature_list=cates1)
df.columns
```

Outlier detection

```
# define a function to detect and remove outliers

def DetectOutlier(dataframe, features, drop=False):
    '''
    Detect outliers using quantile method
    Feature: numerical variables, can be passed as int, float or list
    '''
    for feature in features:

        IQR = dataframe[feature].quantile(0.75) - dataframe[feature].quantile(0.25)
        lower = dataframe[feature].quantile(0.25) - (IQR * 2.5)
        upper = dataframe[feature].quantile(0.75) + (IQR * 2.5)
        out_of_bound = dataframe[feature].loc[(dataframe[feature] < lower)
                                              | (dataframe[feature] > upper)]
        oob_idx = list(out_of_bound.index)

        if len(oob_idx) != 0:
            print('Feature {feature} has outliers.'.format(feature=feature))
            print('Outliers index of feature {feature} are: \n'.format(feature=feature), oob_idx)
            print('Outliers range of feature {feature} are < {lower} or > {upper}\n'.format(
                feature=feature, lower=lower, upper=upper))

        if drop == True:
            dataframe = dataframe.drop(index=oob_idx)

    return dataframe

nums1 = list(set(nums).intersection(set(train3c.columns)))
df_o = DetectOutlier(dataframe=df, features=nums1, drop=True)
```

Feature selection

```
from sklearn.feature_selection import SelectKBest, chi2

X_ = df_o.drop(columns='Attrition')
y_ = df_o['Attrition']
slkb = SelectKBest(score_func=chi2,k=20)
X1 = slkb.fit_transform(X_, y_)

cols = slkb.get_support(indices=True)
df_new = pd.DataFrame(X1)
df_new
```

Scaling numerical data

```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler

nums2 = list(set(nums).intersection(set(df_new.columns)))

def ScaleData(data, feature, scaler, inplace=False):

    if scaler == 'MinMax':
        min_ = np.min(data[feature])
        max_ = np.max(data[feature])

        if inplace == True:
            data[feature] = (data[feature] - min_) / (max_ - min_)

    if scaler == 'Standard':
        mu = np.mean(data[feature])
        std = np.std(data[feature])

        if inplace == True:
            data[feature] = (data[feature] - mu) / std

    return data

for f in df_new.columns:
    ScaleData(df_new, f, 'Standard', inplace=True)

df_new
```

Model training

```python
# split data
X = df_new
y = y_
X_train, X_test, y_train, y_test = train_test_split(
                            X, y, test_size = 0.2, random_state = 2022)

from sklearn.metrics import roc_auc_score, plot_confusion_matrix,
    roc_curve, f1_score

def TrainModelAndEval(model, X_train, X_test, y_train, y_test,
                      plot_ROC=False, plot_confusion_mat=False):
    start = time.time()
    # train model
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    # predict
    y_pred = model.predict(X_test)

    # training accuracy
    f1score = f1_score(y_train, y_train_pred)
    auc = roc_auc_score(y_train, y_train_pred)
    train_time = time.time() - start

    # test accuracy
    f1score_t = f1_score(y_test, y_pred)
    auc_t = roc_auc_score(y_test, y_pred)

    print('-------Training Accuracy--------')
    print('F1 score = {}'.format(f1score))
    print('AUC = {}'.format(auc))
    print('Training time = {} \n'.format(train_time))

    print("-------Testing Accuracy--------")
    print("F1 score of test set = {}".format(f1score_t))
    print("AUC of test set = {}".format(auc_t))

    if plot_ROC == True:
        probs = model.predict_proba(X_train)
        fpr, tpr, thresholds = roc_curve(y_test, probs)
        plot_roc_cur(tpr, fpr)

    if plot_confusion_mat == True:
        plot_confusion_matrix(model, X_test, y_test,cmap=plt.cm.Blues,
    normalize = 'all')

    return model, f1score_t, auc_t, train_time

# random forest
from sklearn.ensemble import RandomForestClassifier

params_rf = {'max_depth': 10,
             'min_samples_leaf': 2,
             'min_samples_split': 2,
             'n_estimators': 100,
             'random_state': 2022}

model_rf = RandomForestClassifier(**params_rf)
model_rf, f1_score_rf, auc_rf, train_time_rf = TrainModelAndEval(
                                    model_rf, X_train, X_test, y_train,
    y_test)

# adaboost
from sklearn.ensemble import AdaBoostClassifier

params_ada = {'n_estimators': 100}
model_ada = AdaBoostClassifier(**params_ada)
model_ada, f1_score_ada, auc_ada, train_time_ada = TrainModelAndEval(
                                     model_ada, X_train, X_test, y_train,
    y_test)
```

Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV


params = {'n_estimators': [10, 25, 50, 75, 100, 200, 300, 500],
          'learning_rate': [0.001, 0.01, 0.1, 0.5, 1., 5, 10, 20, 50, 100]}

adaboost = AdaBoostClassifier()
clf = GridSearchCV(adaboost, params)
clf.fit(X_train, y_train)
best_params = clf.best_params_
print(best_params)
model_ada_b = AdaBoostClassifier(**best_params)
model_ada_b, f1_score_ada_b, auc_ada_b, train_time_ada_b =
    TrainModelAndEval(
                                     model_ada_b, X_train, X_test, y_train
    , y_test)

# plot f1 score VS learning rate

# lrate =  [0.001, 0.01, 0.1, 0.5, 1., 5, 10, 20, 50, 100]
lrate = [0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1, 5, 10, 20, 50,
     100]

f1s = []
for l in lrate:
    ada_clf = AdaBoostClassifier(n_estimators=100, learning_rate =l)
    ada_clf.fit(X_train, y_train)
    y_pred = ada_clf.predict(X_test)
    f1 = f1_score(y_test, y_pred)
    f1s.append(f1)
```

```python

fig1, ax1 = plt.subplots(figsize=(8,6))
log_lrate = np.log(lrate)
plt.plot(log_lrate, f1s, c='b',alpha=0.6)
plt.xlabel('log learning_rate')
plt.ylabel('f1_score')
plt.savefig('lrateADA.jpg')
```

```python
paramgrid_rf = {'max_depth': [5, 10, 15, 20, 25],
              'min_samples_leaf': [i for i in np.linspace(0.01, 1, num = 8)
    ],
              'min_samples_split': [i for i in np.linspace(0.01, 1, num =
    8)],
              'n_estimators': [10,50,100],
              'criterion': ['gini', 'entropy', 'log_loss']
              }
rf_clf = RandomForestClassifier()
clf = GridSearchCV(rf_clf, paramgrid_rf)
clf.fit(X_train, y_train)

depths = [int(i) for i in np.linspace(1, 100, num = 20)]

f1s_rf = []
for d in depths:
    rf_clf = RandomForestClassifier(n_estimators=100, max_depth =d)
    rf_clf.fit(X_train, y_train)
    y_pred = rf_clf.predict(X_test)
    f1_rf = f1_score(y_test, y_pred)
    f1s_rf.append(f1_rf)

fig1, ax1 = plt.subplots(figsize=(8,6))
# log_split = np.log(lrate)
plt.plot(splits, f1s_rf, c='b',alpha=0.6)
plt.xlabel('max_depth')
plt.ylabel('f1_score')
plt.savefig('maxdepthRF.jpg')
```

Feature engineering for test set

```python
# drop useless columns
test1 = test.drop(columns=['EmployeeID','StandardHours', 'EmployeeCount','
    Over18'])
# drop highly correlated features
test2 = test1.drop(columns=features_to_drop)
```

```python
# select features
test3 = slkb.transform(test2)
# rescale data
scaler = StandardScaler()
test4 = scaler.fit_transform(test3)
test4
```

Predict and submit

```python
y_pred_ada = model_ada.predict(test4)
y_pred_ada
y_pred_rf = model_rf_b.predict(test4)
y_pred_rf

ada = pd.DataFrame(y_pred_ada)
submit_ada = ada.reset_index()

submit_ada.columns=[['id','Predicted']]
submit_ada.to_csv('submission_ada_01.csv', index=False)

submit_rf = pd.DataFrame(y_pred_rf)
submit_rf = submit_rf.reset_index()

submit_rf.columns=[['id','Predicted']]
submit_rf.to_csv('submission_rf_01.csv', index=False)
```