Predicting job attrition in the health sector

Introduction

There is a shortage of healthcare employees and we cannot afford losing the workers that we have. The reason for an employee quitting is usually attrition caused by different job or home related reasons. That is why it is important to keep an eye on the employees happiness and job environment.

In this report we are going to introduce a machine learning model that is supposed to predict if an employee is going to quit because of attrition. The type of this machine learning model and the data used to train it is discussed in the problem formulation section. Methods section explains the selection criteria for features and methods. Then we have the results section with the results analysis followed by the conclusion section with a summary of the machine learning problem based on the results. References and the project related python code can be found at the end of the report.

Problem formulation

The dataset is fetched from Kaggle[1].

The dataset is about employees of the healthcare sector. A single row represents a single individual and columns represent different features of the individuals. A single datapoint of an individual is a number or a text. For instance, one feature is age. The total number of the columns in the dataset is 11, and we are going to use 10 of the columns as features to predict our model. The question we are trying to answer is if the employee is going to stop working in the healthcare sector in their career. Therefore, we are going to use the “attrition” column as our label.

Because we are using labeled input and output data with classification algorithms our machine learning model is supervised.

Methods

Our final dataset has 1676 data points and every data point’s features are represented as number values. Most features are continuous, but some are also binary. For example, employees working overtime are represented as binary values, but on the other hand salary is continuous value.

We are going to choose our features from the original data set based on how we believe they will correlate with job attrition. For instance, satisfaction level is correlated with job attrition and hence will be a good choice for one of the features. The original dataset

includes 35 columns but we will narrow it down to 14 and 13 of them will be used as features. We are going to drop columns that we don't understand and wouldn't have the strongest correlation with job attrition. For instance, the dataset shows a "EmployeeID" column that doesn't correlate with the label so we will drop it. Also we will view the correlation between the features and take out the features that had a strong correlation between each other. We think it would lead to more precise results to not have strongly correlating features so that having a one value would not almost certainly lead to another one.

Our problem is basically a classification problem because we have multiple features in the dataset that determine one binary label. The first method we are going to use is Logistic Regression because it fits our classification task where we have two options for the label. It is a common and effective method for binary classification because we are only viewing a distribution between two values.

Logistic regression uses logistic loss as a loss function. The logistic loss function tells us how much the predicted value differs from the actual value. To have the most accurate prediction with the logistic regression the loss function should be minimized. We will use the logistic loss function because it is a ready to use library function and common to be used along with logistic regression.

The second classification method we are going to use is a decision tree. It is constructed from nodes that can be true or false. The best way to understand a decision tree is to see a visual projection, but it basically follows a path of nodes depending whether the feature value in the node is true or false. At the end of the path it will give out the predicted value of the label. The decision tree's path is constructed by how well a specific feature can split the data. If a feature can split the data almost 50/50, it will probably be the first node of the tree. We have only two values for our label so the decision tree will work well and it will probably be even more accurate than the logistic regression method.

We are going to choose Entropy or Gini impurity as our loss function for our second method. The Entropy is one of the loss functions used for the decision tree and it measures how many wrong data points are in the node. When using the Entropy as the loss function *a split is only performed if the Entropy of each the resulting nodes is lower than the Entropy of the parent node*[2]. Gini impurity is calculating the variance between different nodes.

To evaluate our models we are going to calculate the accuracy and the F1 score of the models. The accuracy score tells the fraction of the predictions that were predicted right. F1 score is merging two important values - Precision and Recall. When they both are taken into consideration, the F1 score takes into account both false positives and false negatives. If examined separately, precision measures the errors caused by false

positives whereas recall measures the errors caused by false negatives[3]. Based on these values we will decide if the Logistic regression or the Decision tree is a better choice to make the prediction.

Empirical studies have shown a split of 20-30% for validation and 70-80% for training has shown the best prediction results[4]. First we are going to use just a basic split to 30/70 from which 70% is for the training. We will furthermore split the 30% that is left to a test set of ⅓ and a validation set of ⅔ . We are going to split our data into three groups because we believe a simple split is going to have enough accuracy for our prediction and it would not make our data set too divided, and therefore too hard to understand.

Results

Logistic regression accuracy and error:

	Training	Validation	Test	F1-score
accuracy	0.92583	0.89021	0.88554	0.51282
error	2.56171	3.79211	3.95324	

The accuracy values for the training, validation and test sets are all quite high which indicates that the model would be a good choice for the prediction model. In addition, the error values are not too high and very different between the sets. This indicates that the model is working well with this data split.

Decision tree accuracy:

Training	Validation	Test	F1-score
0.96249	0.87537	0.88554	0.53659

The accuracy values for the training, validation and test sets are all quite high which indicates that the model would be a good choice for the prediction model. We selected the Gini impurity as our loss function because with that we got slightly more accurate results than by using Entropy. We tried different max depths for the decision tree and ended with the max depth six which delivered the most accurate predictions. The max depth larger than 6 would have caused overfitting due to the training accuracy being too close to one.

Final chosen method:

We decided to choose logistic regression because the accuracy of the validation set is slightly higher than it would be if we would have used the decision tree. Furthermore, the errors that we were able to calculate with the logistic loss are more reliable than just giving the algorithm an input of Gini impurity or entropy. Even though the decision tree has a better F1-score, the logistic regression's F1-score is not too far away and false positives are not serious in our prediction. Ultimately, we chose logistic regression as our model based on the accuracy scores and the seen errors.

The test set of the logistic regression method was constructed by first splitting the original dataset to 20/80 sets and furthermore taking the ⅓ out of the 20%. The test set was therefore the smallest set from the three. As it is seen in the table above the test error for our final chosen method was 3.95. The validation error and the test error are almost the same and there is not too big of a gap between the test error and the training error which means that the model is fitting. Also the logistic regression has smaller training accuracy than the decision tree which could indicate that the decision tree is a worse choice due to overfitting.

Conclusion

In conclusion, the final chosen method is valid for prediction purposes. The accuracy of the final chosen method is good enough for a working model. The accuracies produced by the logistic regression are also quite close to each other which indicates a fitting model. The F1-score has some room for improvement even though it was not too much off. The confusion matrix tells us that we have very many true negatives but very few true positives. This means that our data is imbalanced which decreases the precision of our model.

There are a couple things we should consider and do to make the model work better. First, we should make sure the data is balanced and has an almost 50/50 label ratio so that the training could be done better. We could have taken more of the features from the original data set so that the prediction would have more to compare and evaluate.

References:

1. https://www.kaggle.com/datasets/jpmiller/employee-attrition-for-healthcare
2. https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575
3. https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3
4. https://scholarworks.utep.edu/cs_techrep/1209/

project

October 7, 2022

[211]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  #data visualization library

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix  # evaluation␣
 ↪metrics
from sklearn.metrics import log_loss
from sklearn.tree import DecisionTreeClassifier
```

[212]:
```
df = pd.read_csv('watson_healthcare_modified.csv')
df.drop(columns=['EmployeeID','EmployeeCount','Department','Gender',
                 'EducationField','Over18','BusinessTravel','JobRole',
                 'Shift','Education','MaritalStatus','TrainingTimesLastYear',␣
 ↪'JobInvolvement', 'JobLevel',
                 'RelationshipSatisfaction', 'StandardHours',␣
 ↪'YearsInCurrentRole', 'YearsWithCurrManager',
                 'YearsInCurrentRole', 'YearsSinceLastPromotion',␣
 ↪'PerformanceRating', 'YearsAtCompany', 'TotalWorkingYears'
                ],inplace=True)
df.replace('No', 0, inplace=True)
df.replace('Yes', 1, inplace=True)
df.head(5)
```

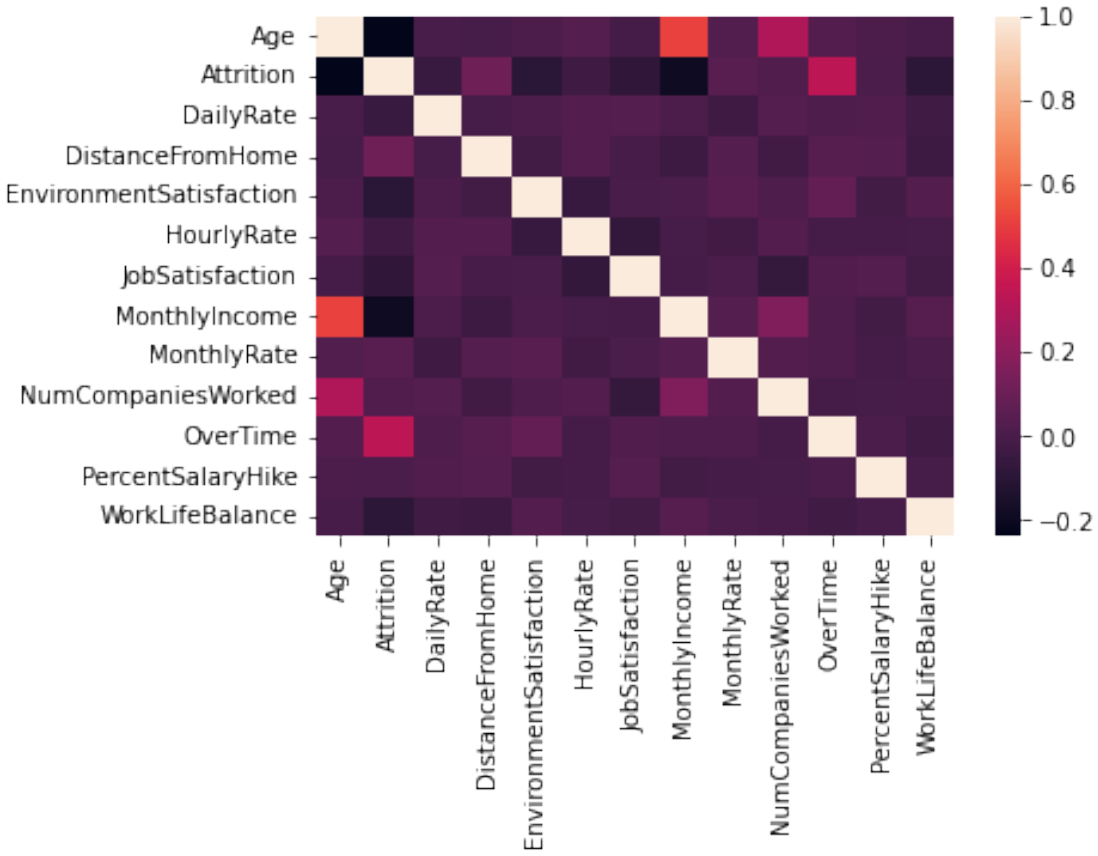
[212]:

	Age	Attrition	DailyRate	DistanceFromHome	EnvironmentSatisfaction	HourlyRate	JobSatisfaction	MonthlyIncome	MonthlyRate
0	41	0	1102	1	2	94	4	5993	19479
1	49	0	279	8	3	61	2	5130	24907
2	37	1	1373	2	4	92	3	2090	2396
3	33	0	1392	3	4				
4	27	0	591	2	1				

```
3          56                3            2909         23159
4          40                2            3468         16632

   NumCompaniesWorked  OverTime  PercentSalaryHike  WorkLifeBalance
0                   8         1                 11                1
1                   1         0                 23                3
2                   6         1                 15                3
3                   1         1                 11                3
4                   9         0                 12                3
```

[213]:
```
sns.heatmap(df.corr())
plt.show()
```



[214]:
```
X = df.drop('Attrition', axis=1)
y = df['Attrition']

X_train, X_, y_train, y_ = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

```
X_val, X_test, y_val, y_test = train_test_split(X_, y_, test_size=0.33,␣
 ↪random_state=42)
```

[215]:
```
# Logistic regression:

logr = LogisticRegression(max_iter = 2000)
logr.fit(X_train, y_train)

y_pred_train = logr.predict(X_train)
acc_train = accuracy_score(y_train, y_pred_train)

y_pred_val = logr.predict(X_val)
acc_val = accuracy_score(y_val, y_pred_val)

y_pred_test = logr.predict(X_test)
acc_test = accuracy_score(y_test, y_pred_test)

print("Train accuracy:",acc_train)
print("Validation accuracy:",acc_val)
print("Test accuracy:",acc_test)
```

```
Train accuracy: 0.9258312020460358
Validation accuracy: 0.8902077151335311
Test accuracy: 0.8855421686746988
```

[216]:
```
# Logistic loss:

logloss_train = log_loss(y_train, y_pred_train)
logloss_val = log_loss(y_val, y_pred_val)
logloss_test = log_loss(y_test, y_pred_test)

print("Logistic loss for train: ", logloss_train)
print("Logistic loss for val: ", logloss_val)
print("Logistic loss for test: ", logloss_test)
```

```
Logistic loss for train:  2.56170770803614
Logistic loss for val:  3.792105413045331
Logistic loss for test:  3.9532430764949607
```

[219]:
```
# confusion matrix for test data:
conf_mat = confusion_matrix(y_test, y_pred_test)
print("Confusion matrix:\n",conf_mat)
```

```
Confusion matrix:
 [[137   2]
 [ 17  10]]
```

[258]:
```
precision = conf_mat[1][1]/(conf_mat[1][1] + conf_mat[0][1])
recall = conf_mat[1][1]/(conf_mat[1][1] + conf_mat[1][0])
f1 = 2 * ((precision*recall)/(precision+recall))

print("Precision: ", precision)
print("Recall: ", recall)
print("F1-score: ", f1)
```

```
Precision:  0.8333333333333334
Recall:  0.37037037037037035
F1-score:  0.5128205128205128
```

[240]:
```
# Decision tree

# choosing the max_depth:

accuracies_train = []
accuracies_val = []
accuracies_test = []
for i in range(1, 13):
    tree = DecisionTreeClassifier(criterion="gini", max_depth=i)
    tree.fit(X_train, y_train)
    y_pred_train = tree.predict(X_train)
    acc_train = accuracy_score(y_train, y_pred_train)
    accuracies_train.append(acc_train)
    y_pred_val = tree.predict(X_val)
    acc_val = accuracy_score(y_val, y_pred_val)
    accuracies_val.append(acc_val)
    y_pred_test = tree.predict(X_test)
    acc_test = accuracy_score(y_test, y_pred_test)
    accuracies_test.append(acc_test)
print(accuracies_train, "\n")
print(accuracies_val, "\n")
print(accuracies_test, "\n")
```

```
[0.8934356351236147, 0.9130434782608695, 0.9266837169650469, 0.9335038363171355,
0.9514066496163683, 0.9624893435635123, 0.9744245524296675, 0.989769820971867,
0.9957374254049446, 0.9982949701619779, 0.9982949701619779, 0.9982949701619779]

[0.8605341246290801, 0.8783382789317508, 0.8783382789317508, 0.8783382789317508,
0.8605341246290801, 0.8783382789317508, 0.8664688427299704, 0.8694362017804155,
0.8694362017804155, 0.8694362017804155, 0.8635014836795252, 0.8635014836795252]

[0.8373493975903614, 0.8614457831325302, 0.8493975903614458, 0.8614457831325302,
0.8795180722891566, 0.891566265060241, 0.891566265060241, 0.8795180722891566,
0.891566265060241, 0.8855421686746988, 0.891566265060241, 0.8734939759036144]
```

[255]:
```
tree = DecisionTreeClassifier(criterion="gini",max_depth=6)
tree.fit(X_train, y_train)

y_pred_train2 = tree.predict(X_train)
acc_train2 = accuracy_score(y_train, y_pred_train2)

y_pred_val2 = tree.predict(X_val)
acc_val2 = accuracy_score(y_val, y_pred_val2)

y_pred_test2 = tree.predict(X_test)
acc_test2 = accuracy_score(y_test, y_pred_test2)

print("Train accuracy: ",acc_train2)
print("Validation accuracy: ",acc_val2)
print("Test accuracy: ",acc_test2)
```

```
Train accuracy:  0.9624893435635123
Validation accuracy:  0.8753709198813057
Test accuracy:  0.8855421686746988
```

[248]:
```
# confusion matrix for test data:
conf_mat2 = confusion_matrix(y_test, y_pred_test2)
print("Confusion matrix:\n",conf_mat2)
```

```
Confusion matrix:
 [[136   3]
 [ 16  11]]
```

[256]:
```
precision = conf_mat2[1][1]/(conf_mat2[1][1] + conf_mat2[0][1])
recall = conf_mat2[1][1]/(conf_mat2[1][1] + conf_mat2[1][0])
f1 = 2 * ((precision*recall)/(precision+recall))
print("Precision: ", precision)
print("Recall: ", recall)
print("F1-score: ", f1)
```

```
Precision:  0.7857142857142857
Recall:  0.4074074074074074
F1-score:  0.5365853658536585
```

[ ]:
```

```