Duke CS671 Fall 2022 Kaggle Competition

Shucheng Zhang sz255

December 10, 2022

1 Exploratory Analysis

1.1 Dataset Description and Clean

The dataset used in this competition contains 35 features for 1470 samples of two classes (Attrition and No Attrition). In these features, 26 features are numbers and 9 are categories. After encoding, the feature details can be shown in Figure 1.

Figure 1: Dataset

From Figure 1, we can see this dataset is relatively clean, so any further clean will be done in this project is in order to improve the prediction performance [1-4]. There are 5 points we noticed should be improved.

- *EmployeeID* is index number and will be deleted, because it doesn't related to the classification.
- *EmployeeCount*, *Over18* and *StandardHours* are constant, which will not impact our prediction.
- In *JobRole*, *administrative* and *admin* is the same feature and should be combined.
- Additional process may be required for numerical columns like *education* that actually represent categorical variables.
- Some features like *JobLevel* and *YearsAtCompany*, may not totally independent, which need further process.

1.2 Feature Engineering

After one-hot encoding, representing the categorical variables and omitting the unnecessary features, the dimension of features is 49 now. To comprehend the relationship between the dataset's features, the correlation matrix is frequently used. The correlation matrix for our dataset is shown in Figure 2.

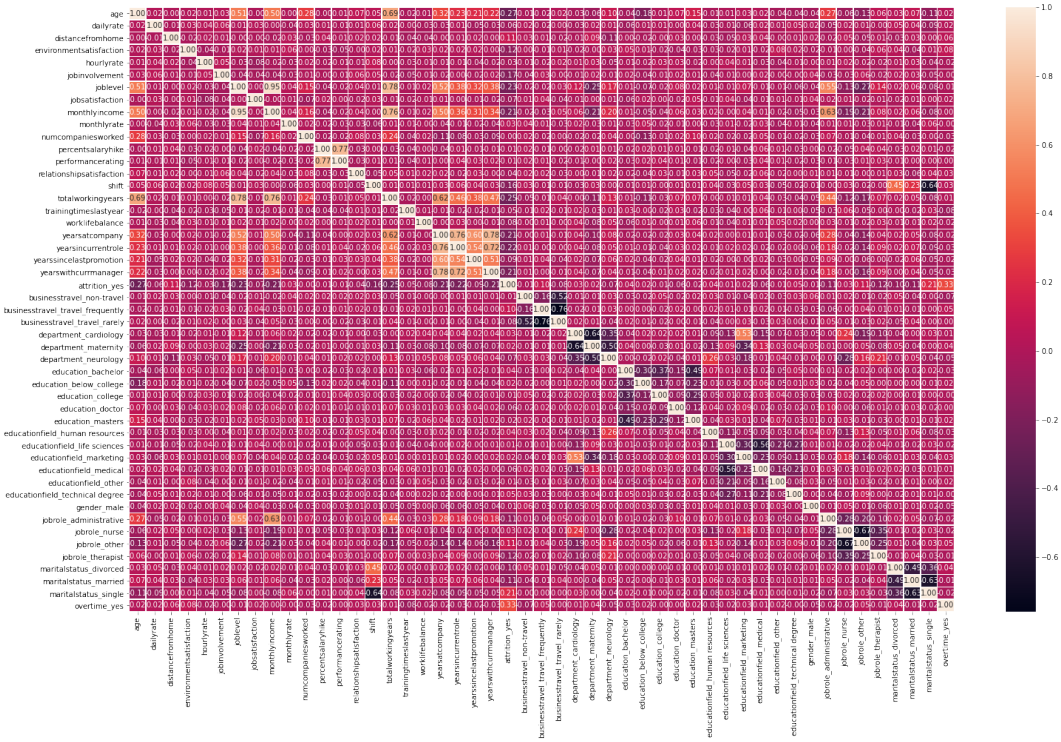


Figure 2: Correlation Matrix

Age From the Figure 3, the *Age* feature shows a normal distribution but with a slightly positive skew. Also, the attrition is related to the *Age*. Employee in early 20s are most likely to leave. After that, the turnover rate tends to decline until it reaches its lowest point in the 40s, and then, slightly increase until 60s.

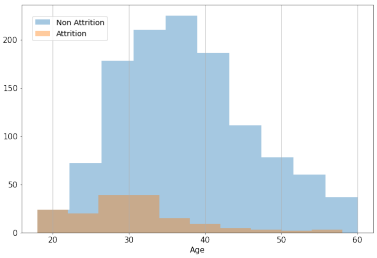


Figure 3: Age Distribution

Monthly Income From Figure 4, basically, higher paid jobs averages the lower attrition, while more employee with low income will leave. In the correlation matrix, we can see there are several features are related to the *MonthlyIncome*. Their relationship can be described as Figure 5.

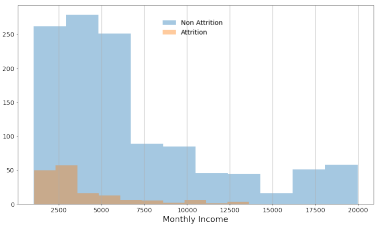


Figure 4: Monthly Income Distribution

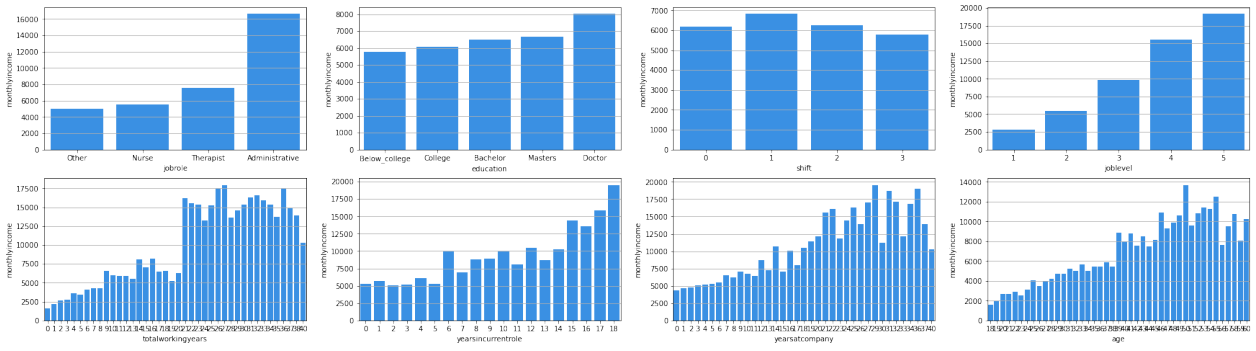


Figure 5: Monthly Income with its Related Features

OverTime Moreover, people who work overtime are more likely to quit.

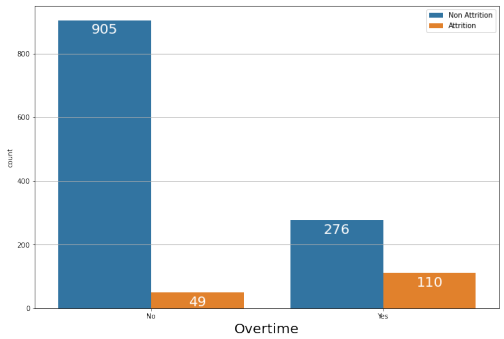


Figure 6: OverTime Distribution

Therefore, to take advantage of the relationship between the features, we multiply them together and get $48 * 48$ size of features (after minus *Attrition*). Then, in order to reduce dimension of input and prevent overfitting, we select more related features by Lasso Regression [8]. By decreasing some of the regression coefficients to zero, the LASSO approach regularizes model parameters. After that, the features corresponding to every non-zero weight is important here and will be used in the models later. Here, after lasso feature selection, we only left 319 features.

Attrition The predicted target *Attrition* shows that the samples in this dataset is not balanced. In another word, the original dataset contains 1470 employee records. Only 237 employees have left the organization, whereas 1233 employees still working, which bias the dataset towards the working employees. Due to this imbalance, the prediction model performs relatively poorly. To improve the model's performance, we use the Adaptive synthetic (ADASYN) sampling approach [7] to transform the dataset into its balanced version.

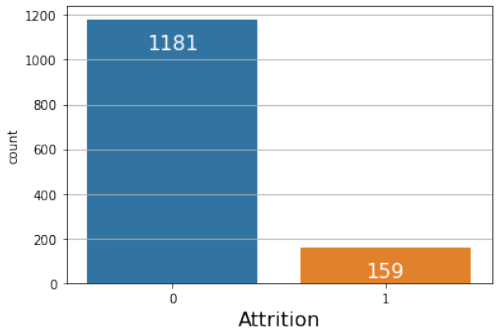


Figure 7: Attrition Distribution

Thus, the size of final training dataset is $(2382, 319)$, including 1191 attrition and 1191 non-attrition.

2 Models

In this project, we choose Adaboost [5] and Deep Neural Network [6]. The reasons are described in this section.

Adaboost

- AdaBoost has high accuracy and is easy to use with less need for adjusting parameters.
- AdaBoost is not prone to overfitting because of it joint optimization
- Compared with bagging algorithm and Random Forest algorithm, AdaBoost fully considers the weight of each classifier and can use different classification algorithms as weak classifiers.

Deep Neural Network

- Compared with traditional machine learning algorithm, DNN can fit almost any function, because the nonlinear fitting ability of DNN is very strong.
- Typically, DNN has a good performance on classification problem.
- There are many methods like *dropout* and *NomalizationLayer* can be used to prevent overfitting.
- Because of the development tool named Pytorch, DNN is much easy to implement and can be trained with a high computational efficiency.

3 Training

Adaboost

Here, we use choose *Grid Search* method to select the optimal parameters. The base estimator is Decision Tree due to the great performance of it in classification. The number of estimator will be set as large as possible to overcome overfitting. As for the learning rate will be set according to the number of estimator, because a higher learning rate increases the contribution of each classifier and there is a trade-off between the learning rate and the number of estimator. One Adaboost model can be trained in two minutes.

Deep Neural Network

For DNN, there are many parameters can be optimized because of the complex construct of it. The key elements affecting the accuracy includes training epoch, learning rate, the number of hidden layer, the number of neurons and the activation function. Among them, the activation functions of hidden layers are softplus, which is a curvy version of ReLU, while the activation functions of input and output layers are sigmoid. The reason why we use sigmoid function in output layer is that it exists between $(0, 1)$, so it is especially used for models where we have to predict the probability as an output. Other parameters will be chosen by grid search. Moreover, the Dropout layer is used to prevent overfitting. Typically, the dropout probability is 0.2 to balance the accuracy and the generalization. Finally, we choose Adam and MSEloss as optimizer and loss function repectively. Because of the Cuda, one DNN model can be trained less than one minutes if the training epoch is less than 100.

4 Hyperparameter Selection

As described before, most of the hyperparameters are selected by grid search in a reasonable range.

Adaboost The parameters chosen ranges are shown in this Table 1.

Table 1: **Adaboost Training Details**

Learning rate	$[0.01, 0.1, 1]$
Number of estimator	$[500, 600, 700, 800, 900, 1000, 1100, 1200]$

From Figure 8, we choose $learn\_rate = 0.01$ and $n\_estimator = 1100$.

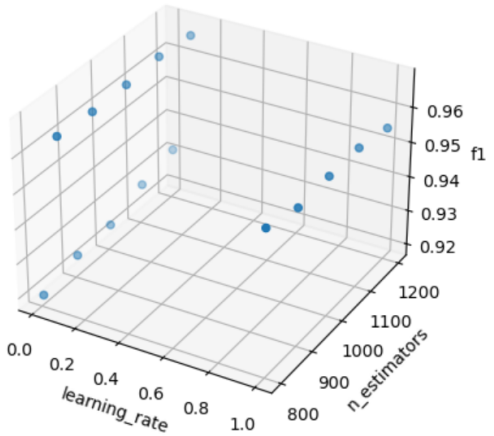


Figure 8: Adaboost Grid Search Result

Deep Neural Network For DNN, we does not use Lasso to select features, because the features will be multiply together in the network.

Table 2: **DNN Training Details**

Epoch e	[40, 60, 80]
Learning rate r	[0.0001, 0.001, 0.01, 0.1]
Batch size b	128
Decay step	[20, 40]
Decay rate	0.1
Number of hidden layer	[3, 5, 7]
Number of neurons	[50, 70, 100, 200]

From huge number of experiments (part of the results shown in the Figure 9), we finally choose $epoch = 80$, $learn\_rate = 0.01$, $decay\_step = 20$, $Number\_of\_hidden\_layer = 3$ and $Number\_of\_neurons = 100$. The training results with these parameters are shown in Figure 10.

	50 neurons	70 neurons	100 neurons	200 neurons
3 layers	0.0643	0.0660	0.0630	0.0641
5 layers	0.0652	0.0642	0.0667	0.4703
7 layers	0.0657	0.0649	0.0680	0.4703

Figure 9: DNN Grid Search Result

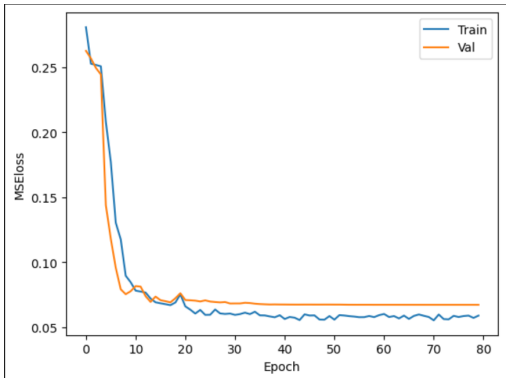


Figure 10: DNN Training Result

5 Data Splits

We use 5-fold CV to evaluate the Adaboost model's performance and prevent overfitting. The results shown in the Figure prove our model is robust. Also, we did not use the whole

training set to train the model, while split 10% samples as test set. The results are also shown here.

	Train					Test
CV	1	2	3	4	5	
F1 Score	0.87	0.93	0.94	0.95	0.94	0.69

Figure 11: Adaboost CV Result

6 Errors and Mistakes

The hardest part is feature engineering, because as a student who doesn't have data science background, I need to study from beginning. As for the mistakes, at the beginning of the project, I used a inner function in *pandas* to encode the training data and the test data respectively, which leads to the encoding rules are different in training and testing, so all models performed worse and it takes a long time for me to find this bug.

7 Predictive Accuracy

Kaggle Name: Shucheng Zhang

The adaboost model with Lasso got a 0.68421 in both private score and public score. However, surprisingly, the adaboost model with label encoding show a great performance: 0.63157 in public score and 0.82051 in private score. The DNN model show 0.56521 in public score and 0.64 in private score. The training result of adaboost is shown in Figure 11 and the training result of DNN is shown in Figure 10.

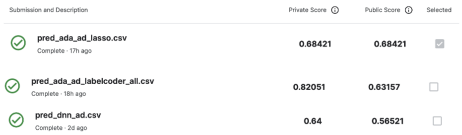


Figure 12: Kaggle Result

8 Reference

[1] https://www.kaggle.com/learn/feature-engineering

[2] https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

[3] https://www.knime.com/blog/predicting-employee-attrition-with-machine-learning

[4] https://www.enjoyalgorithms.com/blog/attrition-rate-prediction-using-ml

[5] Schapire, Robert E. "Explaining adaboost." Empirical inference. Springer, Berlin, Heidelberg, 2013. 37-52.

[6] Al-Darraji, Salah, et al. "Employee Attrition Prediction Using Deep Neural Networks." Computers 10.11 (2021): 141.

[7] He, H.; Bai, Y.; Garcia, E.A.; Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008; pp. 1322–1328.

[8] Fonti, Valeria, and Eduard Belitser. "Feature selection using lasso." VU Amsterdam research paper in business analytics 30 (2017): 1-25.

9 Code

Shown in next pages.

ada_lasso

December 10, 2022

Adaboost with Lasso

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
# Feature Engineering
data_train = pd.read_csv('train.csv')
data_train_len = data_train.shape[0]
data_test = pd.read_csv('test.csv')
y = data_train.pop('Attrition')
data = pd.concat([data_train, data_test])


data = data.drop(columns=['EmployeeID', 'StandardHours', 'Over18',␣
 ↪'EmployeeCount'])
data.replace({'JobRole':{'Admin': 'Administrative'}}, inplace=True)

# Encoding
# for colname in data.select_dtypes("O"):
#     data[colname], _ = data[colname].factorize()
data.replace({'Education':{1:'Below_college', 2:'College', 3:'Bachelor', 4:
 ↪'Masters', 5:'Doctor'}}, inplace=True)
data = pd.get_dummies(data)
data = data.drop(columns=['Gender_Female','OverTime_No'])

# Multiply features together
data_mult = pd.DataFrame()

for i in range(data.shape[1]):
    for j in range(data.shape[1]):
        data_mult = pd.concat([data_mult, data.iloc[:,i]* data.iloc[:,j]],␣
 ↪axis=1)

# Save pre-processing data
data_mult.to_csv('mult.csv', index = 0)
```

[2]:
```
# Load Data
data_mult = pd.read_csv('mult.csv')
data_train = pd.read_csv('train.csv')
data_train_len = data_train.shape[0]
y = data_train.pop('Attrition')

train = data_mult.iloc[:data_train_len, :].to_numpy()
test = data_mult.iloc[data_train_len:, :].to_numpy()
train.shape, test.shape
```

[2]:
```
((1340, 2304), (336, 2304))
```

[3]:
```
# Encode y
def y_enco(y):
    if y == 'No':
        return 0
    elif y == 'Yes':
        return 1
    else:
        pass

y_encoder = y.apply(y_enco)
```

[5]:
```
from imblearn.over_sampling import ADASYN
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(train, y_encoder,␣
 ↪test_size=0.1, random_state=42)

# Data Normalization
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X_train)
X_train_norm = scaler.transform(X_train)

# Oversampling
ada = ADASYN(random_state=42)
X_res, y_res = ada.fit_resample(X_train_norm, y_train)
X_res.shape, y_res.shape
```

[5]:
```
((2115, 2304), (2115,))
```

[6]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

# Lasso feature selection
```

```
select = SelectFromModel(LogisticRegression(C=1, penalty='l1',␣
 ↪solver='liblinear'))
select.fit(X_res, y_res)
X_new = select.transform(X_res)
X_new.shape
```

[6]:
```
(2115, 309)
```

[11]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import RandomForestClassifier

# Defining parameters for hyper-parameter tuning
params = {
        'n_estimators': [int(x) for x in np.linspace(start = 800, stop = 1200,␣
 ↪num = 5)],
        'learning_rate': [0.01, 0.1, 1.0],

}

# Initializing Grid Search with Adaboost and keeping f1 as the performance␣
 ↪metrics
grid_search = GridSearchCV(estimator = AdaBoostClassifier(),
                           param_grid=params,
                           cv = 10,
                           n_jobs=-1,
                           verbose=0,
                           scoring="f1",
                           return_train_score=True)

# Training
grid_search.fit(X_new, y_res)

# Best Performing Parameter
print('='*20)
print("best params: " + str(grid_search.best_estimator_))
print("best params: " + str(grid_search.best_params_))
print('best score:', grid_search.best_score_)
print('='*20)
```

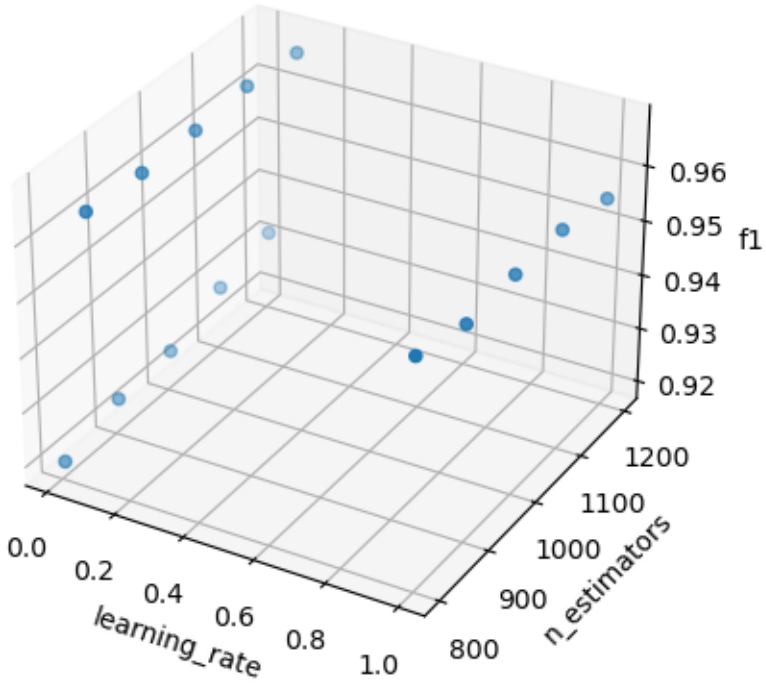
```
====================
best params: AdaBoostClassifier(learning_rate=0.1, n_estimators=1100)
best params: {'learning_rate': 0.1, 'n_estimators': 1100}
best score: 0.9674340129254417
====================
```

[12]:
```
# All performance
means = grid_search.cv_results_['mean_test_score']
stds = grid_search.cv_results_['std_test_score']
params = grid_search.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
0.920094 (0.040531) with: {'learning_rate': 0.01, 'n_estimators': 800}
0.923357 (0.039975) with: {'learning_rate': 0.01, 'n_estimators': 900}
0.924200 (0.042850) with: {'learning_rate': 0.01, 'n_estimators': 1000}
0.928343 (0.040455) with: {'learning_rate': 0.01, 'n_estimators': 1100}
0.931228 (0.038760) with: {'learning_rate': 0.01, 'n_estimators': 1200}
0.966858 (0.046052) with: {'learning_rate': 0.1, 'n_estimators': 800}
0.966219 (0.046464) with: {'learning_rate': 0.1, 'n_estimators': 900}
0.966521 (0.042641) with: {'learning_rate': 0.1, 'n_estimators': 1000}
0.967434 (0.042898) with: {'learning_rate': 0.1, 'n_estimators': 1100}
0.966644 (0.046885) with: {'learning_rate': 0.1, 'n_estimators': 1200}
0.958507 (0.062317) with: {'learning_rate': 1.0, 'n_estimators': 800}
0.955887 (0.069068) with: {'learning_rate': 1.0, 'n_estimators': 900}
0.956661 (0.066830) with: {'learning_rate': 1.0, 'n_estimators': 1000}
0.956894 (0.064679) with: {'learning_rate': 1.0, 'n_estimators': 1100}
0.954989 (0.068783) with: {'learning_rate': 1.0, 'n_estimators': 1200}
```

[21]:
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# plot grid search
lr = []
es = []
f1 = []
for mean, stdev, param in zip(means, stds, params):
    lr.append(param['learning_rate'])
    es.append(param['n_estimators'])
    f1.append(mean)
fig = plt.figure()
ax1 = plt.axes(projection='3d')
ax1.scatter3D(lr,es,f1, cmap='Blues')
ax1.set_xlabel('learning_rate')
ax1.set_ylabel('n_estimators')
ax1.set_zlabel('f1')
```

[21]:
```
Text(0.5, 0, 'f1')
```



[25]:
```
from sklearn.model_selection import cross_val_score

# Cross validation final model
model =  AdaBoostClassifier(n_estimators=1100, learning_rate=0.01)
cross_val_score(model, X_new, y_res, cv=5)
```

[25]:
```
array([0.87234043, 0.93617021, 0.94799054, 0.94799054, 0.94799054])
```

[8]:
```
from sklearn.metrics import f1_score

# Test in val set
X_val_norm = scaler.transform(X_test)
X_val_new = select.transform(X_val_norm)

y_pred = grid_search.predict(X_val_new)

f1_score(y_test, y_pred)
```

[8]:
```
0.6896551724137931
```

[8]:
```
# Predict and save
X_test_norm = scaler.transform(test)
X_test_new = select.transform(X_test_norm)
```

```
y_pred = grid_search.predict(X_test_new)

id = range(0,len(y_pred))
y_test = pd.DataFrame()
y_test['Id'] = id
y_test['Predicted'] = y_pred
y_test.to_csv('pred_ada_ad_lasso_rf.csv', index = 0)
```

dnn

December 10, 2022

DNN

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data.dataset import random_split, TensorDataset
from torchvision import datasets
from sklearn.metrics import confusion_matrix

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data_train = pd.read_csv('train.csv')
data_train_len = data_train.shape[0]
data_test = pd.read_csv('test.csv')
y = data_train.pop('Attrition')
data = pd.concat([data_train, data_test])

data = data.drop(columns=['EmployeeID', 'StandardHours', 'Over18',␣
 ↪'EmployeeCount'])
data.replace({'JobRole':{'Admin': 'Administrative'}}, inplace=True)
# for colname in data.select_dtypes("O"):
#     data[colname], _ = data[colname].factorize()
data.replace({'Education':{1:'Below_college', 2:'College', 3:'Bachelor', 4:
 ↪'Masters', 5:'Doctor'}}, inplace=True)
data = pd.get_dummies(data)
data = data.drop(columns=['Gender_Female','OverTime_No'])

train = data.iloc[:data_train_len, :]
test = data.iloc[data_train_len:, :]
```

[3]:
```
def y_enco(y):
    if y == 'No':
        return 0
    elif y == 'Yes':
        return 1
    else:
        pass

y_encoder = y.apply(y_enco)
```

[4]:
```
from imblearn.over_sampling import ADASYN
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# Data Normalization

scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(train)
X_norm = scaler.transform(train)

ada = ADASYN(random_state=42)
X_res, y_res = ada.fit_resample(X_norm, y_encoder)
X_res.shape

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.1,␣
 ↪random_state=42)
X_train.shape
# ada = ADASYN(random_state=42)
# X_res, y_res = ada.fit_resample(X_train, y_train)
# X_res.shape, y_res.shape
```

[4]:
```
(2116, 48)
```

[5]:
```
X_data = torch.tensor(X_train, dtype = torch.float)
y_data = torch.tensor(y_train.to_numpy().reshape((-1,1)), dtype = torch.float)

dataset = TensorDataset(X_data, y_data)

# train_set, val_set = random_split(dataset, [1071, 269])
train_set, val_set = random_split(dataset, [1500, 616])

train_loader = torch.utils.data.DataLoader(train_set, batch_size=128,␣
 ↪shuffle=True)
val_loader = torch.utils.data.DataLoader(val_set, batch_size=128, shuffle=False)
```

[35]:
```
class Net(nn.Module):
    def __init__(self, s):
        super(Net, self).__init__()
```

```
        self.fc1 = nn.Linear(48, s)
        self.fc2 = nn.Linear(s, s)
        self.fc3 = nn.Linear(s, s)
        self.fc4 = nn.Linear(s, s)
        self.fc5 = nn.Linear(s, s)
        self.fc6 = nn.Linear(s, s)
        self.fc7 = nn.Linear(s, 1)
        self.dropout = nn.Dropout(p=0.2)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = self.dropout(x)
        x = F.softplus(self.fc2(x))
        x = self.dropout(x)
        x = F.softplus(self.fc3(x))
        # x = self.dropout(x)
        # x = F.softplus(self.fc4(x))
        # x = self.dropout(x)
        # x = F.softplus(self.fc5(x))
        # x = self.dropout(x)
        # x = F.softplus(self.fc6(x))
        # x = self.dropout(x)
        x = torch.sigmoid(self.fc7(x))
        return x
```

[36]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
if device =='cuda':
    print("Train on GPU...")
else:
    print("Train on CPU...")
max_epochs = 80
l = []
```

```
Train on GPU...
```

[37]:
```
s_loss_5 = []
for s in [50, 60, 70, 80, 90, 100, 200]:
    loss_list, loss_list_val = [], []

    criterion = nn.MSELoss()
    net = Net(s).to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.01)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.1)

    for epoch in range(max_epochs):
        net.train()
        epoch_loss = 0.0
        for batch_idx, (data, labels) in enumerate(train_loader):
            data, labels = data.to(device), labels.to(device)
```

```
            optimizer.zero_grad()
            output = net(data)
            loss = criterion(output, labels)

            loss.backward()
            optimizer.step()

            epoch_loss += loss

        avg_loss = epoch_loss/(batch_idx+1)
        loss_list.append(avg_loss.cpu().detach().numpy())

        # validation
        net.eval()
        with torch.no_grad():
            loss_val = 0.0
            for batch_idx, (data, labels) in enumerate(val_loader):
                data, labels = data.to(device), labels.to(device)
                output = net(data)
                loss = criterion(output, labels)

                loss_val += loss

            avg_loss_val = loss_val/(batch_idx+1)

        loss_list_val.append(avg_loss_val.cpu().detach().numpy())
        scheduler.step()

        #print('[epoch %d] loss: %.5f val loss: %.5f' % (epoch + 1, avg_loss,
↪avg_loss_val))

    s_loss_5.append(avg_loss_val.cpu().detach().numpy())

    print(avg_loss_val.cpu().detach().numpy())
```

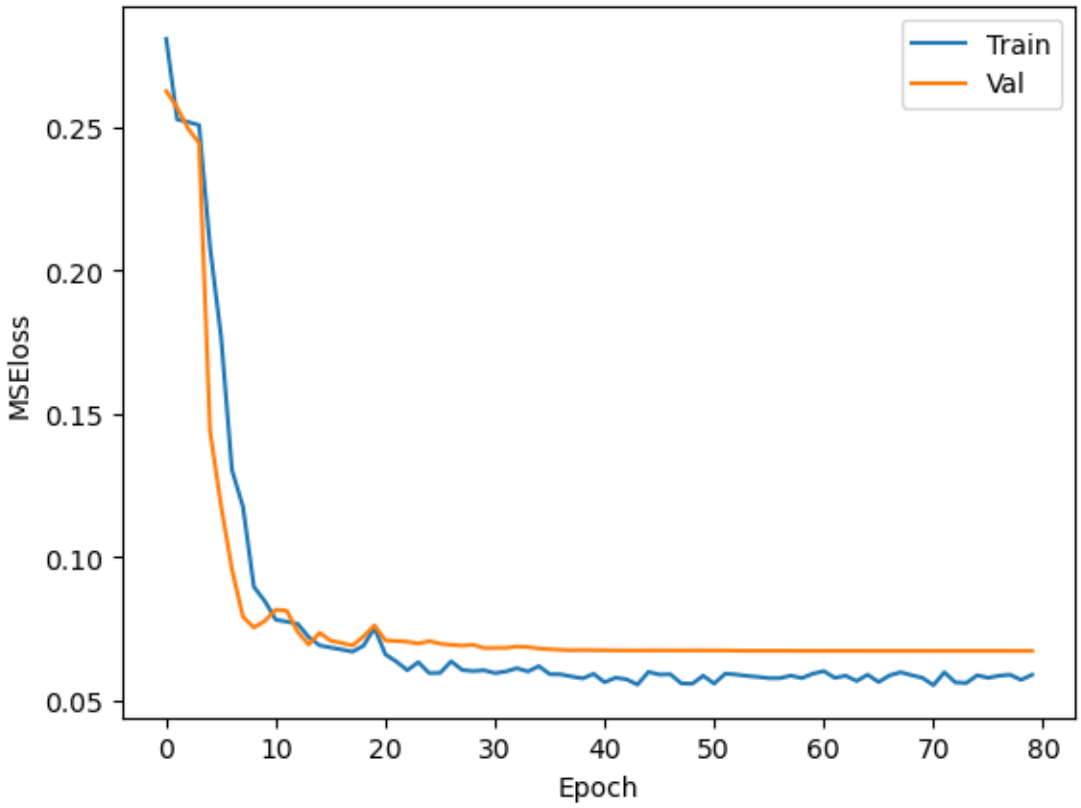
```
0.06570879
0.06491885
0.066951685
0.06636639
0.06801868
0.25045785
0.4703125
```

[38]:
```
# Part Grid Search Result
s_loss_3, s_loss_4, s_loss_5
```

[38]:
```
([array(0.06429266, dtype=float32),
  array(0.06599749, dtype=float32),
  array(0.06528779, dtype=float32),
  array(0.06416414, dtype=float32),
  array(0.06299269, dtype=float32),
  array(0.06324429, dtype=float32),
  array(0.06417646, dtype=float32)],
 [array(0.06520052, dtype=float32),
  array(0.06420612, dtype=float32),
  array(0.06402928, dtype=float32),
  array(0.06424574, dtype=float32),
  array(0.06671084, dtype=float32),
  array(0.06696578, dtype=float32),
  array(0.4703125, dtype=float32)],
 [array(0.06570879, dtype=float32),
  array(0.06491885, dtype=float32),
  array(0.06695168, dtype=float32),
  array(0.06636639, dtype=float32),
  array(0.06801868, dtype=float32),
  array(0.25045785, dtype=float32),
  array(0.4703125, dtype=float32)])
```

[13]:
```
# use 3 hidden layers, 100 size
plt.plot(loss_list, label = 'Train')
plt.plot(loss_list_val, label = 'Val')
plt.xlabel('Epoch')
plt.ylabel('MSEloss')
plt.legend()
plt.plot()
```

[13]:
```
[]
```



[14]:
```
# Predict test set
scaler.fit(test)
X_test_scaler = scaler.transform(test)
X_test_scaler = torch.tensor(X_test_scaler, dtype=torch.float).to(device)
X_test_scaler.shape
```

[14]:
```
torch.Size([336, 48])
```

[15]:
```
y_pred = net(X_test_scaler).cpu().detach().numpy()
y_pred_show = np.zeros((y_pred.shape), dtype = int)
for i in range(y_pred.shape[0]):
    if y_pred[i] >= 0.5:
        y_pred_show[i] = 1
    else:
        y_pred_show[i] = 0
```

[132]:
```
id = range(0,len(y_pred_show))
y_test = pd.DataFrame()
y_test['Id'] = id
y_test['Predicted'] = y_pred_show
y_test.to_csv('pred_dnn_ad.csv', index = 0)
```

plot_figure

December 10, 2022

Feature Engineering

[181]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[199]:
```
df = pd.read_csv('train.csv')
df.head()
```

[199]:

	EmployeeID	Age	Attrition	BusinessTravel	DailyRate	Department	\
0	1317087	40	No	Travel_Rarely	1398	Cardiology	
1	1548175	40	No	Travel_Rarely	1300	Maternity	
2	1215433	25	No	Travel_Rarely	622	Cardiology	
3	1375351	33	No	Travel_Rarely	922	Maternity	
4	1028734	39	No	Travel_Frequently	505	Maternity	

	DistanceFromHome	Education	EducationField	EmployeeCount	...	\
0	2	4	Life Sciences	1	...	
1	24	2	Technical Degree	1	...	
2	13	1	Medical	1	...	
3	1	5	Medical	1	...	
4	2	4	Technical Degree	1	...	

	RelationshipSatisfaction	StandardHours	Shift	TotalWorkingYears	\
0	4	80	0	21	
1	1	80	2	9	
2	3	80	0	7	
3	3	80	1	10	
4	4	80	0	20	

	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	YearsInCurrentRole	\
0	2	3	20	15	
1	3	3	9	8	
2	1	3	7	4	
3	2	3	6	1	
4	1	3	19	6	

```
   YearsSinceLastPromotion  YearsWithCurrManager
0                        1                    12
1                        4                     7
2                        0                     6
3                        0                     5
4                       11                     8

[5 rows x 35 columns]
```

[179]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   EmployeeID                1340 non-null   int64
 1   Age                       1340 non-null   int64
 2   Attrition                 1340 non-null   object
 3   BusinessTravel            1340 non-null   object
 4   DailyRate                 1340 non-null   int64
 5   Department                1340 non-null   object
 6   DistanceFromHome          1340 non-null   int64
 7   Education                 1340 non-null   int64
 8   EducationField            1340 non-null   object
 9   EmployeeCount             1340 non-null   int64
 10  EnvironmentSatisfaction   1340 non-null   int64
 11  Gender                    1340 non-null   object
 12  HourlyRate                1340 non-null   int64
 13  JobInvolvement            1340 non-null   int64
 14  JobLevel                  1340 non-null   int64
 15  JobRole                   1340 non-null   object
 16  JobSatisfaction           1340 non-null   int64
 17  MaritalStatus             1340 non-null   object
 18  MonthlyIncome             1340 non-null   int64
 19  MonthlyRate               1340 non-null   int64
 20  NumCompaniesWorked        1340 non-null   int64
 21  Over18                    1340 non-null   object
 22  OverTime                  1340 non-null   object
 23  PercentSalaryHike         1340 non-null   int64
 24  PerformanceRating         1340 non-null   int64
 25  RelationshipSatisfaction  1340 non-null   int64
 26  StandardHours             1340 non-null   int64
 27  Shift                     1340 non-null   int64
 28  TotalWorkingYears         1340 non-null   int64
 29  TrainingTimesLastYear     1340 non-null   int64
 30  WorkLifeBalance           1340 non-null   int64
 31  YearsAtCompany            1340 non-null   int64
```

```
 32  YearsInCurrentRole         1340 non-null   int64
 33  YearsSinceLastPromotion    1340 non-null   int64
 34  YearsWithCurrManager       1340 non-null   int64
dtypes: int64(26), object(9)
memory usage: 366.5+ KB
```

[180]:
```
X = pd.read_csv('train.csv')

# Label encoding for categoricals
for colname in X.select_dtypes("O"):
    X[colname], _ = X[colname].factorize()

# All discrete features should now have integer dtypes (double-check this before␣
 ↪using MI!)
discrete_features = X.dtypes == int
X.hist(figsize=(15,15))
plt.tight_layout()
plt.show()
```

[182]:
```
df.drop(['EmployeeCount','EmployeeID', 'StandardHours'],axis=1,inplace=True)
df.drop('Over18',axis=1,inplace=True)

df.replace({'JobRole':{'Admin': 'Administrative'}}, inplace=True)
df.replace({'Education':{1:'Below_college', 2:'College', 3:'Bachelor', 4:
 ↪'Masters', 5:'Doctor'}}, inplace=True)
low_col = []
for i in df.columns:
    i = i.lower()
    low_col.append(i)
df.columns = low_col
```

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

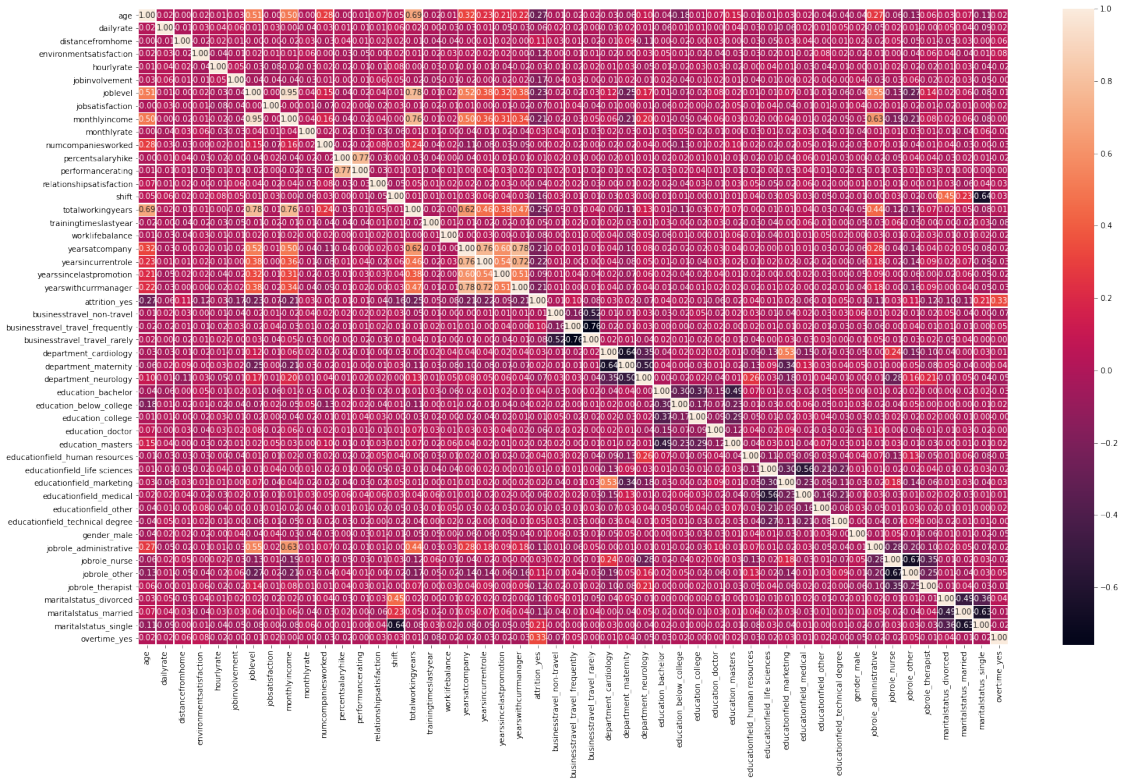
[183]:
```
df_dum.drop(['gender_Female','overtime_No','attrition_No'],axis=1,inplace=True)
low_col = []
for i in df_dum.columns:
    i = i.lower()
    low_col.append(i)
df_dum.columns = low_col
df_dum.columns
```

[183]:
```
Index(['age', 'dailyrate', 'distancefromhome', 'environmentsatisfaction',
       'hourlyrate', 'jobinvolvement', 'joblevel', 'jobsatisfaction',
       'monthlyincome', 'monthlyrate', 'numcompaniesworked',
       'percentsalaryhike', 'performancerating', 'relationshipsatisfaction',
       'shift', 'totalworkingyears', 'trainingtimeslastyear',
       'worklifebalance', 'yearsatcompany', 'yearsincurrentrole',
       'yearssincelastpromotion', 'yearswithcurrmanager', 'attrition_yes',
       'businesstravel_non-travel', 'businesstravel_travel_frequently',
       'businesstravel_travel_rarely', 'department_cardiology',
       'department_maternity', 'department_neurology', 'education_bachelor',
       'education_below_college', 'education_college', 'education_doctor',
       'education_masters', 'educationfield_human resources',
       'educationfield_life sciences', 'educationfield_marketing',
       'educationfield_medical', 'educationfield_other',
       'educationfield_technical degree', 'gender_male',
       'jobrole_administrative', 'jobrole_nurse', 'jobrole_other',
       'jobrole_therapist', 'maritalstatus_divorced', 'maritalstatus_married',
       'maritalstatus_single', 'overtime_yes'],
      dtype='object')
```

[184]:
```
df_dum.shape
```

[184]:
```
(1340, 49)
```

[185]:
```
corr_matrix = df_dum.corr()
fig,ax=plt.subplots(figsize=(25,15))
ax=sns.heatmap(corr_matrix,
               annot=True,
               linewidths=0.5,
               fmt=".2f"
              )
```

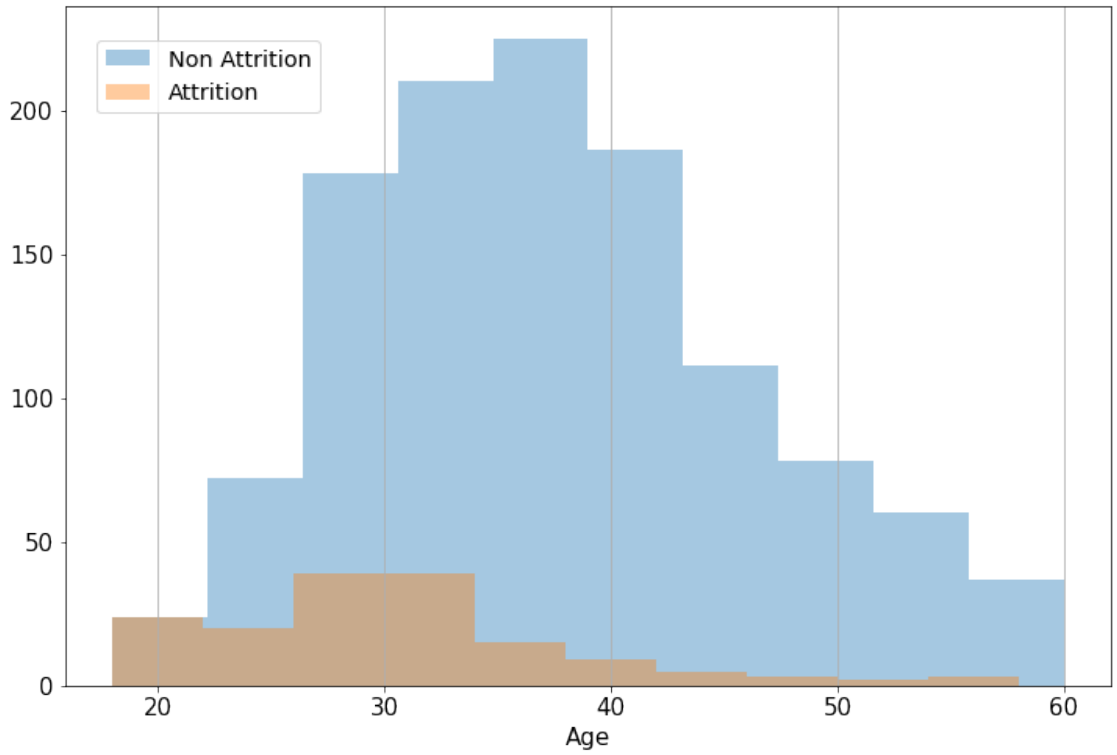


[186]:
```
df.loc[df.attrition == 'Yes', 'attrition'] = 1
df.loc[df.attrition == 'No', 'attrition'] = 0
```

[187]:
```
""" Plot the distribution of Age where attrition is true and false. """

plt.figure(figsize=(12,8))

""" Adjusting the bin size can alter the look of your graph, worth testing␣
 ↪different sizes to see various plots. """
fig = sns.distplot(df[df['attrition'] == 0]['age'], label='Non Attrition',␣
 ↪kde=0, bins=10)
sns.distplot(df[df['attrition'] == 1]['age'], label='Attrition', kde=0, bins=10)

#sns.despine(left=1)

""" Removes the vertical gridlines. """
fig.grid(axis='x')

plt.xlabel('Age',fontsize=15)
fig.yaxis.labelpad = 30
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

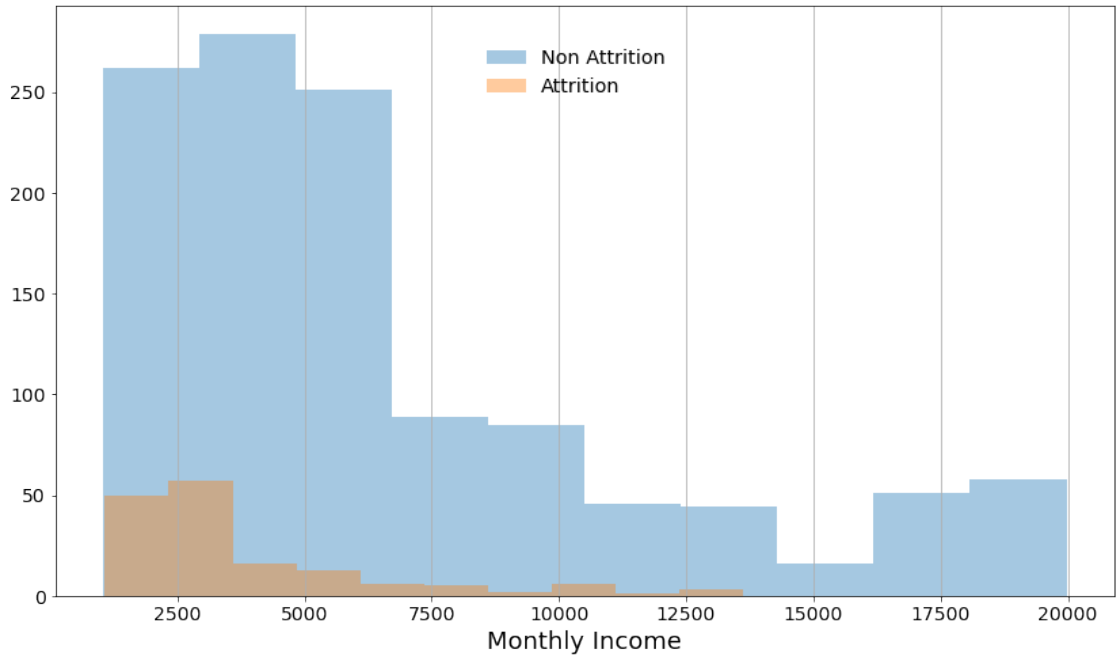
```
fig.yaxis.labelpad = 35

""" Control the size and positioning of the legend. """
plt.legend(fontsize='x-large', bbox_to_anchor=(0.03, 0.95), loc=2,␣
 ↪borderaxespad=0., frameon=1)
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



[188]:
```
plt.figure(figsize=(14,8))
fig = sns.distplot(df[df['attrition'] == 0]['monthlyincome'], label='Non␣
 ↪Attrition', kde=0, bins=10)
sns.distplot(df[df['attrition'] == 1]['monthlyincome'], label='Attrition',␣
 ↪kde=0, bins=10)
fig.grid(axis='x')
plt.xlabel('Monthly Income',fontsize=18)
fig.yaxis.labelpad = 30
plt.xticks(fontsize=14)
```

```
plt.yticks(fontsize=14)
fig.yaxis.labelpad = 35
plt.legend(fontsize='x-large', bbox_to_anchor=(0.4, 0.94), loc=2,␣
 ↪borderaxespad=0., frameon=0)
plt.show()
print('Average Monthly Income:',df.monthlyincome.mean())
print('Average Monthly Income for Males:',df[df.gender ==␣
 ↪'Male']['monthlyincome'].mean())
print('Average Monthly Income for Females:',df[df.gender ==␣
 ↪'Female']['monthlyincome'].mean())
```




```
Average Monthly Income: 6433.381343283582
Average Monthly Income for Males: 6334.13567839196
Average Monthly Income for Females: 6578.601102941177
```

[189]:
```
income = df.groupby(by='jobrole').mean().monthlyincome
inc = pd.DataFrame(income)
inc = inc.sort_values(by='monthlyincome')
inc
```

[189]:

jobrole	monthlyincome
Other	5006.112903
Nurse	5519.843318
Therapist	7517.263158
Administrative	16621.708738

[190]:
```
education = df.groupby(by='education').mean().monthlyincome
edu = pd.DataFrame(education)
edu = edu.sort_values(by='monthlyincome')
edu
```

[190]:

education	monthlyincome
Below_college	5783.493827
College	6071.212500
Bachelor	6505.622605
Masters	6650.588076
Doctor	8015.148936

[191]:
```
shift = df.groupby(by='shift').mean().monthlyincome
shi = pd.DataFrame(shift)
shi = shi.sort_values(by='monthlyincome')
shi
```

[191]:

shift	monthlyincome
3	5781.687500
0	6197.030142
2	6242.476190
1	6822.271403

[192]:
```
totalworkingyears = df.groupby(by='totalworkingyears').mean().monthlyincome
twy = pd.DataFrame(totalworkingyears)
twy = twy.sort_values(by='monthlyincome')
```

[193]:
```
yearsincurrentrole = df.groupby(by='yearsincurrentrole').mean().monthlyincome
ycr = pd.DataFrame(yearsincurrentrole)
ycr = ycr.sort_values(by='monthlyincome')
```

[194]:
```
yearsatcompany = df.groupby(by='yearsatcompany').mean().monthlyincome
ysc = pd.DataFrame(yearsatcompany)
ysc = ysc.sort_values(by='monthlyincome')
```

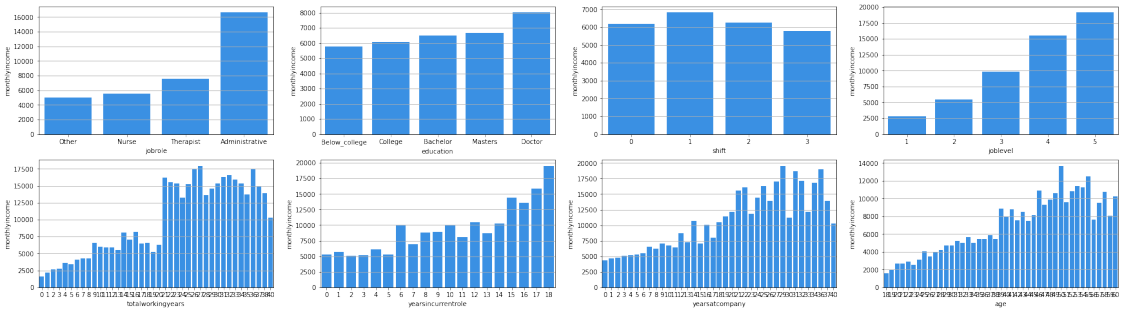
[195]:
```
level = df.groupby(by='joblevel').mean().monthlyincome
lev = pd.DataFrame(level)
lev = lev.sort_values(by='monthlyincome')
lev
```

[195]:

joblevel	monthlyincome
1	2771.292829
2	5472.848361
3	9783.963158
4	15473.131313

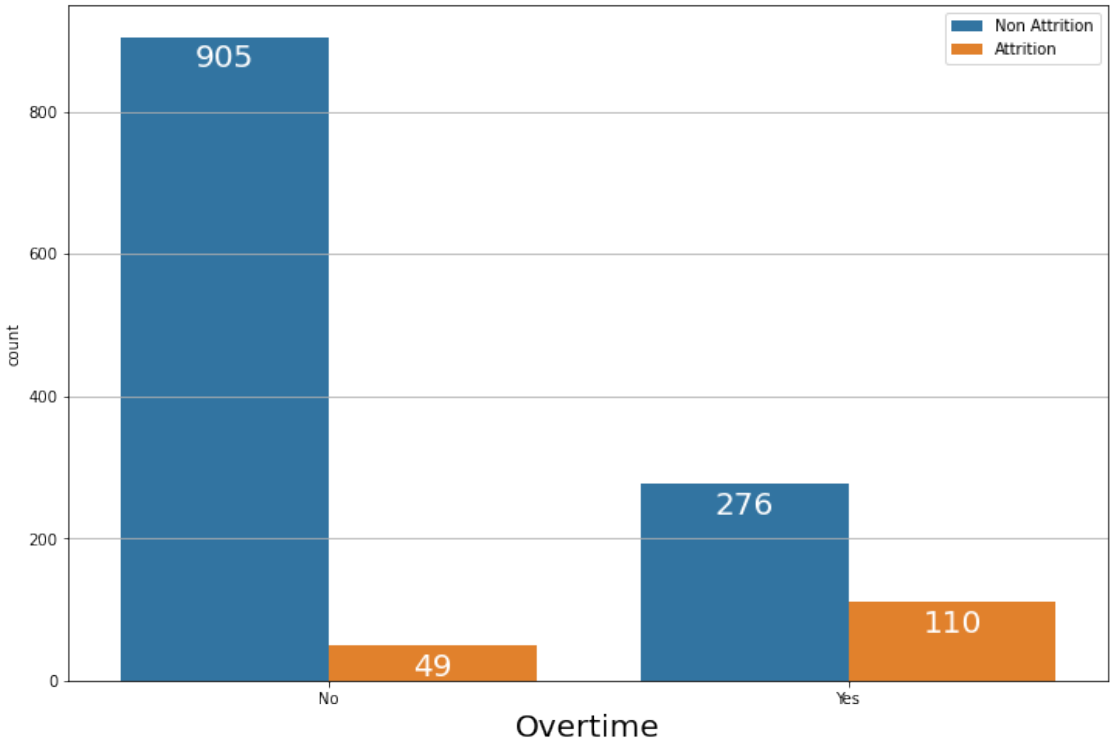
```
5          19147.524590
```

[196]:
```
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(30,8))
axes[0][0] = sns.barplot(y='monthlyincome', x=inc.index, data=inc,␣
 ↪color='dodgerblue', ax = axes[0][0])
axes[0][1] = sns.barplot(y='monthlyincome', x=edu.index, data=edu,␣
 ↪color='dodgerblue', ax = axes[0][1])
axes[0][2] = sns.barplot(y='monthlyincome', x=shi.index, data=shi,␣
 ↪color='dodgerblue', ax = axes[0][2])
axes[0][3] = sns.barplot(y='monthlyincome', x=lev.index, data=lev,␣
 ↪color='dodgerblue', ax = axes[0][3])
axes[1][0] = sns.barplot(y='monthlyincome', x=twy.index, data=twy,␣
 ↪color='dodgerblue', ax = axes[1][0])
axes[1][1] = sns.barplot(y='monthlyincome', x=ycr.index, data=ycr,␣
 ↪color='dodgerblue', ax = axes[1][1])
axes[1][2] = sns.barplot(y='monthlyincome', x=ysc.index, data=ysc,␣
 ↪color='dodgerblue', ax = axes[1][2])
axes[1][3] = sns.barplot(y='monthlyincome', x=age.index, data=age,␣
 ↪color='dodgerblue', ax = axes[1][3])

axes[0][0].grid(axis='y')
axes[0][1].grid(axis='y')
axes[0][2].grid(axis='y')
axes[0][3].grid(axis='y')
axes[1][0].grid(axis='y')
axes[1][1].grid(axis='y')
axes[1][2].grid(axis='y')
axes[1][3].grid(axis='y')
plt.show()
```



[197]:
```
plt.figure(figsize=(12,8))
fig = sns.countplot(x='overtime', hue='attrition', data=df)
fig.set_xlabel('Overtime', fontsize=20)
fig.grid(axis='y')
```

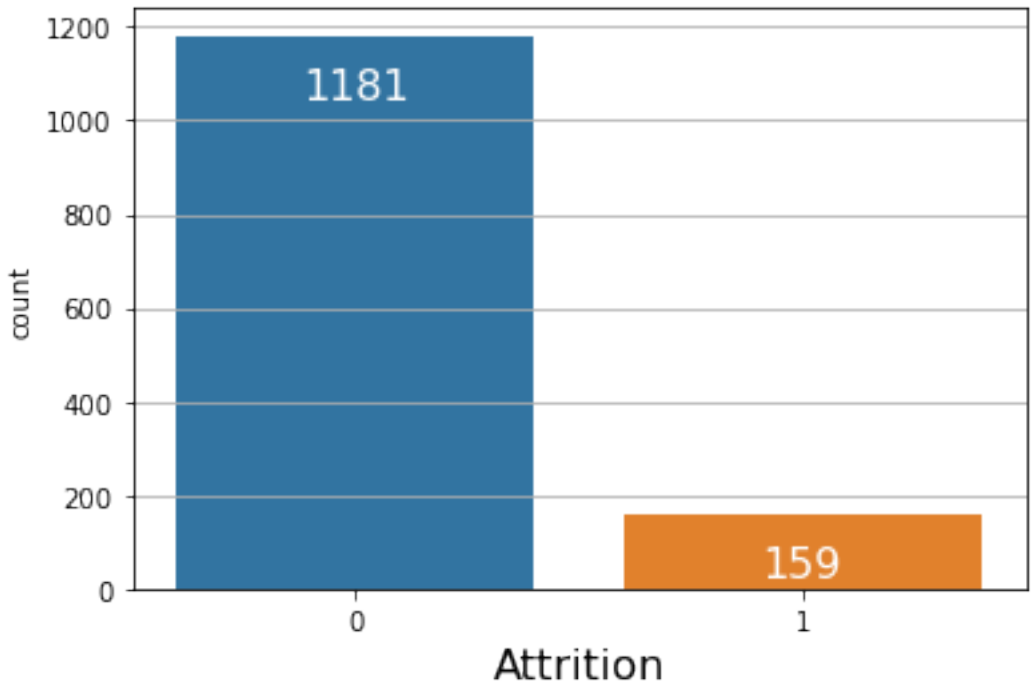
```
for p in fig.patches:
    x=p.get_bbox().get_points()[:,0]
    y=p.get_bbox().get_points()[1,1]
    fig.annotate('{:}'.format(p.get_height()), (x.mean(), y-50), ha='center',␣
 ↪va='bottom', fontsize=20, color='white')
plt.legend(labels =['Non Attrition','Attrition'])
plt.show()
```



[198]:
```
df.loc[df.attrition == 'Yes', 'attrition'] = 1
df.loc[df.attrition == 'No', 'attrition'] = 0

""" Set the figure size. """
plt.figure(figsize=(6,4))

fig = sns.countplot(df.attrition)

fig.grid(axis='y')

fig.set_xlabel('Attrition', fontsize=16)


for p in fig.patches:
```

```
    x=p.get_bbox().get_points()[:,0]
    y=p.get_bbox().get_points()[1,1]
    fig.annotate('{:}'.format(p.get_height()), (x.mean(), y-150), ha='center',␣
 ↪va='bottom', fontsize=16, color='white')
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[ ]: