Forecasting Employee Attrition in Healthcare Companies

Yeon-woo Shin
Introduction to Machine Learning Using Python

I. Introduction

As people become more sensitive to a variety of issues of a company involving work-life balance, job satisfaction, promotion period nowadays, people are more likely to leave their current job and find the company that better suits them. This trend puts hiring managers under great pressure to hire the right employees who wouldn't leave the company. It already takes a hiring manager a great effort to do interviews with candidates and elaborately investigate the resumes of every applicant to find a single perfect candidate for the company. However, if an employee attrites the company soon after being hired, a hiring manager has to take another tiresome process of hiring another employee, leading to work inefficiency. A forecasting model for employee attrition can notify a hiring manager of a candidate who would stay long-term, thereby improving work efficiency of a hiring manager. The proposed forecasting model focused on attrition of employees in Healthcare Companies and was built based on a dataset from https://www.kaggle.com/datasets/jpmiller/employee-attrition-for-healthcare.

II. Module requirements

scikit-learn, imbalanced-learn (sci-kit version 1.1.0 minimum)

III. Program Description

First, the healthcare employee attrition dataset was imported to the notebook using read_csv, and categorical variables and number variables were separately grouped to be later used in feature engineering. To explore the dataset, age and monthly income distributions each for employees who attrited and who didn't were compared. Also, the Bayesian probability theorem was used to compare the likelihood of attribution based on work departments. To change the categorical variables into numerical, cat.codes() method was used. To normalize the number variables, a standard scaler from sklearn library was used. Since 35 columns of a dataset may possibly cause overfitting problems, PCA was used to reduce the number of features to 24. At last, because the number of data from employees who attrited is relatively deficient compared to the number of data from employees who didn't attrite, SMOTE(Synthetic Minority Oversampling Technique) was used to solve attrition class imbalance. Before applying SMOTE, the number of data for attrition and for non-attrition were respectively 1477 and 199. After applying SMOTE, the number of data for

attrition and for non-attrition were respectively 1477 and 1477. Supervised classification models such as logistic regression, support vector classifier, and random forest were built to predict the employee attrition.

IV. Python Codes & Cell Outputs

Data Preparation

Import dataset in csv format

In [4]:
```
attr = pd.read_csv(r"C:\Users\A\Desktop\employee_attr.csv")
```

Print the first five rows of dataset

In [5]:
```
attr.head(5)
```

Out[5]:

	EmployeeID	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	...	RelationshipSatisfac
0	1313919	41	No	Travel_Rarely	1102	Cardiology	1	2	Life Sciences	1	...	
1	1200302	49	No	Travel_Frequently	279	Maternity	8	1	Life Sciences	1	...	
2	1060315	37	Yes	Travel_Rarely	1373	Maternity	2	2	Other	1	...	
3	1272912	33	No	Travel_Frequently	1392	Maternity	3	4	Life Sciences	1	...	
4	1414939	27	No	Travel_Rarely	591	Maternity	2	1	Medical	1	...	

5 rows × 35 columns

Count the number of categorical variables(dtype='object') and numerical variables(dtype=int64)

In [6]:
```
print(attr.dtypes.value_counts())
```

```
int64     26
object     9
dtype: int64
```

Group the features based on whether it is categorical or numerical

In [7]:
```
attr.columns.to_series().groupby(attr.dtypes).groups
```

Out[7]:
```
{int64: ['EmployeeID', 'Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount', 'EnvironmentSatisfaction', 'Hourly
Rate', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHik
e', 'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours', 'Shift', 'TotalWorkingYears', 'TrainingTimesLastYear', 'W
orkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager'], object: ['Attritio
n', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']}
```

Data Visualization

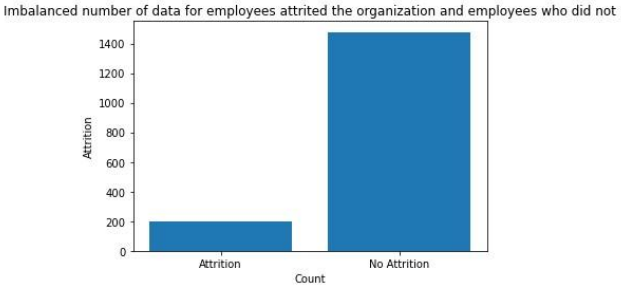
Compare the number of employees who responded "yes" on Attrition and who responded "no" on Attrition

In [8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [9]:
```
whetherAttrition = ['Attrition', 'No Attrition']
count = [len(attr[attr['Attrition'] == 'Yes']), len(attr[attr['Attrition'] == 'No'])]

plt.bar(whetherAttrition, count)
plt.title('Imbalanced number of data for employees attrited the organization and employees who did not')
plt.xlabel('Count')
plt.ylabel('Attrition')
```

Out[9]:
```
Text(0, 0.5, 'Attrition')
```

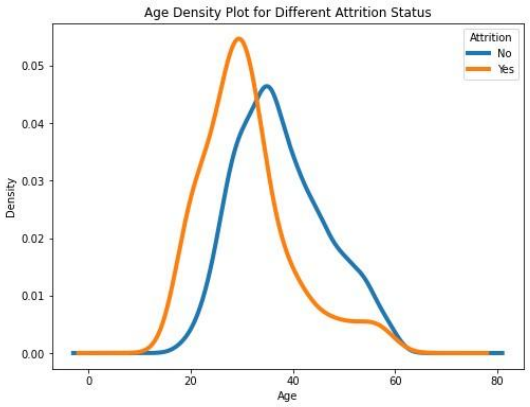


Obviously, the employees who didn't attrite the company is much more frequently represented in dataset than the employees who attrited the company. Therefore, oversampling method such as SMOTE might be necessary to address this class imbalance problem.

Compare age distributions of employees based on their attrition status

In [10]:
```
age_dist = attr.pivot(columns = 'Attrition',
                      values = 'Age')
age_dist.plot.kde(figsize = (8, 6),
                  linewidth = 4)
plt.title('Age Density Plot for Different Attrition Status')
plt.xlabel('Age')
```

Out[10]:
```
Text(0.5, 0, 'Age')
```



Around ages of 20 to mid-30, employees are more likely to attrite, while in ages of 40 to 60, employees are less likely to attrite.

Compare the likelihood of attrition by Departments

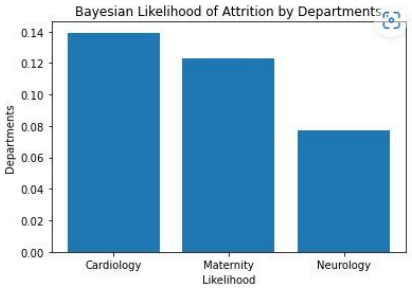
In [11]:
```
prop = []
for department in attr['Department'].unique():
    prop.append(len(attr[attr['Department'] == department][attr['Attrition'] == 'Yes']) /
                len(attr[attr['Department'] == department]))
prop
```

```
C:\Users\A\AppData\Local\Temp\ipykernel_20088\2797141860.py:3: UserWarning: Boolean Series key will be reindexed to match DataF
rame index.
  prop.append(len(attr[attr['Department'] == department][attr['Attrition'] == 'Yes']) /
```

Out[11]:
```
[0.1393596986817326, 0.12311557788944724, 0.07736389684813753]
```

In [12]:
```
Departments = attr['Department'].unique()
plt.bar(Departments, prop)
plt.title('Bayesian Likelihood of Attrition by Departments')
plt.xlabel('Likelihood')
plt.ylabel('Departments')
```

Out[12]:
```
Text(0, 0.5, 'Departments')
```

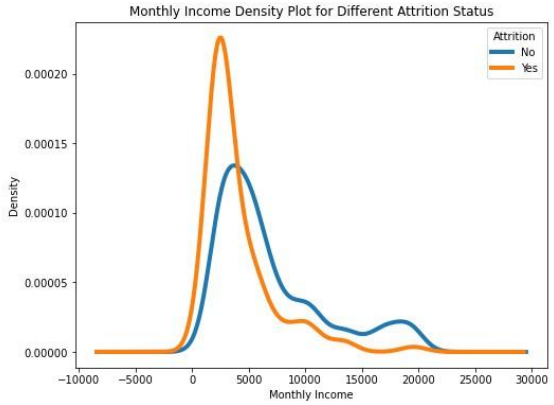


Based on conditional probability inferences, the employees working for Cardiology are the most likely to attrite.

Monthly Income Distribution by attrition status

In [13]:
```
age_dist = attr.pivot(columns = 'Attrition',
                      values = 'MonthlyIncome')
age_dist.plot.kde(figsize = (8, 6),
                  linewidth = 4)
plt.title('Monthly Income Density Plot for Different Attrition Status')
plt.xlabel('Monthly Income')
```

Out[13]: Text(0.5, 0, 'Monthly Income')



Employees who earns less than 5000 USD is far more likely to attrite, while employees who earns more than 5000 USD is more likely to stay at the company.

Feature Correlation Heatmap

In [14]:
```
plt.figure(figsize=(32, 10))
heatmap = sns.heatmap(attr.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```

In [15]:
```
#Since the correlation between monthly income ad job level is very high as 0.95, drop one of them.
attr = attr.drop("JobLevel", axis=1)
```

Data Engineering

Label Encode Categorical Variables

In [16]:
```
cat_vars = ['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'Over1
for cat_var in cat_vars:
    attr[cat_var] = attr[cat_var].astype('category')
    attr[cat_var] = attr[cat_var].cat.codes
attr[cat_vars]
```

Out[16]:

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	Over18	OverTime
0	0	2	0	1	0	2	2	0	1
1	0	1	1	1	1	3	1	0	0
2	1	2	1	4	1	2	2	0	1
3	0	1	1	1	0	3	1	0	1
4	0	2	1	3	1	2	1	0	0
...	...	...	...	...	...	...	...	...	...
1671	1	2	2	5	1	2	2	0	1
1672	0	2	0	2	0	2	1	0	1
1673	0	2	1	1	0	3	2	0	0
1674	0	2	2	1	0	4	1	0	0
1675	0	2	0	3	0	2	2	0	0

1676 rows × 9 columns

Scale Numerical Variables using Standard Scaler

In [17]:
```
from sklearn.preprocessing import StandardScaler

num_vars = ['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount', 'EnvironmentSatisfaction', 'HourlyRate', 'JobIr
            'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike',
            'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours', 'Shift',
            'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSincel
attr[num_vars] = StandardScaler().fit_transform(attr[num_vars])
```

In [18]:
```
attr[num_vars]
```

Out[18]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobSatisfaction	MonthlyIncom
0	0.452933	0.750837	-1.008126	-0.884927	0.0	-0.651470	1.412260	0.385287	1.142851	-0.11074
1	1.329511	-1.299105	-0.149829	-1.860033	0.0	0.259935	-0.221278	-1.015454	-0.669276	-0.29331
2	0.014644	1.425849	-0.885512	-0.884927	0.0	1.171341	1.313258	-1.015454	0.236787	-0.93642
3	-0.423644	1.473174	-0.762898	1.065286	0.0	1.171341	-0.468784	0.385287	0.236787	-0.76316
4	-1.081078	-0.521970	-0.885512	-1.860033	0.0	-1.562875	-1.260803	0.385287	-0.669276	-0.64490
...	...	...	...	...	...	...	...	...	...	...
1671	-1.190650	-0.820869	1.811993	0.090180	0.0	0.259935	0.026227	-2.416195	1.142851	-0.88353
1672	1.000794	0.808126	0.095399	0.090180	0.0	0.259935	1.412260	-1.015454	1.142851	0.54039
1673	-1.848083	0.394650	-1.008126	0.090180	0.0	1.171341	0.867747	-1.015454	-0.669276	-0.77860
1674	0.233789	-0.833323	-1.008126	-1.860033	0.0	1.171341	-0.023274	-1.015454	1.142851	1.31699
1675	-1.081078	-0.721236	-0.885512	-0.884927	0.0	-1.562875	1.164754	1.786028	0.236787	-0.00349

1676 rows × 24 columns

Because there are 35 features in total, PCA, dimensionality reduction method, could be used to prevent the overfitting the classification model by reducing the number of features.

PCA

In [19]:
```
from sklearn.decomposition import PCA
attr_X = attr.drop(["Attrition", "EmployeeID"], axis=1)
pca = PCA(n_components=30)
p_c = pca.fit_transform(attr_X)
p_attri = pd.DataFrame(data = p_c)
print("Explained Variance Ratio: ", sum(pca.explained_variance_ratio_))
```

```
Explained Variance Ratio:  0.9999999999999998
```

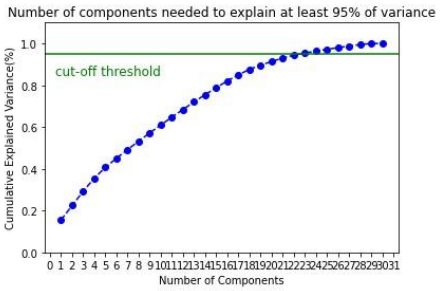
In [20]:
```
fix, axs = plt.subplots()
x = np.arange(1, 31, step=1)
y = np.cumsum(pca.explained_variance_ratio_)

plt.ylim(0.0, 1.1)
plt.plot(x, y, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 32, step=1))
plt.ylabel('Cumulative Explained Variance(%)')
plt.title('Number of components needed to explain at least 95% of variance')

plt.axhline(y=0.95, color='g', linestyle='-')
plt.text(0.5, 0.85, 'cut-off threshold', color = 'green', fontsize=12)
```

Out[20]:
```
Text(0.5, 0.85, 'cut-off threshold')
```



95% of variance can be explained by 24 components, 11 less than the original number of components.

In [21]:
```
pca = PCA(n_components=24)
p_c_final = pca.fit_transform(attr_X)
dataset = pd.DataFrame(data = p_c_final)
dataset
```

Out[21]:

	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17
0	-0.347277	-2.168564	0.550450	0.036645	-0.693835	0.915648	1.779300	-0.122578	-1.680746	-1.550377	...	-1.121753	-0.379135	0.315681	2.003244
1	0.763084	1.527927	2.943251	0.016039	-0.824654	-1.014640	-0.222221	0.168387	2.094838	-1.146386	...	-0.468547	-0.154376	0.421395	-0.738544
2	-2.272660	0.352369	-1.528655	1.652065	-0.542191	-0.083843	-0.931325	-0.885824	-0.650541	-0.870423	...	-0.179121	0.507278	0.228738	1.010570
3	-0.554440	-1.073476	-0.165857	-1.304711	-0.513145	-0.641514	0.482613	-1.532449	0.380021	0.206446	...	0.108676	-1.534827	-0.023140	-0.481539
4	-1.631552	-0.525507	-1.273997	0.591570	-0.287787	0.015776	-0.795590	-0.656300	0.498302	-0.478411	...	0.992411	-0.237281	0.745892	2.291507
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1671	-2.957007	1.985775	-1.293350	0.577011	-1.516639	-0.790056	2.295570	2.441431	-1.718019	-1.288667	...	-0.753847	-0.303678	-0.128300	-0.341357
1672	-0.186638	-0.598778	0.604071	1.118557	0.317303	-0.472633	-0.339378	0.141720	-0.796287	-0.817751	...	-0.863837	0.193863	-0.663437	-0.427179
1673	-2.921367	-0.846443	0.167998	-1.187607	-1.284067	0.461647	-0.301127	-1.212650	1.622090	0.700880	...	-0.319509	-1.483950	-0.576985	-0.699302
1674	3.315999	0.392869	0.434401	-1.990055	-0.296603	-1.757526	-0.596594	-1.058656	0.409339	-0.363268	...	-1.142116	0.858510	1.702285	-0.662610
1675	-0.205601	0.753140	-0.894389	-1.459781	-0.784269	1.028781	-0.861738	0.754617	-0.559854	-1.856142	...	-1.422423	0.463774	-0.578398	0.649289

1676 rows × 24 columns

SMOTE(Synthetic Minority Oversampling Technique)

In [22]:
```
import sklearn
from imblearn.over_sampling import SMOTE
from collections import Counter
X = dataset
y = attr['Attrition']
sm = SMOTE(random_state=42)
X_sm, y_sm = sm.fit_resample(X, y)
print('Original dataset shape %s' % Counter(y))
print('Resampled dataset shape %s' % Counter(y_sm))
```

```
Original dataset shape Counter({0: 1477, 1: 199})
Resampled dataset shape Counter({0: 1477, 1: 1477})
```

Building Models

Split Train and Test set

In [23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size=0.2, random_state=42)
```

Build Random Forest

In [24]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=7, random_state=0)
clf.fit(X_train, y_train)
```

Out[24]:
```
RandomForestClassifier(max_depth=7, random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluate Random Forest

In [25]:
```
print("Test Accuracy of random forest:", clf.score(X_test, y_test))
```

```
Test Accuracy of random forest: 0.9120135363790186
```

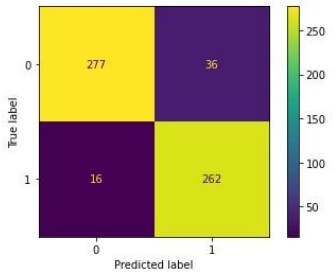
In [26]:
```
from sklearn.metrics import classification_report
y_preds = clf.predict(X_test)
labels = ['No Attrition', 'Attrition']
print(classification_report(y_test, y_preds, target_names=labels))
```

```
              precision    recall  f1-score   support

No Attrition       0.95      0.88      0.91       313
   Attrition       0.88      0.94      0.91       278

    accuracy                           0.91       591
   macro avg       0.91      0.91      0.91       591
weighted avg       0.91      0.91      0.91       591
```

In [27]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_preds, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Build Logistic Regression

In [28]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear')
lr.fit(X_train, y_train)
```

Out[28]:
```
LogisticRegression(solver='liblinear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluate Logistic Regression

In [33]:
```
print("Test Accuracy of logistic regression:", lr.score(X_test, y_test))
```

```
Test Accuracy of logistic regression: 0.8307952622673435
```

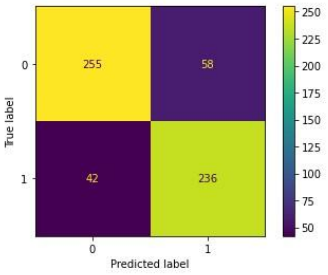
In [30]:
```
from sklearn.metrics import classification_report
y_preds_lr = lr.predict(X_test)
labels = ['No Attrition', 'Attrition']
print(classification_report(y_test, y_preds_lr, target_names=labels))
```

```
              precision    recall  f1-score   support

No Attrition       0.86      0.81      0.84       313
   Attrition       0.80      0.85      0.83       278

    accuracy                           0.83       591
   macro avg       0.83      0.83      0.83       591
weighted avg       0.83      0.83      0.83       591
```

In [31]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_preds_lr, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Build Support Vector Classifier

In [32]:
```
from sklearn.svm import SVC
svc = SVC(kernel='poly')
svc.fit(X_train, y_train)
```

Out[32]:
```
SVC(kernel='poly')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evluate Support Vector Classifier

In [34]:
```
print("Test Accuracy of support vector classifier:", svc.score(X_test, y_test))
```

```
Test Accuracy of support vector classifier: 0.9306260575296108
```

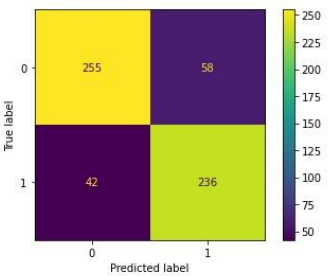
In [37]:
```
from sklearn.metrics import classification_report
y_preds_svc = svc.predict(X_test)
labels = ['No Attrition', 'Attrition']
print(classification_report(y_test, y_preds_svc, target_names=labels))
```

```
              precision    recall  f1-score   support

No Attrition       0.98      0.89      0.93       313
   Attrition       0.89      0.97      0.93       278

    accuracy                           0.93       591
   macro avg       0.93      0.93      0.93       591
weighted avg       0.93      0.93      0.93       591
```

In [36]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_preds_svc, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



V. Conclusion

Reducing the waste of efforts in hiring employees, a hiring manager needs to acknowledge which candidates are the most likely to stay at the company long-term. To notify a hiring manager whether a candidate would likely apply from company or not, a forecasting model for employee attrition, supported by SMOTE, a novel data oversampling method, and PCA, a dimensionality reduction method, was proposed. To select the best performing forecasting model, three models were built and tested: logistic regression, random forest, and support vector classifier. Support vector classifier outperformed the other two by achieving the test accuracy of 93.86%. Even with PCA, the number of features in a resulting dataset was still more than 20. The reason why the support vector classifier performed the best might be because the support vector classifier generally performs well in high-dimensional datasets. In future study, neural network, which are the best at dealing with high number of features out of all machine learning models, should be applied to forecasting employee attrition and be compared with support vector classifier.