Unleashing Machine Learning for Credit Risk Modeling

aiquants Research | © Dayton Nyamai

[63]:
```
import os
import time
import datetime
import numpy as np
import pandas as pd
import scipy.stats as scs
from pylab import plt, mpl

plt.style.use('seaborn-v0_8')
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['font.family'] = 'serif'
pd.set_option('mode.chained_assignment', None)
pd.set_option('display.float_format', '{:.4f}'.format)
np.set_printoptions(suppress=True, precision=4)
os.environ['PYTHONHASHSEED'] = '0'

import warnings
warnings.filterwarnings('ignore')
```

Credit Risk Estimation

Credit risk management plays a pivotal role in the overall risk management framework of banks. While market risk has received considerable research attention, it is crucial to recognize that a significant portion of banks' economic capital is allocated to credit risk. However, the current level of sophistication in traditional standard methods for measuring, analyzing, and managing credit risk may not adequately align with its inherent significance.

Credit risk is commonly defined as the likelihood that a borrower or counterparty fails to fulfill its obligations as per the agreed terms. The primary objective of credit risk management is to optimize a bank's risk-adjusted rate of return by effectively managing credit risk exposure within acceptable thresholds.

In this case, we present a comprehensive approach to estimating credit risk using state-of-the-art machine learning (ML) models. Let's create a practice exercise using German credit risk data.

Data

```
[64]: dataFrame = pd.read_csv('credit_risk_data.csv')
dataFrame.head()
```

```
[64]:    Unnamed: 0  Age     Sex  Job Housing Saving accounts Checking account  \
0           0   67    male    2     own             NaN           little
1           1   22  female    2     own          little         moderate
2           2   49    male    1     own          little              NaN
3           3   45    male    2    free          little           little
4           4   53    male    2    free          little           little

   Credit amount  Duration              Purpose  Risk
0           1169         6             radio/TV  good
1           5951        48             radio/TV   bad
2           2096        12            education  good
3           7882        42  furniture/equipment  good
4           4870        24                  car   bad
```

```
[65]: del dataFrame['Unnamed: 0']

dataFrame.describe()
```

```
[65]:             Age       Job  Credit amount  Duration
count 1000.0000 1000.0000      1000.0000 1000.0000
mean    35.5460    1.9040      3271.2580   20.9030
std     11.3755    0.6536      2822.7369   12.0588
min     19.0000    0.0000       250.0000    4.0000
25%     27.0000    2.0000      1365.5000   12.0000
50%     33.0000    2.0000      2319.5000   18.0000
75%     42.0000    2.0000      3972.2500   24.0000
max     75.0000    3.0000     18424.0000   72.0000
```

Frequency distribution of the numerical variables

Figure 1-1. shows the frequency distribution of the numerical variables in the dataset.

```
[66]: dataFrame_ = dataFrame.select_dtypes(exclude='O')
dataFrame_.hist(column=['Age', 'Job', 'Credit amount', 'Duration'], bins=10,␣
 ↪figsize=(10, 8))
plt.figtext(0.5, 0.0001, 'Fig 1-1. Credit risk data histogram', style='italic',␣
 ↪ha='center')
plt.show()
```

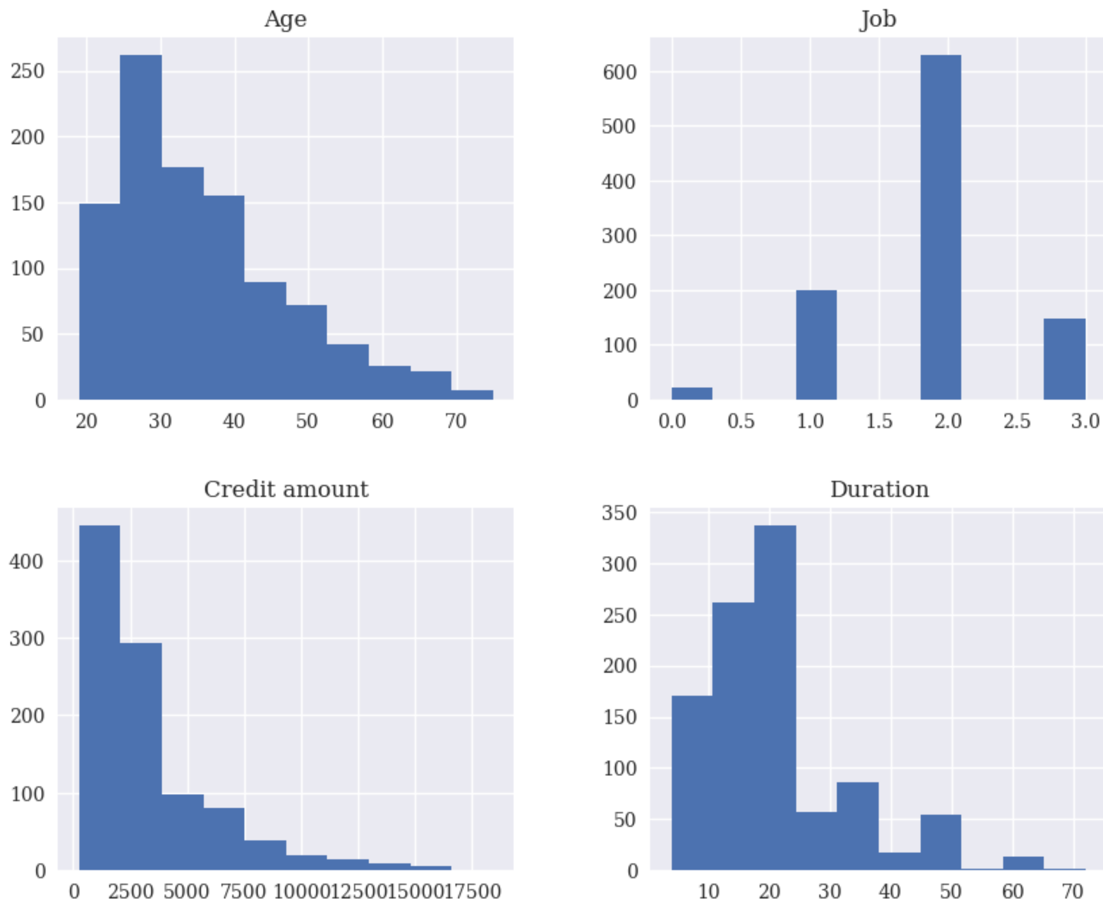


Fig 1-1. Credit risk data histogram

It turns out none of the variables follow a normal distribution. The age, credit amount, and duration variables are positively skewed as we can see in Figure 1-1.

Risk Bucketing

Risk bucketing is nothing but categorizing borrowers based on their creditworthiness. The underlying objective of risk bucketing is to create homogeneous groups or clusters that enable us to accurately assess credit risk. Various statistical methods can be employed to achieve risk bucketing, but in this case, we will utilize the K-means clustering technique to generate homogeneous clusters.

The elbow method The elbow method is a commonly used technique in cluster analysis. By observing the slope of the curve, we can identify the cut-off point where the curve becomes flatter. This indicates a decrease in inertia, which measures the distance between points within a cluster. Decreasing inertia is desirable for effective clustering. However, it is important to consider the trade-off between decreasing inertia and increasing the complexity of the analysis due to a higher number of clusters. Therefore, the stopping criteria for determining the optimal number of clusters is when the curve reaches a point of flatter slope. This can be implemented in code as follows:

[67]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(dataFrame_)

# Apply the elbow method to find the optimal number of clusters
distance = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)  # Explicitly set␣
↪n_init to suppress the warning
    kmeans.fit(scaled_data)
    distance.append(kmeans.inertia_)

# Plot the elbow method curve
plt.plot(range(1, 11), distance, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
#plt.title('The Elbow Method')
plt.figtext(0.5, -0.03, 'Fig 1-2. The Elbow method', style='italic',␣
↪ha='center')
plt.show()
```

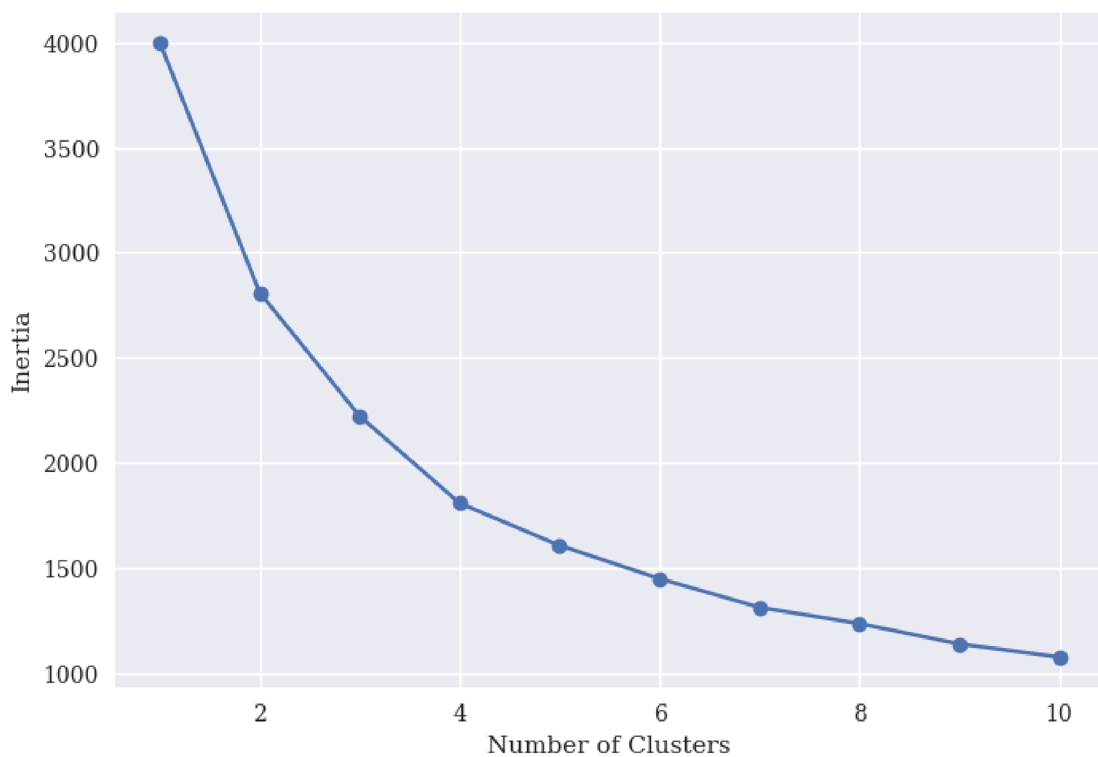


Fig 1-2. The Elbow method

Figure 1-2 above shows that the curve gets flatter after four clusters. Thus, the elbow method suggests that we stop at four clusters.

The Silhouette Scores The optimal number of clusters can be determined by looking at the average Silhouette score represented by the dashed line. In this case, the optimal number of clusters is two.

[68]:
```
# Plot the Silhouette scores for clusters 2 to 10
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from sklearn.metrics import silhouette_score

fig, ax = plt.subplots(1, 9, figsize=(18, 4))
for i, k in enumerate(range(2, 11)):
    #kmeans = KMeans(n_clusters=k)
    kmeans = KMeans(n_clusters=k, n_init=10)
    visualizer = SilhouetteVisualizer(kmeans, ax=ax[i])
    visualizer.fit(scaled_data)

plt.figtext(0.5, -0.03, 'Fig 1-3. Silhouette score', style='italic',␣
 ↪ha='center')
plt.show()
```

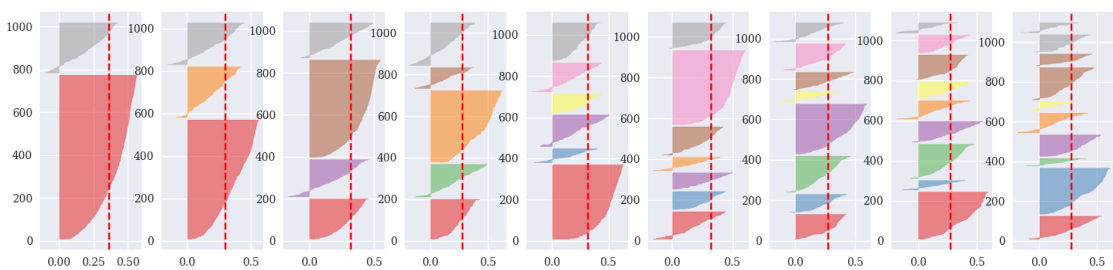


Fig 1-3. Silhouette score

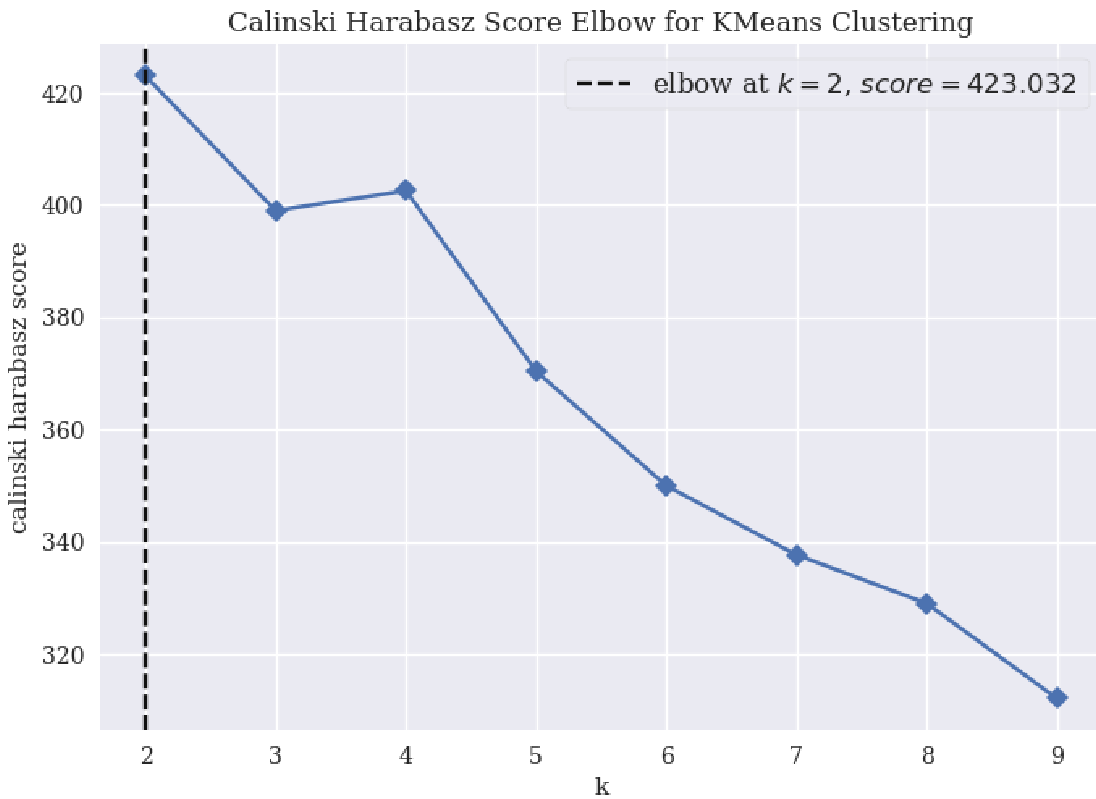
The CH Method

As previously stated, the CH method serves as a convenient tool for identifying the optimal clustering solution. Our objective is to identify the cluster with the highest CH score, and upon analysis, we observe that this score is attained by cluster 2.

[69]:
```
model = KMeans(n_init=10)

visualizer = KElbowVisualizer(model, k=(2, 10),
                              metric='calinski_harabasz',
                              timings=False)
visualizer.fit(scaled_data)
```

```
visualizer.show()
plt.show()
```



In light of these discussions, two clusters are chosen to be the optimal number of clusters, and the K-means clustering analysis is conducted accordingly.

[70]:
```
kmeans = KMeans(n_clusters=2)
clusters = kmeans.fit_predict(scaled_data)

plt.figure(figsize=(10, 12))
plt.subplot(311)
plt.scatter(scaled_data[:, 0], scaled_data[:, 2],
            c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 2], s = 80,
            marker= 'x', color = 'k')
plt.title('Age vs Credit')
plt.subplot(312)
plt.scatter(scaled_data[:, 0], scaled_data[:, 2],
            c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0],
```

```
            kmeans.cluster_centers_[:, 2], s = 80,
            marker= 'x', color = 'k')
plt.title('Credit vs Duration')
plt.subplot(313)
plt.scatter(scaled_data[:, 2], scaled_data[:, 3],
            c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 2],
            kmeans.cluster_centers_[:, 3], s = 120,
            marker= 'x', color = 'k')
plt.title('Age vs Duration')
plt.figtext(0.5, 0.001, 'Fig 1-5. K-means clusters', style='italic',␣
 ↪ha='center')
plt.show()
```

Fig 1-5. K-means clusters

Figure 1-5 illustrates the behavior of the observations, with the cluster center indicated by the cross sign 'x,' representing the centroid. The age variable exhibits a higher dispersion of data, with its centroid positioned above the credit variable. In the second subplot of Figure 1-5, we observe distinct clusters for two continuous variables: credit and duration. This suggests that the duration variable displays greater volatility compared to the credit variable. Finally, the last subplot examines the relationship between age and duration through scatter analysis, revealing a significant overlap of observations across these two variables.

Probability of Default Estimation with Logistic Regression

Having obtained the clusters, we can now treat customers with similar characteristics in a more efficient and reliable manner. By providing the model with data that has similar distributions, the learning process becomes easier and more stable. On the other hand, using all the customers in the entire sample may lead to poor and unstable predictions. This section is ultimately about calculating the probability of default with logistic regression.

To begin our application, we need to prepare the data. First, we will distinguish the clusters as 0 and 1. The credit data includes a column called "risk," which indicates the risk level of the customers. Next, we will examine the number of observations per risk in cluster 0 and cluster 1.

[71]:
```python
clusters, counts = np.unique(kmeans.labels_, return_counts=True)
cluster_dict = {}
for i in range(len(clusters)):
    cluster_dict[i] = scaled_data[np.where(kmeans.labels_==i)]

dataFrame['clusters'] = pd.DataFrame(kmeans.labels_)
df_scaled = pd.DataFrame(scaled_data)
df_scaled['clusters'] = dataFrame['clusters']
df_scaled['Risk'] = dataFrame['Risk']
df_scaled.columns = ['Age', 'Job', 'Credit amount', 'Duration', 'Clusters',␣
↪'Risk']

# Observing the number of observations of categories within a cluster
df_scaled[df_scaled.Clusters == 0]['Risk'].value_counts()
```

[71]:
```
good    569
bad     192
Name: Risk, dtype: int64
```

[72]:
```python
# Finding number of observations per category
df_scaled[df_scaled.Clusters == 1]['Risk'].value_counts()
```

[72]:
```
good    131
bad     108
Name: Risk, dtype: int64
```

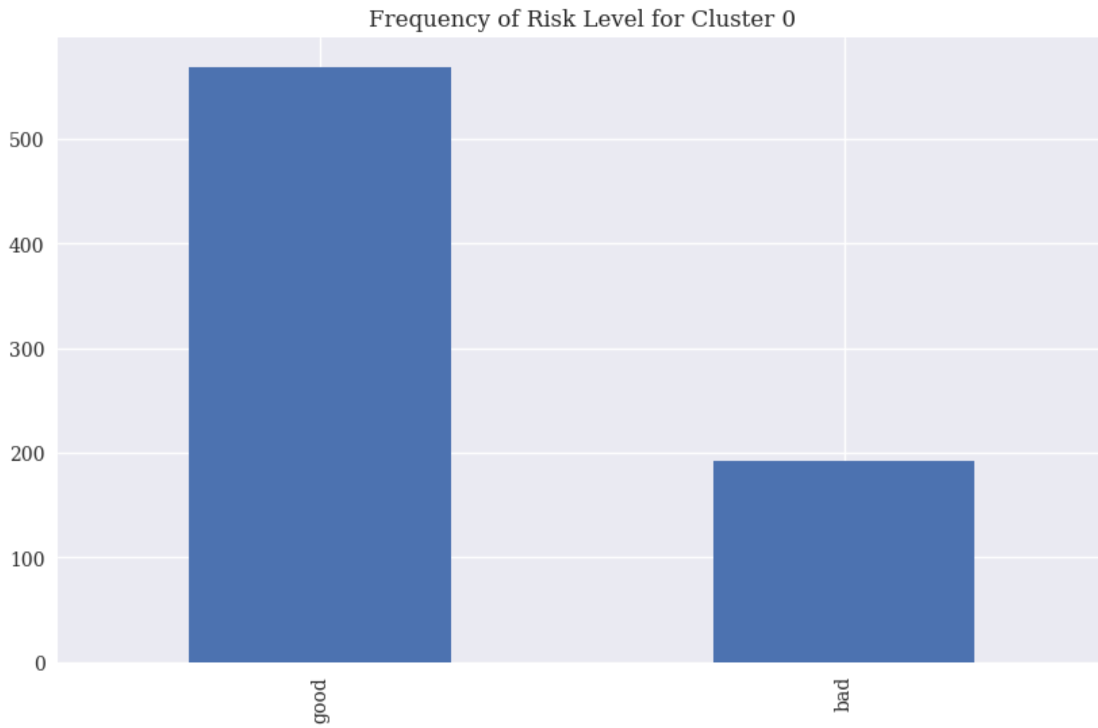
Next, we draw a couple of bar plots to show the difference of the number of observations per risk level category

[73]:
```python
# Plotting frequency of risk level for Cluster 0
df_scaled[df_scaled.Clusters == 0]['Risk'].value_counts().plot(kind='bar',␣
↪figsize=(10, 6), title="Frequency of Risk Level for Cluster 0")
```

```
plt.show()

# Plotting frequency of risk level for Cluster 1
df_scaled[df_scaled.Clusters == 1]['Risk'].value_counts().plot(kind='bar',␣
↪figsize=(10, 6), title="Frequency of Risk Level for Cluster 1")
plt.figtext(0.5, -0.03, 'Fig 1-6. Frequency of cluster risk level',␣
↪style='italic', ha='center')

plt.show()
```



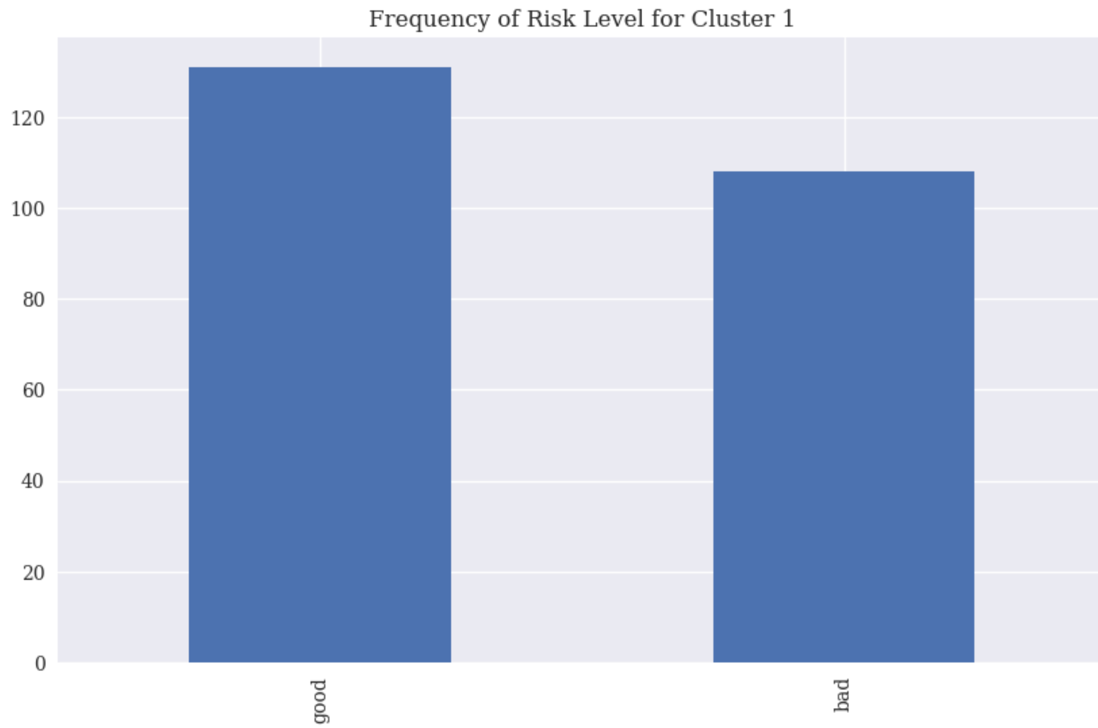


Fig 1-6. Frequency of cluster risk level

Based on the previously defined clusters, we can analyze the frequency of risk levels using a histogram. Figure 1-6, the top image illustrates an imbalanced distribution across risk levels in the first cluster. However, the bottom image shows a more balanced, if not perfectly balanced frequency of good and bad risk levels.

Next, we apply a train-test split. To perform a train-test split, we first need to convert the categorical variable "Risk" into a discrete variable. We assign the value of 1 to the category "good" and the value of 0 to the category "bad". In a train-test split, we allocate 80% of the data for training samples and reserve 20% for the test sample. This allows us to train our model on a majority of the data and evaluate its performance on the remaining portion.

[74]:
```
from sklearn.model_selection import train_test_split

# Replace 'good' with 1 and 'bad' with 0 in the 'Risk' column
df_scaled['Risk'] = df_scaled['Risk'].replace({'good': 1, 'bad': 0})

# Separate the features (X) and the target variables (y)
X = df_scaled.drop('Risk', axis=1)
y = df_scaled[['Risk', 'Clusters']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

```python
# Create separate training sets for each cluster
first_cluster_train = X_train[X_train['Clusters'] == 0].iloc[:, :-1]
second_cluster_train = X_train[X_train['Clusters'] == 1].iloc[:, :-1]
```

After these preparations, we are ready to move ahead and run the logistic regression to predict the probability of default.

[75]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve
from imblearn.combine import SMOTEENN
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')

X_train1 = first_cluster_train
y_train1 = y_train[y_train.Clusters == 0]['Risk']
smote = SMOTEENN(random_state = 2)
X_train1, y_train1 = smote.fit_resample(X_train1, y_train1.ravel())
logit = sm.Logit(y_train1, X_train1)
logit_fit1 = logit.fit()
print(logit_fit1.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.437683
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  374
Model:                          Logit   Df Residuals:                      370
Method:                           MLE   Df Model:                            3
Date:                Fri, 22 Sep 2023   Pseudo R-squ.:                  0.3627
Time:                        08:53:30   Log-Likelihood:                -163.69
converged:                       True   LL-Null:                       -256.87
Covariance Type:            nonrobust   LLR p-value:                 3.731e-40
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Age               1.7225      0.195      8.819      0.000       1.340
2.105
Job               0.3171      0.167      1.901      0.057      -0.010
0.644
Credit amount     1.2884      0.314      4.098      0.000       0.672
1.905
```

```
Duration          -1.4400      0.264      -5.458      0.000      -1.957
-0.923
================================================================================
=
```

The analysis of the first cluster data (imbalanced) yields the following results. Based on the findings, it can be observed that the variables of age, credit amount, and job exhibit a positive correlation with the creditworthiness of customers. Conversely, a negative association is observed between the dependent variable and the duration variable. These significant results, obtained at a 1% significance level, suggest that all estimated coefficients are statistically significant.

In general, this implies that a decrease in duration, coupled with an increase in credit amount, age, and job, indicates a higher probability of default."

Prediction analysis

The following analysis is done with test data

[76]:
```
first_cluster_test = X_test[X_test.Clusters == 0].iloc[:, :-1]
second_cluster_test = X_test[X_test.Clusters == 1].iloc[:, :-1]
X_test1 = first_cluster_test
y_test1 = y_test[y_test.Clusters == 0]['Risk']
pred_prob1 = logit_fit1.predict(X_test1)
false_pos, true_pos, _ = roc_curve(y_test1.values,  pred_prob1)
auc = roc_auc_score(y_test1, pred_prob1)
plt.plot(false_pos,true_pos, label="AUC for cluster 1={:.4f} "
                        .format(auc))
plt.plot([0, 1], [0, 1], linestyle = '--', label='45 degree line')
plt.legend(loc='best')
plt.title('ROC-AUC Curve 1')
plt.figtext(0.5, -0.03, 'Fig 1-7. ROC-AUC curve of the first cluster',␣
 ↪style='italic', ha='center')
plt.show()
```

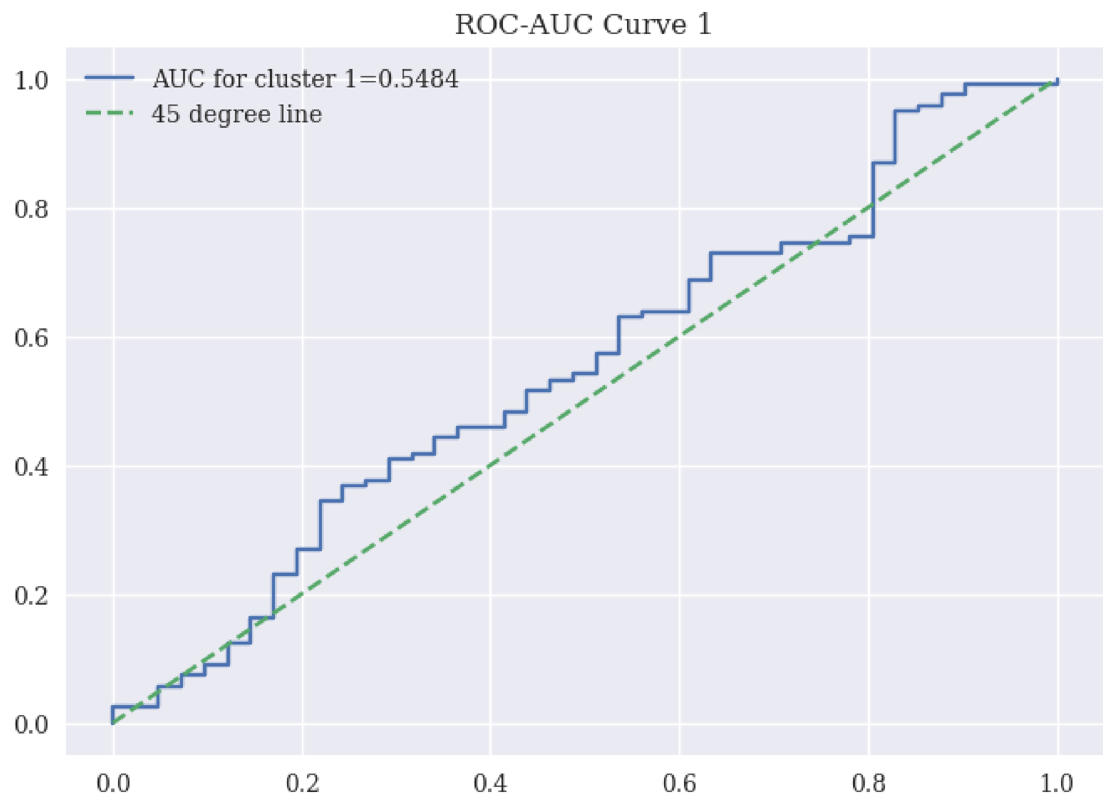


Fig 1-7. ROC-AUC curve of the first cluster

The ROC-AUC curve is a valuable tool when dealing with imbalanced data. Upon analyzing the ROC-AUC curve depicted in Figure 1-7, it becomes evident that the model's performance is subpar, as it barely surpasses the 45-degree line. In general, a desirable ROC-AUC curve should approach a value of 1, indicating a near-perfect separation.

Moving on to the second set of training samples obtained from the second cluster (balanced), the signs of the estimated coefficients of job, duration, and age are positive. This suggests that customers with a job type of 1 and a longer duration tend to default. Additionally, the credit amount variable shows a negative relationship with the dependent variable. However, it is important to note that all the estimated coefficients are statistically insignificant at a 95% confidence interval. Therefore, it is not meaningful to further interpret these findings.

Following the same approach as with the initial test data, we generate a second set of test data to execute the prediction and visualize the ROC-AUC curve. This process yields Figure 1-8.

[77]:
```
X_train2 = second_cluster_train
y_train2 = y_train[y_train.Clusters == 1]['Risk']
logit = sm.Logit(y_train2, X_train2)
logit_fit2 = logit.fit()
print(logit_fit2.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.687701
         Iterations 4
                           Logit Regression Results
==============================================================================
Dep. Variable:                   Risk   No. Observations:                  202
Model:                          Logit   Df Residuals:                      198
Method:                           MLE   Df Model:                            3
Date:                Fri, 22 Sep 2023   Pseudo R-squ.:              -0.0007228
Time:                        08:53:30   Log-Likelihood:                -138.92
converged:                       True   LL-Null:                       -138.82
Covariance Type:            nonrobust   LLR p-value:                     1.000
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Age               0.0100      0.145      0.069      0.945      -0.275
0.295
Job               0.1712      0.150      1.143      0.253      -0.122
0.465
Credit amount    -0.1110      0.115     -0.966      0.334      -0.336
0.114
Duration          0.1017      0.125      0.811      0.417      -0.144
0.348
================================================================================
=
```

[78]:

```
X_test2 = second_cluster_test
y_test2 = y_test[y_test.Clusters == 1]['Risk']
pred_prob2 = logit_fit2.predict(X_test2)

false_pos, true_pos, _ = roc_curve(y_test2.values,  pred_prob2)
auc = roc_auc_score(y_test2, pred_prob2)
plt.plot(false_pos,true_pos,label="AUC for cluster 2={:.4f} "
                 .format(auc))
plt.plot([0, 1], [0, 1], linestyle = '--', label='45 degree line')
plt.legend(loc='best')
plt.title('ROC-AUC Curve 2')
plt.figtext(0.5, -0.03, 'Fig 1-8. ROC-AUC curve of the second cluster',␣
 ↪style='italic', ha='center')
plt.show()
```

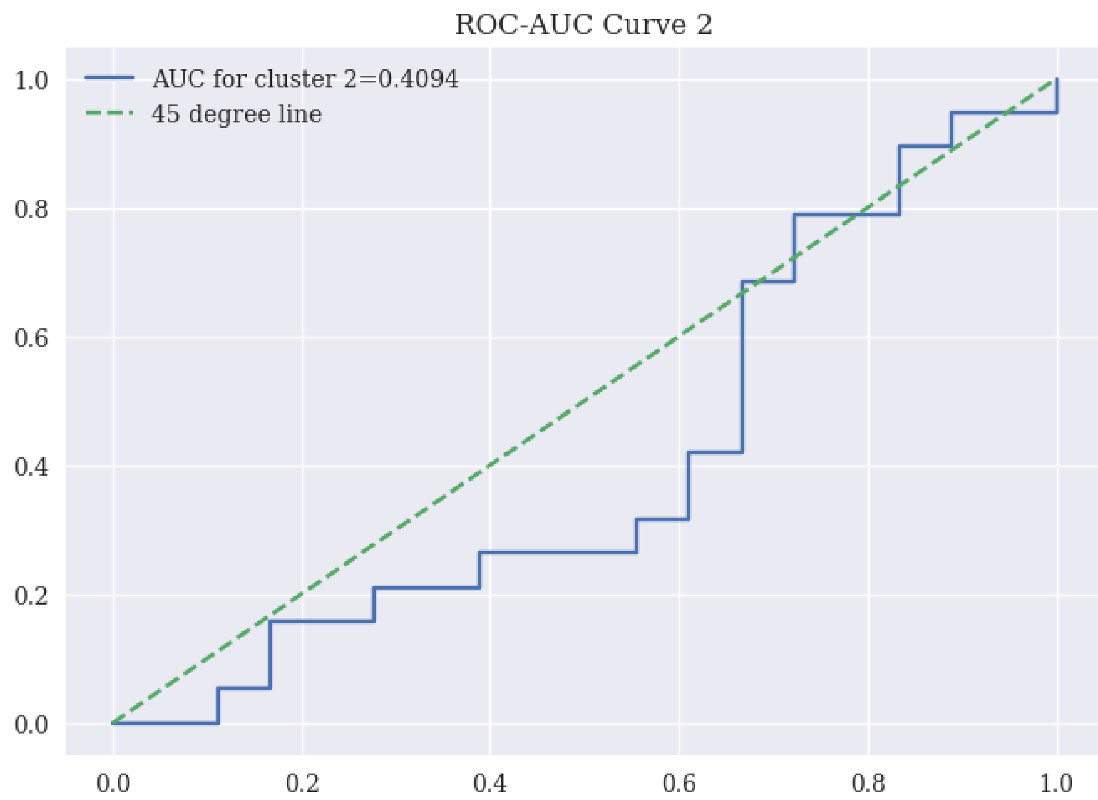


Fig 1-8. ROC-AUC curve of the second cluster

Given the test data, the result depicted in Figure 1-8 demonstrates a lack of promising improvement compared to the previous application. This lack of improvement is evident from the AUC score of 0.4. Considering this data, it can be cautiously concluded that the logistic regression approach does not effectively model the probability of default with the German credit risk dataset.

Now, let's explore alternative methodologies to evaluate the performance of logistic regression in accurately addressing this specific problem.

Probability of Default Estimation with Support Vector Machines

Support Vector Machines (SVM) are widely recognized as a robust parametric model that excels in handling high-dimensional data. In the realm of multivariate analysis, where predicting default cases is of utmost importance, SVM emerges as a promising choice. Its ability to navigate complex, multidimensional spaces makes it an ideal candidate for exploring the intricacies of default probability.

[ ]:
```
from sklearn.svm import SVC
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
```

```
param_svc = {
    'gamma': [1e-6, 1e-2],
    'C': [0.001, .09, 1, 5, 10],
    'kernel': ('linear', 'rbf')}

# Running the halving search using parallel processing
svc = SVC(class_weight='balanced')
halve_SVC = HalvingRandomSearchCV(svc, param_svc,
                                  scoring = 'roc_auc', n_jobs=-1)
halve_SVC.fit(X_train1, y_train1)
print('Best hyperparameters for first cluster in SVC {} with {}'.
      format(halve_SVC.best_score_, halve_SVC.best_params_))
```

[80]:
```
# Running a prediction analysis
y_pred_SVC1 = halve_SVC.predict(X_test1)
print('The ROC AUC score of SVC for first cluster is {:.4f}'.
      format(roc_auc_score(y_test1, y_pred_SVC1)))
```

```
The ROC AUC score of SVC for first cluster is 0.4968
```

The second cluster

[81]:
```
halve_SVC.fit(X_train2, y_train2)
print('Best hyperparameters for second cluster in SVC {} with {}'.
      format(halve_SVC.best_score_, halve_SVC.best_params_))
```

```
Best hyperparameters for second cluster in SVC 0.5548092105263158 with
{'kernel': 'rbf', 'gamma': 1e-06, 'C': 0.001}
```

[82]:
```
y_pred_SVC2 = halve_SVC.predict(X_test2)
print('The ROC AUC score of SVC for first cluster is {:.4f}'.
      format(roc_auc_score(y_test2, y_pred_SVC2)))
```

```
The ROC AUC score of SVC for first cluster is 0.5000
```

It turns out that the only difference across the two different samples occurs in the gamma and C hyperparameters. In the first cluster, the optimal C score is 1, whereas it is 0.001 in the second one. The higher C value indicates that we should choose a smaller margin to make a better classification. As for the gamma hyperparameter, both clusters take the same value. Having a lower gamma amounts to a larger influence of the support vector on the decision. The optimal kernel is Gaussian, and the gamma value is 0.01 for both clusters.

Probability of Default Estimation with Random Forest

The random forest classifier is an alternative model that can be utilized to estimate the probability of default. While random forest may not perform well in high dimensional cases, our dataset is not overly complex. The true strength of the random forest lies in its ability to provide accurate predictions when dealing with a large number of samples. Therefore, it is reasonable to consider that the random forest model may outperform the SVC model in our specific scenario.

In analyzing the initial cluster data, a comprehensive evaluation was conducted to determine the performance of various models. The results, quantified by the AUC score of 0.5385, unequivocally demonstrate that the random forest model outperforms its counterparts.

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42)
param_rfc = {'n_estimators': [100, 300],
                 'criterion' :['gini', 'entropy'],
                 'max_features': ['auto', 'sqrt', 'log2'],
                 'max_depth' : [3, 4, 5, 6],
                 'min_samples_split':[5, 10]}

halve_RF = HalvingRandomSearchCV(rfc, param_rfc,
                                 scoring = 'roc_auc', n_jobs=-1)
halve_RF.fit(X_train1, y_train1)

print('Best hyperparameters for first cluster in RF {} with {}'
      .format(halve_RF.best_score_, halve_RF.best_params_))
```

```
[84]: y_pred_RF1 = halve_RF.predict(X_test1)
print('The ROC AUC score of RF for first cluster is {:.4f}'.
      format(roc_auc_score(y_test1, y_pred_RF1)))
```

```
The ROC AUC score of RF for first cluster is 0.5389
```

The following code shows a random forest run based on the second cluster:

```
[ ]: halve_RF.fit(X_train2, y_train2)
print('Best hyperparameters for second cluster in RF {} with {}'.
      format(halve_RF.best_score_, halve_RF.best_params_))
```

```
[92]: y_pred_RF2 = halve_RF.predict(X_test2)
print('The ROC AUC score of RF for second cluster is {:.4f}'.
      format(roc_auc_score(y_test2, y_pred_RF2)))
```

```
The ROC AUC score of RF for second cluster is 0.6418
```

In the second cluster, the predictive performance of the random forest algorithm stands out with an impressive AUC score of 0.6418. This quantitative measure clearly demonstrates the superiority of the random forest model in accurately predicting outcomes. The exceptional performance of the random forest algorithm can be attributed, in part, to its ability to effectively handle low-dimensional data with a substantial number of observations. When faced with such data characteristics, the random forest algorithm emerges as an optimal choice, showcasing its capability to fit the data accurately and efficiently.

Probability of Default Estimation with Deep Neural Network

Given the intricate nature of estimating the probability of default, uncovering the underlying data structure poses a formidable challenge. However, the neural network (NN) structure excels in handling this complexity, making it an ideal candidate model for such tasks. To set up the NN model, we employ GridSearchCV, a powerful tool for optimizing the number of hidden layers, the choice of optimization technique, and the learning rate. This approach ensures that our model is fine-tuned to deliver accurate and reliable results.

[87]:
```
from sklearn.neural_network import MLPClassifier
param_NN = {"hidden_layer_sizes": [(100, 50), (50, 50), (10, 100)],
            "solver": ["lbfgs", "sgd", "adam"],
            "learning_rate_init": [0.001, 0.05]}

MLP = MLPClassifier(random_state=42)

param_halve_NN = HalvingRandomSearchCV(MLP, param_NN,
                                       scoring = 'roc_auc')
param_halve_NN.fit(X_train1, y_train1)

print('Best hyperparameters for first cluster in NN are {}'.
      format(param_halve_NN.best_params_))
```

```
Best hyperparameters for first cluster in NN are {'solver': 'lbfgs',
'learning_rate_init': 0.001, 'hidden_layer_sizes': (100, 50)}
```

[88]:
```
y_pred_NN1 = param_halve_NN.predict(X_test1)

print('The ROC AUC score of NN for first cluster is {:.4f}'.
      format(roc_auc_score(y_test1, y_pred_NN1)))
```

```
The ROC AUC score of NN for first cluster is 0.4980
```

[89]:
```
param_halve_NN.fit(X_train2, y_train2)

print('Best hyperparameters for first cluster in NN are {}'.
      format(param_halve_NN.best_params_))
```

```
Best hyperparameters for first cluster in NN are {'solver': 'adam',
'learning_rate_init': 0.05, 'hidden_layer_sizes': (100, 50)}
```

[90]:
```
y_pred_NN2 = param_halve_NN.predict(X_test2)

print('The ROC AUC score of NN for first cluster is {:.4f}'.
      format(roc_auc_score(y_test2, y_pred_NN2)))
```

```
The ROC AUC score of NN for first cluster is 0.5000
```

In comparing the ROC-AUC scores achieved by the DNN's second cluster and the Random Forest approach for modeling credit risk dataset default, it is evident that the Random Forest approach significantly outperforms its rival, the DNN method. While the DNN's second cluster achieves a ROC-AUC score of 0.5, the Random Forest approach achieves an impressive score of 0.6418. This substantial difference in performance highlights the superior modeling capabilities of the Random Forest approach in accurately predicting credit risk default.