Support vector machine (SVM) classifier

Supervised non-parametric machine learning method to classify a target variable.

In [16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
```

Dataset

The *default of credit card clients* dataset contains 23 variables from customers in Taiwan to classify if a customer will default or not on their payment.

In [2]:
```
#Load the data
df = pd.read_csv("https://raw.githubusercontent.com/dswede43/ML-methods/main/data/d

#rename the target variable column
df.rename({'default payment next month': 'DEFAULT'}, axis = 'columns', inplace = Tr

#remove the ID's column
df = df.drop('ID', axis = 1)
df.head()
```

Out[2]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_
0	20000	2	2	1	24	2	2	-1	-1	-2	...	
1	120000	2	2	2	26	-1	2	0	0	0	...	
2	90000	2	2	2	34	0	0	0	0	0	...	
3	50000	2	2	1	37	0	0	0	0	0	...	
4	50000	1	2	1	57	-1	0	-1	0	0	...	

5 rows × 24 columns

Feature variables (23)

1. LIMIT_BAL (continuous): credit limit

2. SEX (nominal): sex of the customer

- 1 = male
- 2 = female

3. EDUCATION (nominal): level of education

- 1 = gradudate school
- 2 = university
- 3 = high school
- 4 = others

4. MARRIAGE (nominal): marital status

- 1 = married
- 2 = single
- 3 = other

5. AGE (continuous): age of customer

6. PAY_ (nominal): when the last 6 bills were paid

- -1 = paid on time
- 1 = payment delayed by 1 month
- 2 = 2 month delay
- ...
- 9 = 9 month delay

7. BILL_AMT (continuous): amount of last 6 bills

8. PAY_AMT (continuous): payment amount towards last 6 bills

Target variable (1)

9. DEFAULT (nominal): whether or not the customer has defaulted on their next payment

- 0 = did not default
- 1 = defaulted

Data preprocessing

Removing NA values

In [3]:
```
#check for NA values
df.isna().sum()
```

Out[3]:
```
LIMIT_BAL    0
SEX          0
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_0        0
PAY_2        0
PAY_3        0
PAY_4        0
PAY_5        0
PAY_6        0
BILL_AMT1    0
BILL_AMT2    0
BILL_AMT3    0
BILL_AMT4    0
BILL_AMT5    0
BILL_AMT6    0
PAY_AMT1     0
PAY_AMT2     0
PAY_AMT3     0
PAY_AMT4     0
PAY_AMT5     0
PAY_AMT6     0
DEFAULT      0
dtype: int64
```

In [4]:
```
#explore unique values in features
print(df['EDUCATION'].unique())
print(df['MARRIAGE'].unique())
```

```
[2 1 3 5 4 6 0]
[1 2 3 0]
```

There are no NA values in the dataset. However, according to the variable information on *UC Irvine*, **EDUCATION** and **MARRIAGE** variables contain more unique values than expected. Therefore, I will assume zero's represent NA values and remove them.

In [5]:
```
#remove rows with an NA values
df = df.loc[(df['EDUCATION'] != 0) & (df['MARRIAGE'] != 0)]
len(df)
```

Out[5]:
```
29932
```

Downsamping data

The 29,932 samples is too large for the computational intensity of a support vector machine. Therefore, the data will be downsampled by taking a **simple random sample (SRS)** of the data.

In [6]:
```
#create data subsets of target variable classes
df_no_default = df[df['DEFAULT'] == 0]
df_default = df[df['DEFAULT'] == 1]
```

```
#SRS of no default data subset
df_no_default = df_no_default.sample(n = 1000,
                                     replace = False,
                                     random_state = 42)

#SRS of default data subset
df_default = df_default.sample(n = 1000,
                               replace = False,
                               random_state = 42)

#concatenate both datasets
df_downsampled = pd.concat([df_no_default, df_default])
len(df_downsampled)
```

Out[6]: 2000

Formatting feature and target variables

Defining the feature and target variable datasets.

In [7]:
```
#feature variable dataset
X = df_downsampled.drop('DEFAULT', axis = 1)

#target variable dataset
y = df_downsampled['DEFAULT']
```

One-hot encoding

Apply one-hot encoding to multivariate features to remove ordinal patterns in the data where non exist.

In [8]:
```
#define multivariate feature variables
multivariates = ['EDUCATION',
                 'MARRIAGE',
                 'PAY_0',
                 'PAY_2',
                 'PAY_3',
                 'PAY_4',
                 'PAY_5',
                 'PAY_6']

#apply one-hot encoding to multivariate features
X_encoded = pd.get_dummies(X, columns = multivariates)
X_encoded.head()
```

Out[8]:

	LIMIT_BAL	SEX	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_/
641	130000	2	28	100143	50456	50000	0	0	
4678	170000	1	29	165027	168990	172307	35234	32869	3
16004	180000	2	29	25781	26000	26310	26662	26166	2
22974	210000	2	32	355	975	410	0	0	
17535	190000	2	45	76433	78472	80548	81778	83082	8

5 rows × 80 columns

Dataset train and test splitting

Splitting the data into training and testing datasets. This is done before scaling to prevent data leakage.

In [9]:
```
#train and test dataset split
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size = 0.2,
```

Centering and scaling data

The **Radial Basis Function (RBF)** kernel relies on distance metrics to compute similarities between data points in higher dimensional feature spaces. Therefore, centering and scaling the features to have a mean = 0 and standard deviation = 1 ensures each feature contributes equally to the models decision-making process.

In [10]:
```
#center and scale both the train and test feature data separately
X_train_scaled = scale(X_train)
X_test_scaled = scale(X_test)
```

Cross-validataion: optimization of hyperparameters

Randomized search to optimize hyperparameters.

In [11]:
```
#define parameter grid
param_grid = {
    'C': [0.5, 1, 10, 100], #error bounds
    'gamma': ['scale', 1, 0.1, 0.01, 0.001, 0.0001],
    'kernel': ["linear", "rbf", "poly"]} #kernel function
```

In [12]:
```
#define the model
svc = SVC()

#define the parameters for randomized search
optimal_params = RandomizedSearchCV(estimator = svc,
                                    param_distributions = param_grid,
```

```
                                   cv = 5,
                                   n_iter = 10,
                                   scoring = 'accuracy',
                                   random_state = 42,
                                   return_train_score = False)

#perform the randomized search to optimize hyperparameters
optimal_params.fit(X_train_scaled, y_train)
```

Out[12]:

```
▸ RandomizedSearchCV
  ▸ estimator: SVC
      ▸ SVC
```

In [13]:

```
#print the optimized hyperparameters
print(optimal_params.best_params_)
```

```
{'kernel': 'rbf', 'gamma': 0.01, 'C': 1}
```

Best performing model

Create the best performing model.

In [14]:

```
#fit the best performing model
best_svc = SVC(C = 1, kernel = 'rbf', gamma = 0.01, random_state = 42)
best_svc.fit(X_train_scaled, y_train)
```

Out[14]:

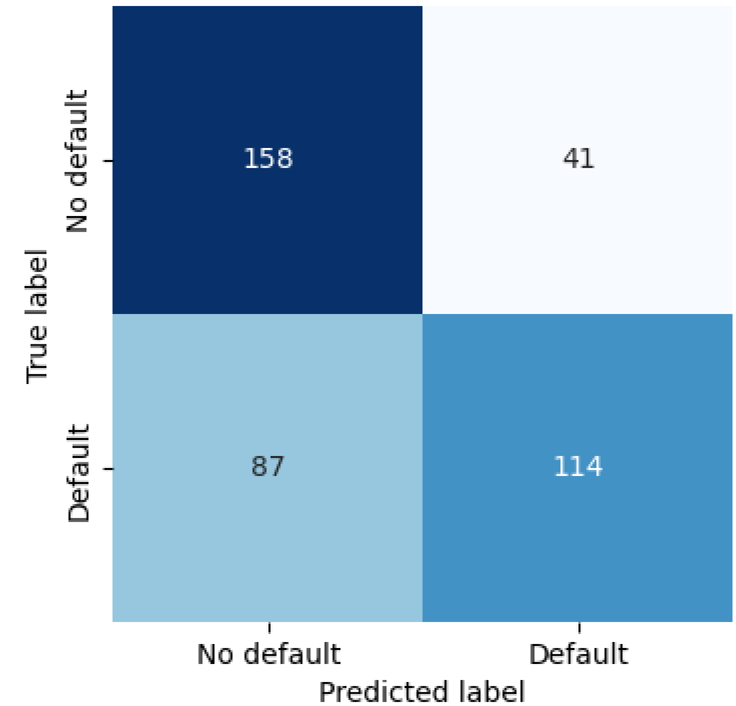
```
▾ SVC
SVC(C=1, gamma=0.01, random_state=42)
```

Confusion matrix

Visualize the model performance of the final model using a confusion matrix.

In [17]:

```
#make model predictions
y_pred = best_svc.predict(X_test_scaled)

#create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

#plot the confusion matrix
plt.figure(figsize = (4,4))
sns.heatmap(cm, annot = True, cmap = 'Blues', cbar = False, fmt = 'd')
plt.xticks([0.5,1.5], ['No default','Default'])
plt.yticks([0.5,1.5], ['No default','Default'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```



The SVC model correctly classified **158/199 (79.4%)** of clients as **not defaulting** on their payment and **114/201 (56.7%)** of clients as **defaulting** on their payments.

Visualize the SVC using PCA

Use PCA to reduce the number of feature variables to allow a 2-dimensional visualization of the SVC.

In [18]:
```
#reduce feature space with PCA
pca = PCA(n_components = 2)
pca_results = pca.fit_transform(X_train_scaled)

#obtain the variance explained by each principal component
per_var = np.round(pca.explained_variance_ratio_ * 100, decimals = 1)
print(f"The variance explained by the first two principal components are {per_var[0
```

```
The variance explained by the first two principal components are 11.20% and 6.30%
respectively.
```

A small proportion of the variance (**17.5%**) in feature variables can be explained by the first two principal components. Therefore, the following visualization will not give an ideal representation of the original dataset and SVC model.

In [19]:
```
#define PCs
X_train_pc1 = pca_results[:, 0]
X_train_pc2 = pca_results[:, 1]

#stack together the PC's for modelling
X_train_pca = np.column_stack((X_train_pc1, X_train_pc2))
```

SVC modelling of principal components

Optimize the SVC model using the first two principal components created from the feature variables.

In [20]:
```
#define the model
svc_pca = SVC()

#define the parameters for randomized search
optimal_params = RandomizedSearchCV(estimator = svc_pca,
                                    param_distributions = param_grid,
                                    cv = 5,
                                    n_iter = 10,
                                    scoring = 'accuracy',
                                    random_state = 42,
                                    return_train_score = False)

#perform the randomized search to optimize hyperparameters
optimal_params.fit(X_train_pca, y_train)
```

Out[20]:
```
▸ RandomizedSearchCV
  ▸ estimator: SVC
      ▸ SVC
```

In [21]:
```
#print the optimized hyperparameters
print(optimal_params.best_params_)
```

```
{'kernel': 'rbf', 'gamma': 0.01, 'C': 0.5}
```

In [22]:
```
#fit the best performing PCA model
best_pca_svc = SVC(C = 0.5, kernel = 'rbf', gamma = 0.01, random_state = 42)
best_pca_svc.fit(X_train_pca, y_train)
```

Out[22]:
```
▾ SVC
SVC(C=0.5, gamma=0.01, random_state=42)
```

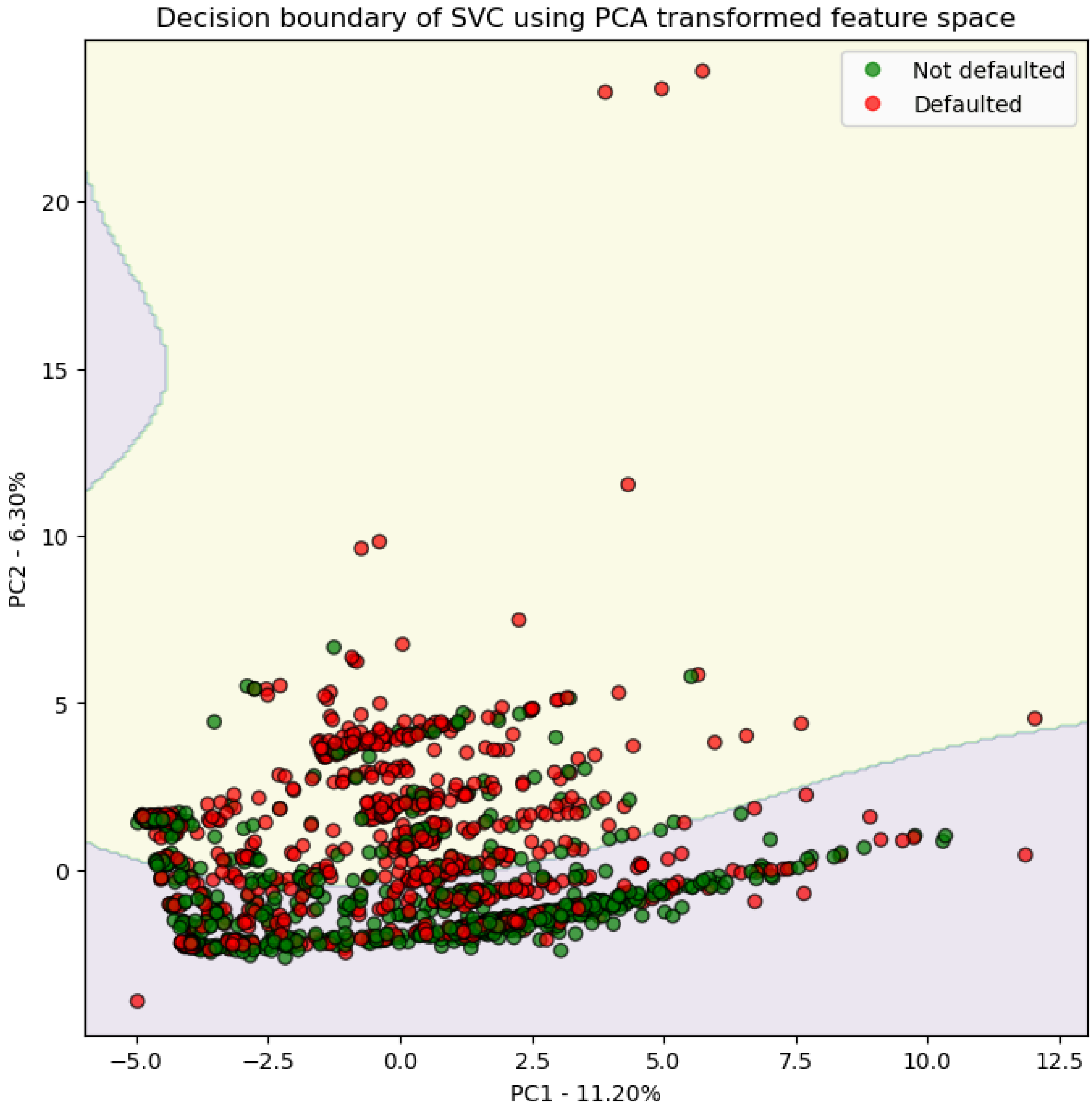
Visualization of the SVC decision boundary

Create a scatterplot of the PCA transformed feature space to visualize the SVC decision boundary.

In [23]:
```
#define the range of x-values
x_min = X_train_pc1.min() - 1
x_max = X_train_pc1.max() + 1

#define the range of y-values
y_min = X_train_pc2.min() - 1
y_max = X_train_pc2.max() + 1
```

```
#create a cartesian plane of x and y coordinates
xx, yy = np.meshgrid(np.arange(start = x_min, stop = x_max, step = 0.1),
                     np.arange(start = y_min, stop = y_max, step = 0.1))

#make predictions on the cartesian plane to create the model decision boundary
Z = best_pca_svc.predict(np.column_stack((xx.ravel(), yy.ravel())))
Z = Z.reshape(xx.shape)
```

In [24]:
```
#create the scatterplot
fig, ax = plt.subplots(figsize = (8,8))

#plot the model decision boundary
ax.contourf(xx, yy, Z, alpha = 0.1)

#define the point colors
cmap = colors.ListedColormap(['green','red'])

#create the scatterplot
scatter = ax.scatter(X_train_pc1, X_train_pc2, c = y_train,
                     cmap = cmap,
                     edgecolors = 'k',
                     alpha = 0.7)

#define the plot legend
legend = ax.legend(scatter.legend_elements()[0],
                   scatter.legend_elements()[1],
                   loc = 'upper right')
legend.get_texts()[0].set_text("Not defaulted")
legend.get_texts()[1].set_text("Defaulted")

#label the plot
ax.set_xlabel(f"PC1 - {per_var[0]:.2f}%")
ax.set_ylabel(f"PC2 - {per_var[1]:.2f}%")
ax.set_title("Decision boundary of SVC using PCA transformed feature space")
plt.show()
```



Conclusion

Steps completed

- imported the dataset
- removed NA values from the data
- downsampled the data to reduce the computational load
- formatted the target and feature data
- split the data into train and test sets
- centered and scaled the data to ensure equal contribution of each feature
- performed cross-validation to tune the model hyperparameters
- created the best performing model
- visualized the models decision boundary using PCA transformed feature space

The SVC model showed a true positive rate of **56.7%** (true client default) and true negative rate of **79.6%** (true client did not default). Perhaps another machine learning model such as the decision tree classifier coupled with random forests would provide a more comlex decision boundary to classify client payment defaults.