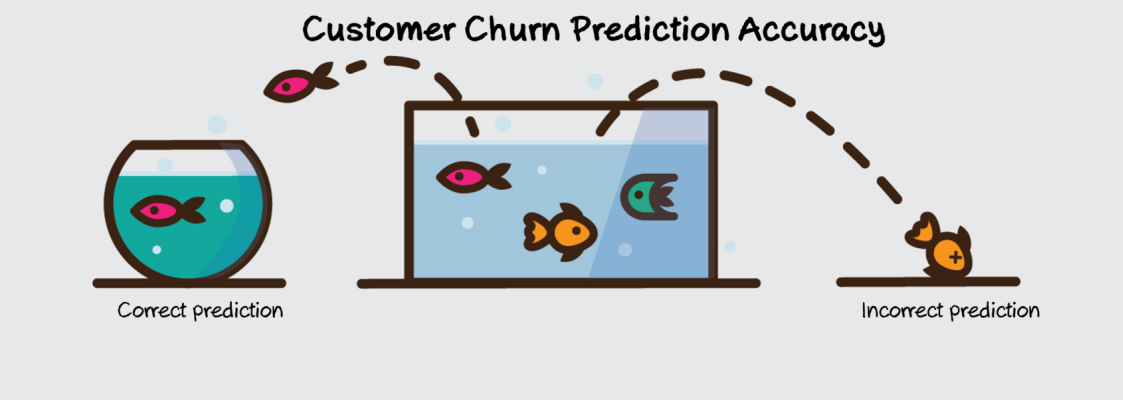
Classification Lab

July 14, 2021

[1]:
```
from IPython.display import Image
Image('churn.png')
```

[1]:



1 1. Data preprocessing and exploration

1.1 1.1. Importing libraries and dataset

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
dataset = pd.read_csv('Churn_Modelling.csv')
```

[4]:
```
dataset.head()
```

[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	\
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	

```
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[5]:
```
dataset.shape
```

[5]:
```
(10000, 14)
```

[6]:
```
dataset.isnull().sum()
```

[6]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

1.2 1.2. Statistical description

[7]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
```

```
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

[8]:
```
dataset.describe()
```

[8]:

	RowNumber	CustomerId	CreditScore	Age	Tenure
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000

	Balance	NumOfProducts	HasCrCard	IsActiveMember
count	10000.000000	10000.000000	10000.00000	10000.000000
mean	76485.889288	1.530200	0.70550	0.515100
std	62397.405202	0.581654	0.45584	0.499797
min	0.000000	1.000000	0.00000	0.000000
25%	0.000000	1.000000	0.00000	0.000000
50%	97198.540000	1.000000	1.00000	1.000000
75%	127644.240000	2.000000	1.00000	1.000000
max	250898.090000	4.000000	1.00000	1.000000

	EstimatedSalary	Exited
count	10000.000000	10000.000000
mean	100090.239881	0.203700
std	57510.492818	0.402769
min	11.580000	0.000000

```
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

[9]:
```
dataset.describe(include=['object'])
```

[9]:
```
        Surname Geography Gender
count    10000     10000  10000
unique    2932         3      2
top      Smith    France   Male
freq        32      5014   5457
```

[10]:
```
dataset['Geography'].unique()
```

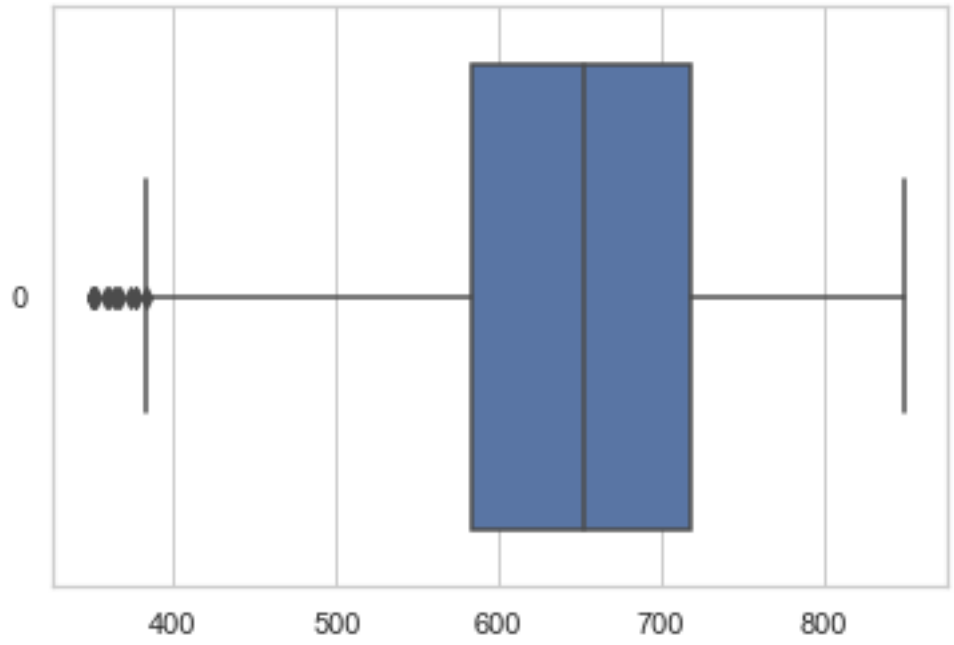
[10]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

[11]:
```
dataset['Geography'].value_counts()
```

[11]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

1.3 1.3. Boxplot

[12]:
```
sns.set(style='whitegrid')
ax = sns.boxplot(data = dataset['CreditScore'], orient = 'h')
```



[13]:
```
IQR = dataset['CreditScore'].quantile(0.75)-dataset['CreditScore'].quantile(0.
↪25)
```

[14]:
```
IQR
```

[14]:
```
134.0
```

[15]:
```
Inf = dataset['CreditScore'].quantile(0.25)-1.5*IQR
```

[16]:
```
Inf
```

[16]:
```
383.0
```

[17]:
```
dataset[dataset['CreditScore']<Inf].count()
```

[17]:
```
RowNumber          15
CustomerId         15
Surname            15
CreditScore        15
Geography          15
Gender             15
Age                15
Tenure             15
Balance            15
NumOfProducts      15
```

```
HasCrCard          15
IsActiveMember     15
EstimatedSalary    15
Exited             15
dtype: int64
```

[18]:
```
Sup = dataset['CreditScore'].quantile(0.75)+1.5*IQR
```

[19]:
```
Sup
```

[19]:
```
919.0
```

[20]:
```
my_list = ['CreditScore', 'Age']
```

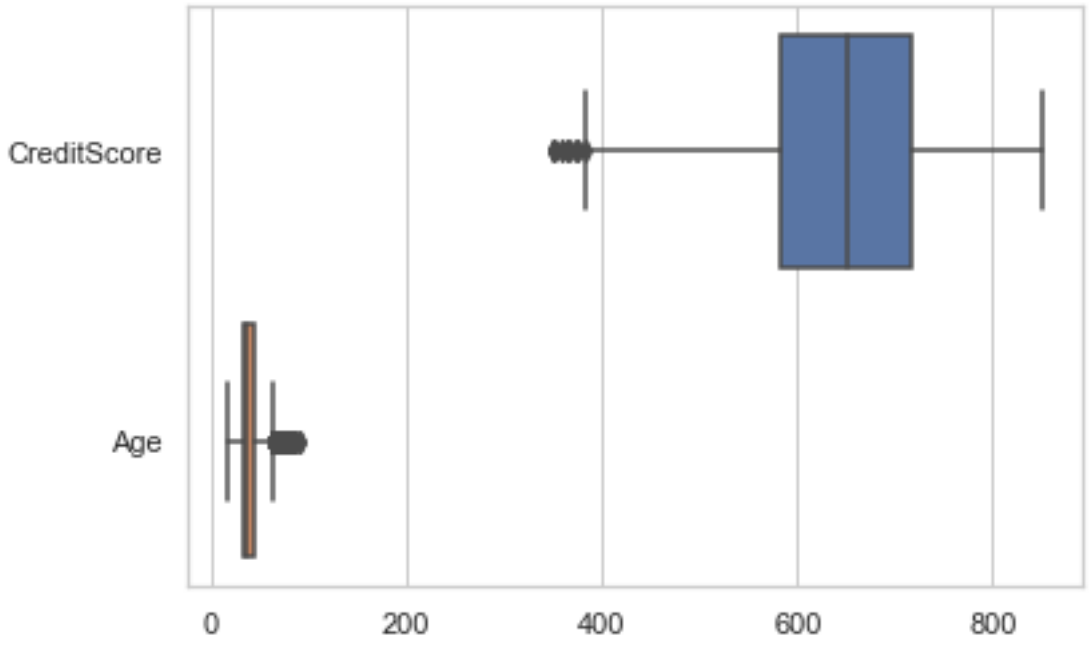
[21]:
```
my_list
```

[21]:
```
['CreditScore', 'Age']
```

[22]:
```
dataset[my_list].head()
```

[22]:
```
   CreditScore  Age
0          619   42
1          608   41
2          502   42
3          699   39
4          850   43
```

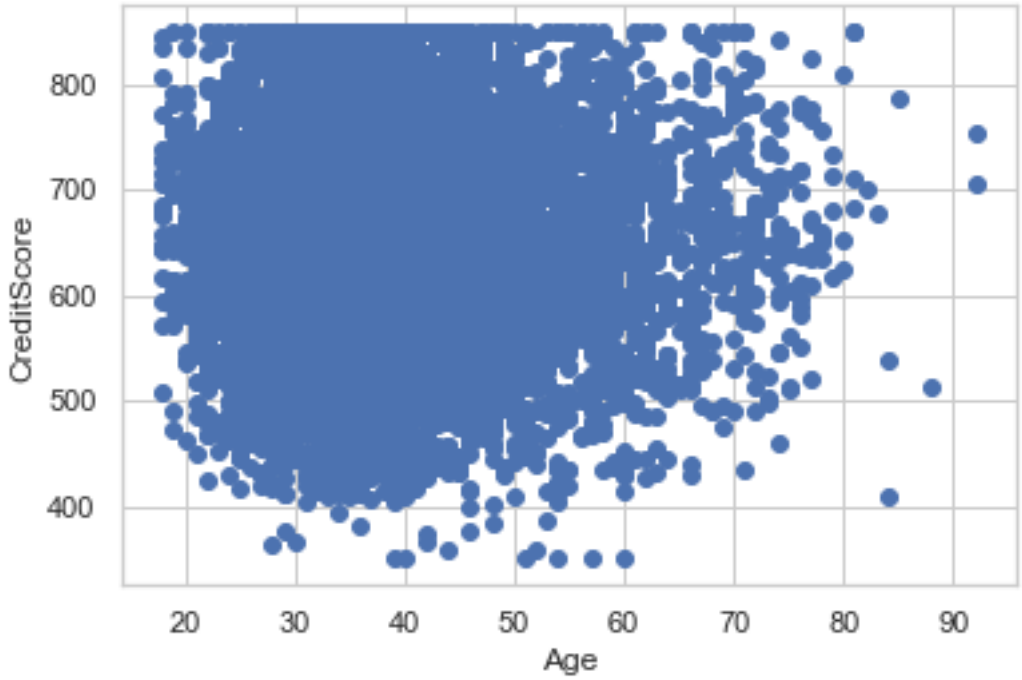
[23]:
```
ax = sns.boxplot(data = dataset[my_list], orient = 'h')
```



1.4 1.4. Scatter Plot

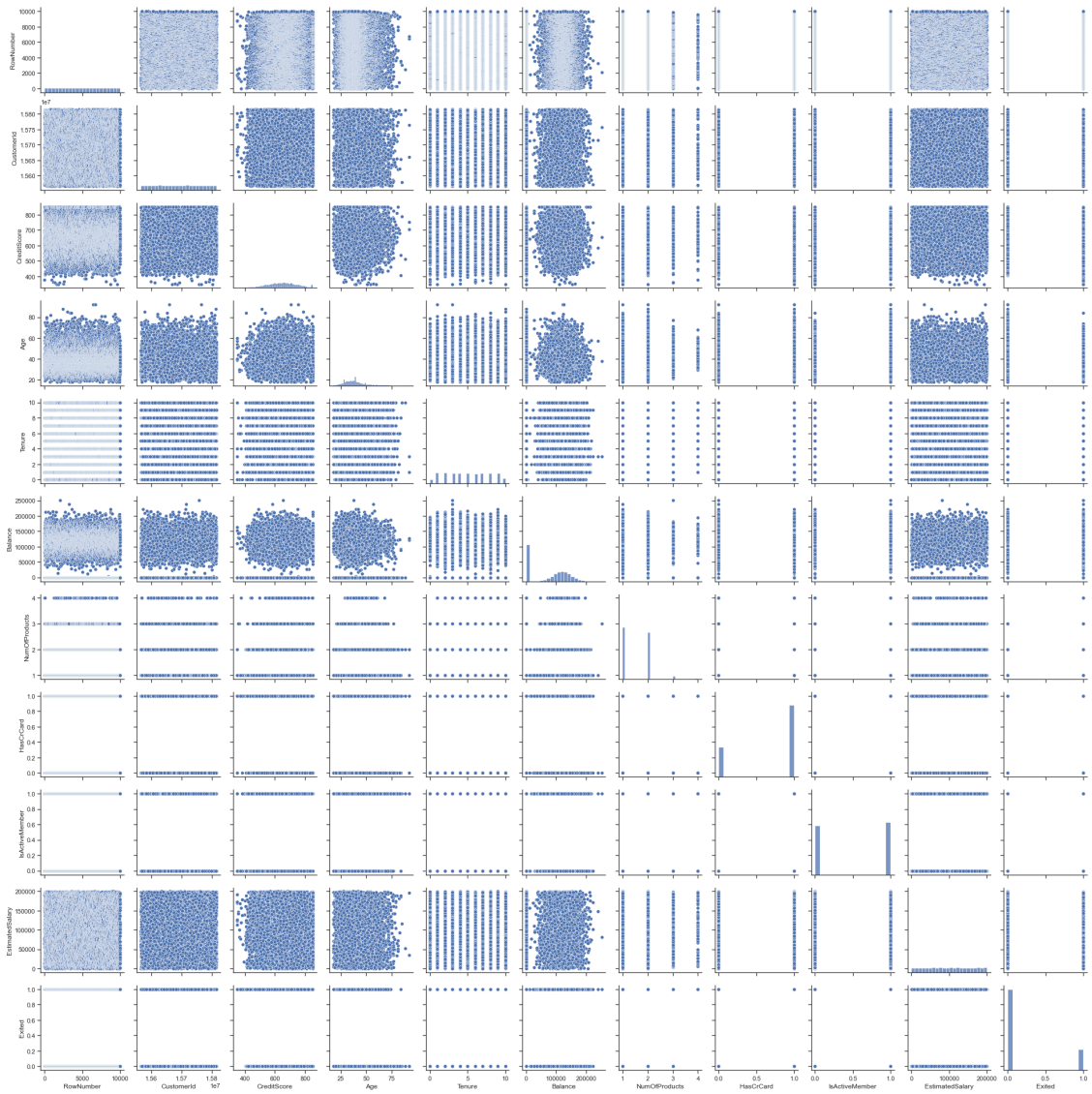
[24]:
```
plt.scatter(x=dataset['Age'],y=dataset['CreditScore'])
plt.xlabel('Age')
plt.ylabel('CreditScore')
```

[24]:
```
Text(0, 0.5, 'CreditScore')
```



[25]:
```
sns.set(style='ticks')
sns.pairplot(dataset)
```

[25]:
```
<seaborn.axisgrid.PairGrid at 0x7fdc3261ffd0>
```



1.5 1.5. Input variables and target

[26]:
```
X = dataset.iloc[:,3:13]
y = dataset.iloc[:,13]
```

[27]:
```
X.head()
```

[27]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	\
0	619	France	Female	42	2	0.00	1	
1	608	Spain	Female	41	1	83807.86	1	
2	502	France	Female	42	8	159660.80	3	

```
3          699    France  Female   39       1       0.00              2
4          850     Spain  Female   43       2  125510.82              1

   HasCrCard  IsActiveMember  EstimatedSalary
0          1               1        101348.88
1          0               1        112542.58
2          1               0        113931.57
3          0               0         93826.63
4          1               1         79084.10
```

[28]:
```
y.head()
```

[28]:
```
0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64
```

[29]:
```
type(X)
```

[29]:
```
pandas.core.frame.DataFrame
```

Convert DataFrame to Numpy Array

[30]:
```
X = X.values
y = y.values
```

[31]:
```
type(X)
```

[31]:
```
numpy.ndarray
```

[32]:
```
X[0:5,:]
```

[32]:
```
array([[619, 'France', 'Female', 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 'Spain', 'Female', 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 'France', 'Female', 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 'France', 'Female', 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 'Spain', 'Female', 43, 2, 125510.82, 1, 1, 1, 79084.1]],
      dtype=object)
```

1.6 1.6. Encoding categorical data

[33]:
```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
```

[34]:
```
X[:,1] = label.fit_transform(X[:,1])
X[:,2] = label.fit_transform(X[:,2])
```

[35]:
```
X[0:10,:]
```

[35]:
```
array([[619, 0, 0, 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 2, 0, 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 0, 0, 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 0, 0, 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 2, 0, 43, 2, 125510.82, 1, 1, 1, 79084.1],
       [645, 2, 1, 44, 8, 113755.78, 2, 1, 0, 149756.71],
       [822, 0, 1, 50, 7, 0.0, 2, 1, 1, 10062.8],
       [376, 1, 0, 29, 4, 115046.74, 4, 1, 0, 119346.88],
       [501, 0, 1, 44, 4, 142051.07, 2, 0, 1, 74940.5],
       [684, 0, 1, 27, 2, 134603.88, 1, 1, 1, 71725.73]], dtype=object)
```

Dummy encoding

[36]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

[37]:
```
X.shape
```

[37]:
```
(10000, 10)
```

[38]:
```
ct = ColumnTransformer([('Geography',OneHotEncoder(),[1])],\
                       remainder = 'passthrough')
```

[39]:
```
X = ct.fit_transform(X)
```

[40]:
```
X.shape
```

[40]:
```
(10000, 12)
```

[41]:
```
X[0:10,:]
```

[41]:
```
array([[1.0, 0.0, 0.0, 619, 0, 42, 2, 0.0, 1, 1, 1, 101348.88],
       [0.0, 0.0, 1.0, 608, 0, 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [1.0, 0.0, 0.0, 502, 0, 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [1.0, 0.0, 0.0, 699, 0, 39, 1, 0.0, 2, 0, 0, 93826.63],
       [0.0, 0.0, 1.0, 850, 0, 43, 2, 125510.82, 1, 1, 1, 79084.1],
       [0.0, 0.0, 1.0, 645, 1, 44, 8, 113755.78, 2, 1, 0, 149756.71],
       [1.0, 0.0, 0.0, 822, 1, 50, 7, 0.0, 2, 1, 1, 10062.8],
       [0.0, 1.0, 0.0, 376, 0, 29, 4, 115046.74, 4, 1, 0, 119346.88],
       [1.0, 0.0, 0.0, 501, 1, 44, 4, 142051.07, 2, 0, 1, 74940.5],
       [1.0, 0.0, 0.0, 684, 1, 27, 2, 134603.88, 1, 1, 1, 71725.73]],
      dtype=object)
```

[42]:
```
X = X[:,1:]
```

[43]:
```
X.shape
```

[43]:
```
(10000, 11)
```

[44]:
```
X[0:5,:]
```

[44]:
```
array([[0.0, 0.0, 619, 0, 42, 2, 0.0, 1, 1, 1, 101348.88],
       [0.0, 1.0, 608, 0, 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [0.0, 0.0, 502, 0, 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [0.0, 0.0, 699, 0, 39, 1, 0.0, 2, 0, 0, 93826.63],
       [0.0, 1.0, 850, 0, 43, 2, 125510.82, 1, 1, 1, 79084.1]],
      dtype=object)
```

[45]:
```
dataset.columns
```

[45]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

[46]:
```
features = ['Geography2','Geography3','CreditScore',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary']
```

[47]:
```
features
```

[47]:
```
['Geography2',
 'Geography3',
 'CreditScore',
 'Gender',
 'Age',
 'Tenure',
 'Balance',
 'NumOfProducts',
 'HasCrCard',
 'IsActiveMember',
 'EstimatedSalary']
```

1.7 1.7. Splitting dataset into train and test sets

[48]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,\
                                                 random_state=0)
```

[49]:
```
X_train.shape
```

[49]: (8000, 11)

[50]:
```
X_test.shape
```

[50]: (2000, 11)

[51]:
```
y.mean()
```

[51]: 0.2037

[52]:
```
print(y_train.mean())
print(y_test.mean())
```

```
0.204
0.2025
```

1.8 1.8. Feature scaling

[53]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

[54]:
```
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

[55]:
```
#from sklearn.preprocessing import MinMaxScaler
```

[56]:
```
X_train[0,:]
```

[56]: array([0.0, 1.0, 667, 0, 34, 5, 0.0, 2, 1, 0, 163830.64], dtype=object)

[57]:
```
X_train_sc[0,:]
```

[57]:
```
array([-0.5698444 ,  1.74309049,  0.16958176, -1.09168714, -0.46460796,
        0.00666099, -1.21571749,  0.8095029 ,  0.64259497, -1.03227043,
        1.10643166])
```

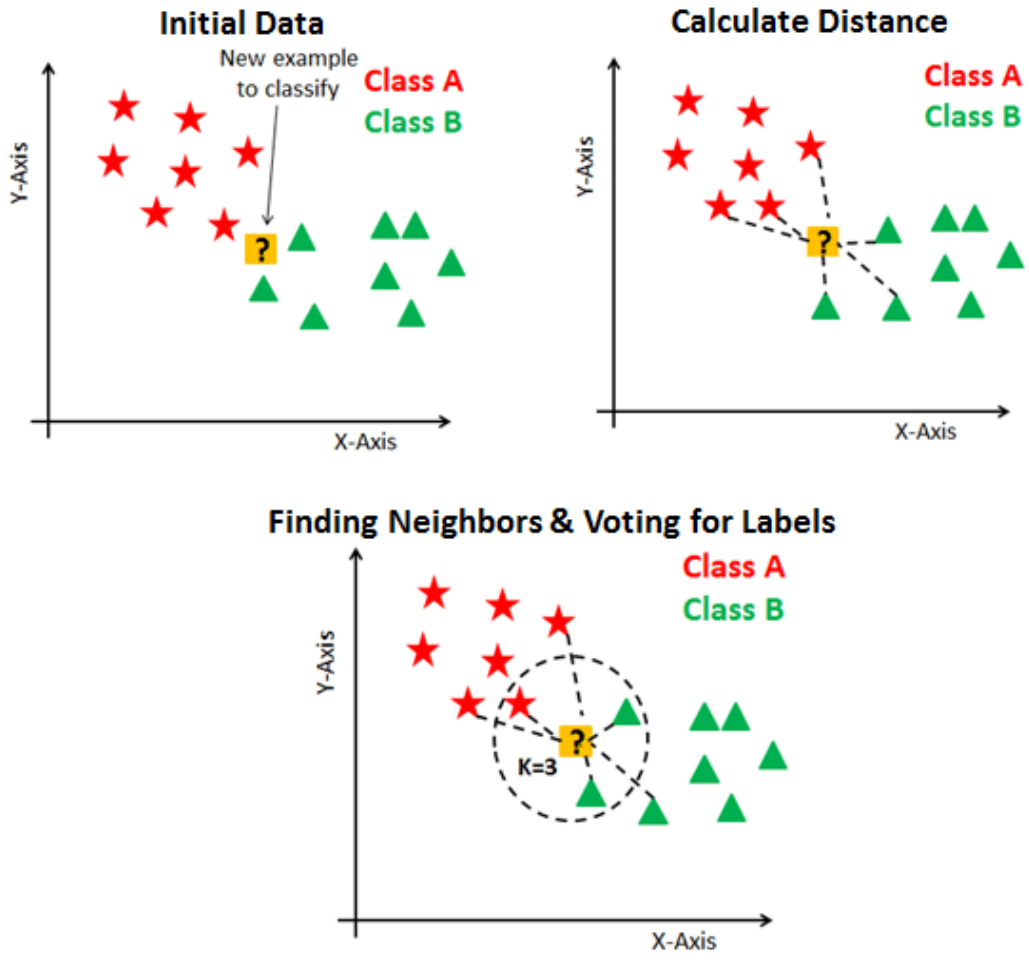
[58]:
```
(667 - X_train[:,2].mean())/(X_train[:,2].std())
```

[58]: 0.16958176236487257

2 2. k Nearest Neighbors

[59]:
```
Image('knn.png')
```

[59]:



[60]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
```

[61]:
```
knn.fit(X_train_sc,y_train)
```

[61]:
```
KNeighborsClassifier()
```

[62]:
```
y_pred_knn = knn.predict(X_test_sc)
```

[63]:
```
for i in range(10):
    print(y_test[i], y_pred_knn[i])
```

```
0 0
```

```
1 0
0 0
0 0
0 0
1 1
0 0
0 0
1 0
1 0
```

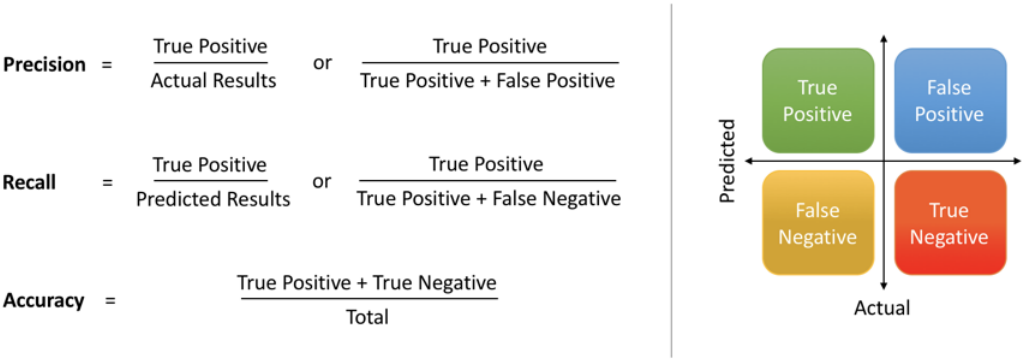
[64]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

[65]:
```
print('Confusion matrix knn \n', confusion_matrix(y_test,y_pred_knn))
print('Accuracy knn', accuracy_score(y_test,y_pred_knn))
```

```
Confusion matrix knn
 [[1486  109]
 [ 237  168]]
Accuracy knn 0.827
```

[66]:
```
Image('metrics.png')
```

[66]:



[67]:
```
from sklearn.metrics import precision_score, recall_score
```

[68]:
```
print('Recall knn', recall_score(y_test,y_pred_knn))
print('Precision knn', precision_score(y_test,y_pred_knn))
```

```
Recall knn 0.4148148148148148
Precision knn 0.6064981949458483
```

[69]:
```
from sklearn.metrics import classification_report
```

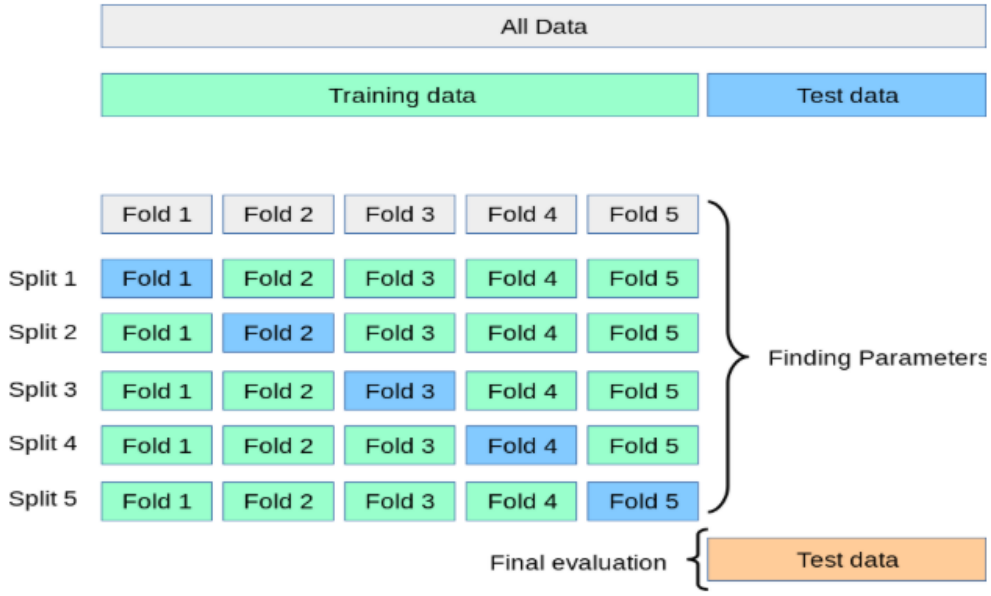
[70]:
```
print(classification_report(y_test,y_pred_knn))
```

	precision	recall	f1-score	support
0	0.86	0.93	0.90	1595
1	0.61	0.41	0.49	405
accuracy			0.83	2000
macro avg	0.73	0.67	0.69	2000
weighted avg	0.81	0.83	0.81	2000

Grid search

[71]:
```
Image('grid.png')
```

[71]:



[72]:
```
from sklearn.model_selection import GridSearchCV
parameters = {'n_neighbors' : [1,3,5,7,9,11,13]}
model = KNeighborsClassifier()
clf = GridSearchCV(model,parameters, scoring='accuracy', cv=5)
grille = clf.fit(X_train_sc,y_train)
print(grille.best_params_)
print(grille.best_score_)
```

```
{'n_neighbors': 9}
0.8303750000000001
```

[73]:
```
y_pred_knn_o = grille.predict(X_test_sc)
```

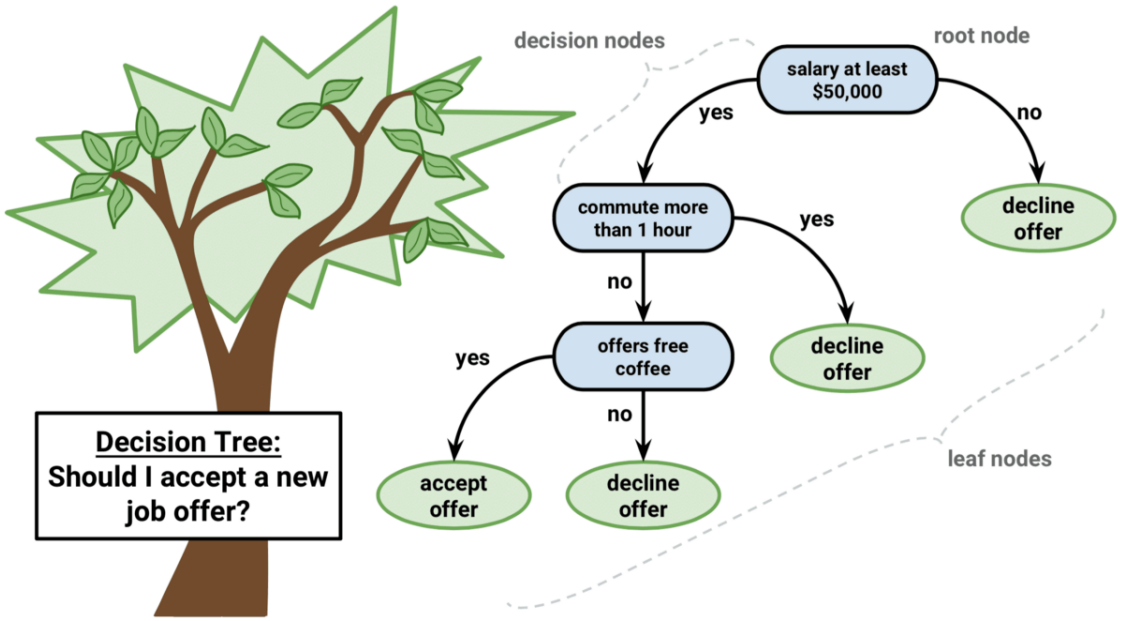
[74]:
```
print('Confusion matrix knn op \n', confusion_matrix(y_test,y_pred_knn_o))
print('Accuracy knn op', accuracy_score(y_test,y_pred_knn_o))
print('Recall knn op', recall_score(y_test,y_pred_knn_o))
print('Precision knn op', precision_score(y_test,y_pred_knn_o))
```

```
Confusion matrix knn op
 [[1516   79]
 [ 252  153]]
Accuracy knn op 0.8345
Recall knn op 0.37777777777777777
Precision knn op 0.6594827586206896
```

3 3. Decision Tree

[85]:
```
Image('decision.png')
```

[85]:



[75]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
```

[76]:
```
dt.fit(X_train,y_train)
```

[76]:
```
DecisionTreeClassifier()
```

[77]:
```
y_pred_dt = dt.predict(X_test)
```

[78]:
```
for i in range(10):
    print(y_test[i],y_pred_dt[i])
```

```
0 0
1 0
0 0
0 0
0 0
1 1
0 0
0 0
1 0
1 1
```

[79]:
```
print('Confusion matrix dt \n', confusion_matrix(y_test,y_pred_dt))
print('Accuracy dt', accuracy_score(y_test,y_pred_dt))
print('Recall dt', recall_score(y_test,y_pred_dt))
print('Precision dt', precision_score(y_test,y_pred_dt))
```

```
Confusion matrix dt
 [[1376  219]
 [ 173  232]]
Accuracy dt 0.804
Recall dt 0.5728395061728395
Precision dt 0.5144124168514412
```

[80]:
```
print(classification_report(y_test,y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88      1595
           1       0.51      0.57      0.54       405

    accuracy                           0.80      2000
   macro avg       0.70      0.72      0.71      2000
weighted avg       0.81      0.80      0.81      2000
```

Visualization

[81]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /opt/anaconda3/lib/python3.8/site-
packages (0.16)
```

[82]:
```
import graphviz
from sklearn import tree
from sklearn.tree import export_graphviz
```

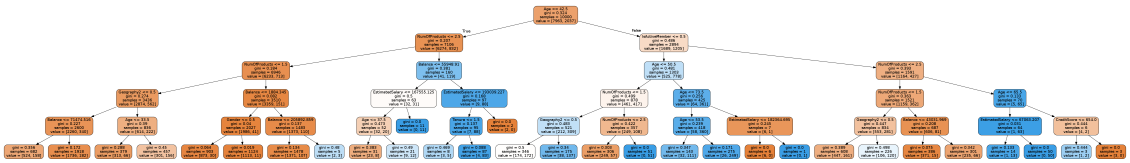
```
model = DecisionTreeClassifier(max_depth = 5)
model.fit(X,y)
```

[82]:
```
DecisionTreeClassifier(max_depth=5)
```

[83]:
```
tree.export_graphviz(model,feature_names = features,\
                     out_file = 'dt_cdoss.dot',\
                     label = 'all',\
                     filled = True,\
                     rounded = True)
```

[84]:
```
Image('dt_cdoss.png')
```

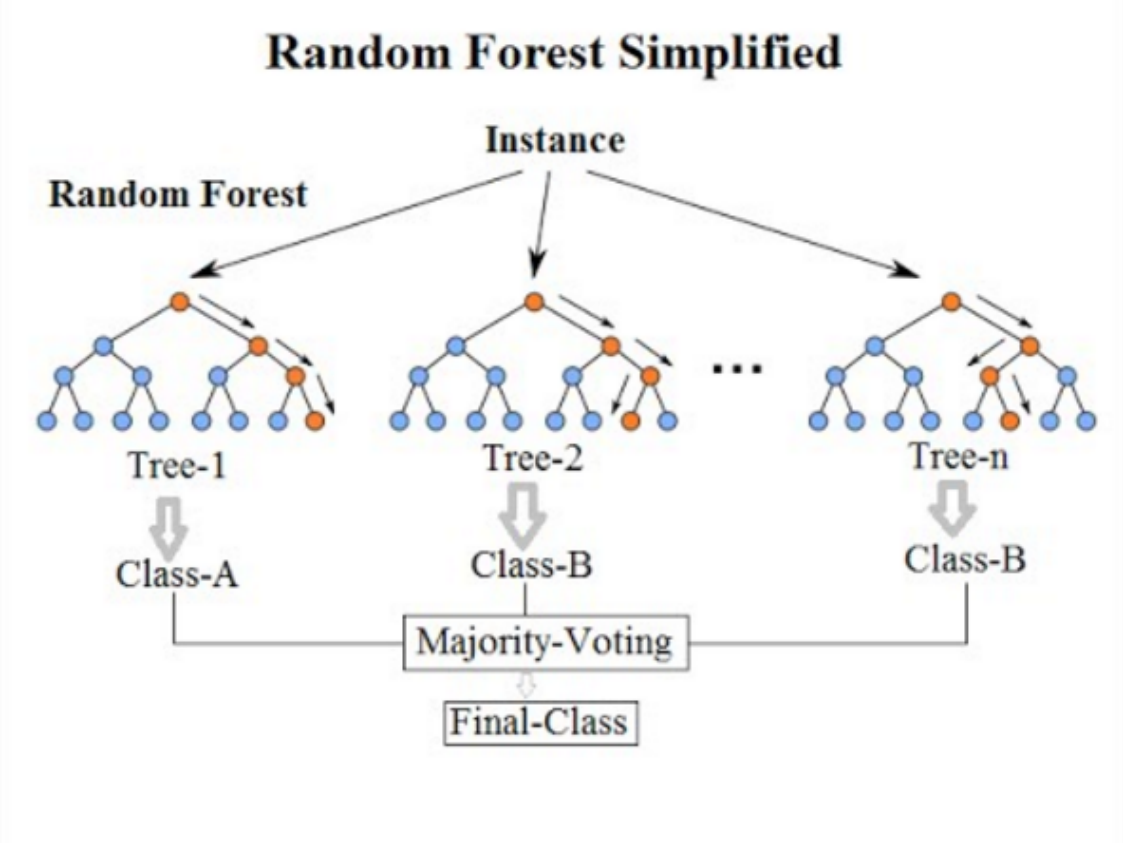
[84]:



4 4. Random Forest

[87]:
```
Image('random_forest.png')
```

[87]:



[90]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=500)
```

[91]:
```
rf.fit(X_train,y_train)
```

[91]:
```
RandomForestClassifier(n_estimators=500)
```

[92]:
```
y_pred_rf = rf.predict(X_test)
```

[93]:
```
print('Confusion matrix rf \n', confusion_matrix(y_test,y_pred_rf))
print('Accuracy rf', accuracy_score(y_test,y_pred_rf))
print('Recall rf', recall_score(y_test,y_pred_rf))
print('Precision rf', precision_score(y_test,y_pred_rf))
```

```
Confusion matrix rf
 [[1519   76]
 [ 193  212]]
Accuracy rf 0.8655
Recall rf 0.5234567901234568
Precision rf 0.7361111111111112
```

[94]:
```
print(classification_report(y_test,y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      1595
           1       0.74      0.52      0.61       405

    accuracy                           0.87      2000
   macro avg       0.81      0.74      0.77      2000
weighted avg       0.86      0.87      0.86      2000
```

5 5. Features selection

[95]:
```
importances = rf.feature_importances_
```

[96]:
```
importances
```

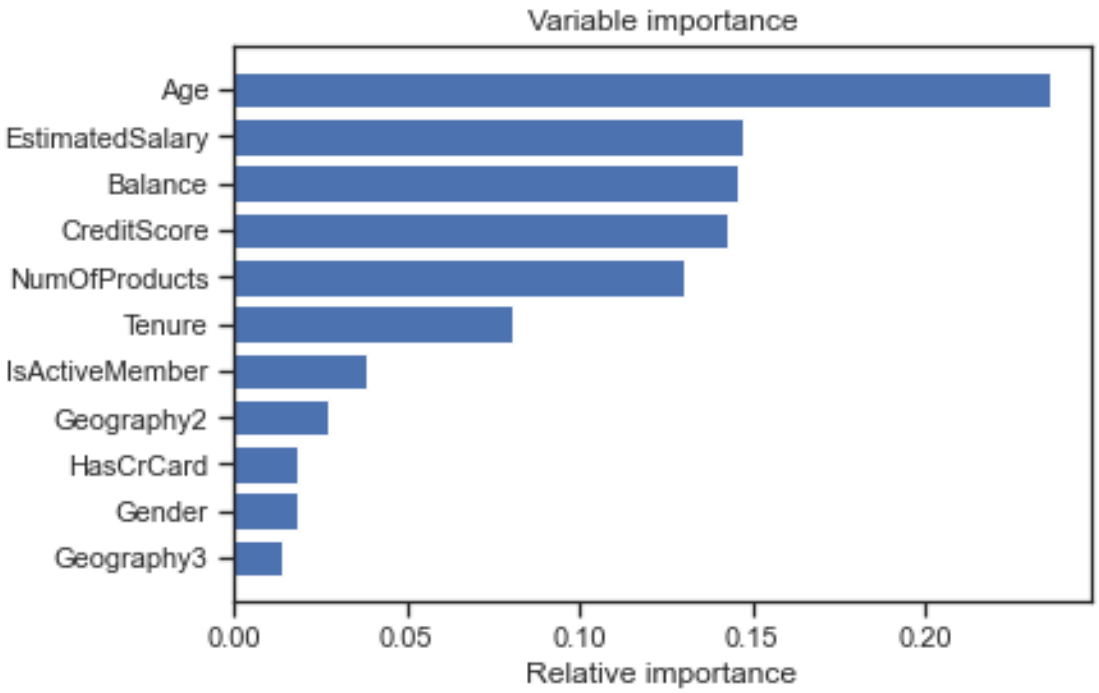
[96]:
```
array([0.0273628 , 0.0139989 , 0.14306505, 0.01853882, 0.23616594,
       0.08063451, 0.14554524, 0.13040445, 0.01861437, 0.03840497,
       0.14726493])
```

[97]:
```
sorted_idx = np.argsort(importances)
```

[98]:
```
features_arr = np.asarray(features)
```

[99]:
```
padding=np.arange(X_train_sc.size/len(X_train_sc))
plt.barh(padding,importances[sorted_idx],align='center')
plt.yticks(padding,features_arr[sorted_idx])
plt.xlabel('Relative importance')
plt.title('Variable importance')
```

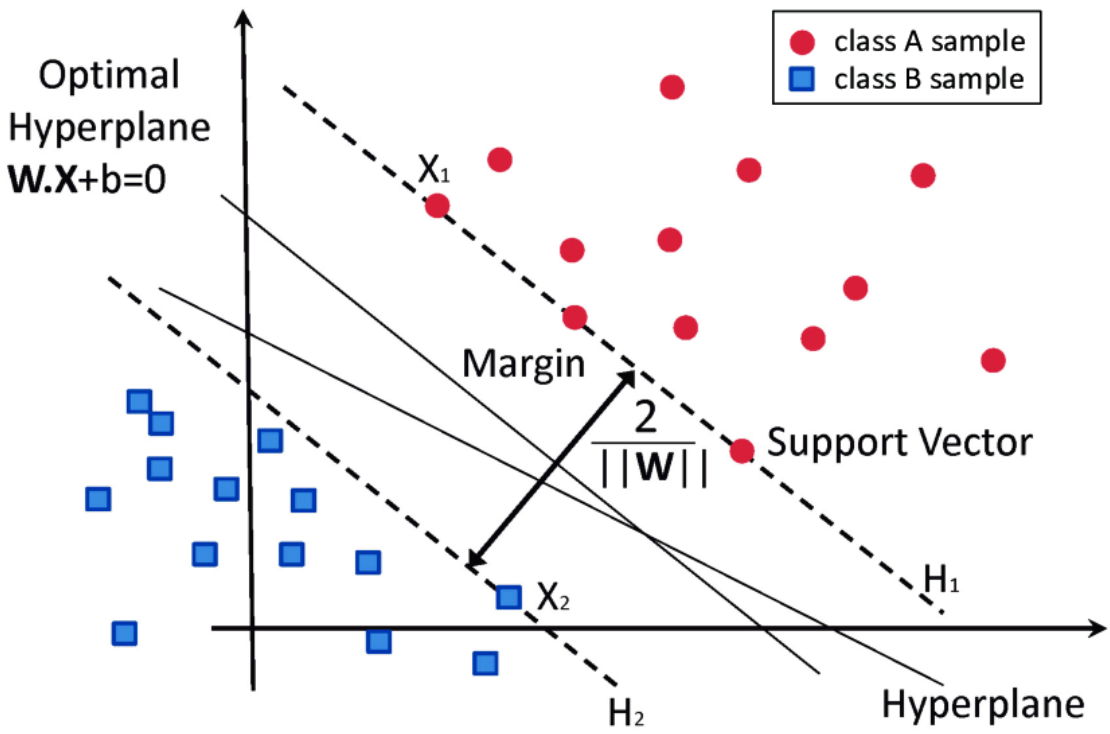
[99]:
```
Text(0.5, 1.0, 'Variable importance')
```



6 6. Support vector Machine

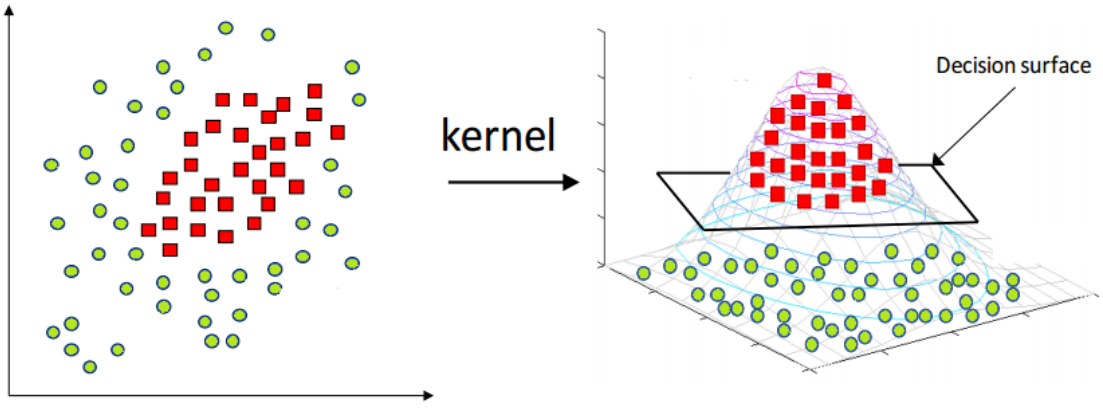
[100]:
```
Image('linear_svm.png')
```

[100]:



[101]:
```
Image('kernel_svm.png')
```

[101]:



[102]:
```
from sklearn.svm import SVC
linear_SVM = SVC(kernel='linear')
linear_SVM.fit(X_train_sc,y_train)
```

[102]:
```
SVC(kernel='linear')
```

[103]:
```
y_predictSVM_l = linear_SVM.predict(X_test_sc)
print(confusion_matrix(y_test,y_predictSVM_l))
print('Accuracy linear SVM {0:.3f}'.
 ↪format(accuracy_score(y_test,y_predictSVM_l)))
print('Precision linear SVM {0:.3f}'.
 ↪format(precision_score(y_test,y_predictSVM_l)))
print('Recall linear SVM {0:.3f}'.format(recall_score(y_test,y_predictSVM_l)))
```

```
[[1595    0]
 [ 405    0]]
Accuracy linear SVM 0.797
Precision linear SVM 0.000
Recall linear SVM 0.000
```

[104]:
```
print(classification_report(y_test,y_predictSVM_l))
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1595
           1       0.00      0.00      0.00       405

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.64      0.80      0.71      2000
```

[105]:
```
kernel_SVM = SVC(kernel='rbf')
kernel_SVM.fit(X_train_sc,y_train)
```

[105]:
```
SVC()
```

[106]:
```
y_predictSVM_k = kernel_SVM.predict(X_test_sc)
print(confusion_matrix(y_test,y_predictSVM_k))
print('Accuracy rbf SVM {0:.3f}'.format(accuracy_score(y_test,y_predictSVM_k)))
print('Precision rbf SVM {0:.3f}'.
 ↪format(precision_score(y_test,y_predictSVM_k)))
print('Recall rbf SVM {0:.3f}'.format(recall_score(y_test,y_predictSVM_k)))
```

```
[[1547   48]
 [ 225  180]]
Accuracy rbf SVM 0.864
Precision rbf SVM 0.789
Recall rbf SVM 0.444
```

[107]:
```
print(classification_report(y_test,y_predictSVM_k))
```

```
              precision    recall  f1-score   support
```

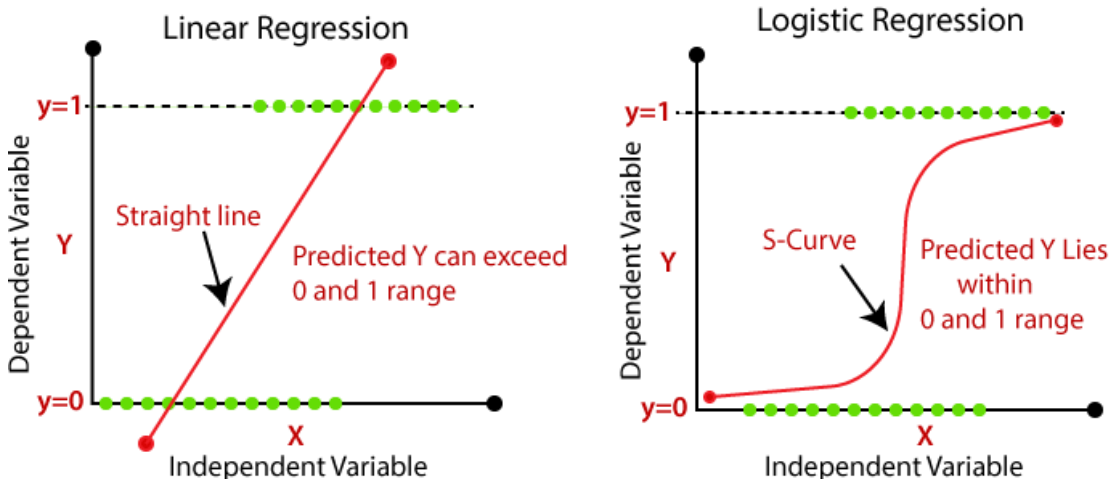
```
              0       0.87      0.97      0.92      1595
              1       0.79      0.44      0.57       405

    accuracy                            0.86      2000
   macro avg        0.83      0.71      0.74      2000
weighted avg        0.86      0.86      0.85      2000
```

7 7. Logistic Regression

[108]:
```
Image(filename='logistic.png')
```

[108]:



[109]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(X_train_sc,y_train)
```

[109]:
```
LogisticRegression()
```

[110]:
```
y_predictLR = LR.predict(X_test_sc)
print(confusion_matrix(y_test,y_predictLR))
print('Accuracy Logistic Regression {0:.3f}'.
 ↪format(accuracy_score(y_test,y_predictLR)))
print('Precision Logistic Regression {0:.3f}'.
 ↪format(precision_score(y_test,y_predictLR)))
print('Recall Logistic Regression {0:.3f}'.
 ↪format(recall_score(y_test,y_predictLR)))
```

```
[[1526   69]
 [ 309   96]]
```

```
Accuracy Logistic Regression 0.811
Precision Logistic Regression 0.582
Recall Logistic Regression 0.237
```

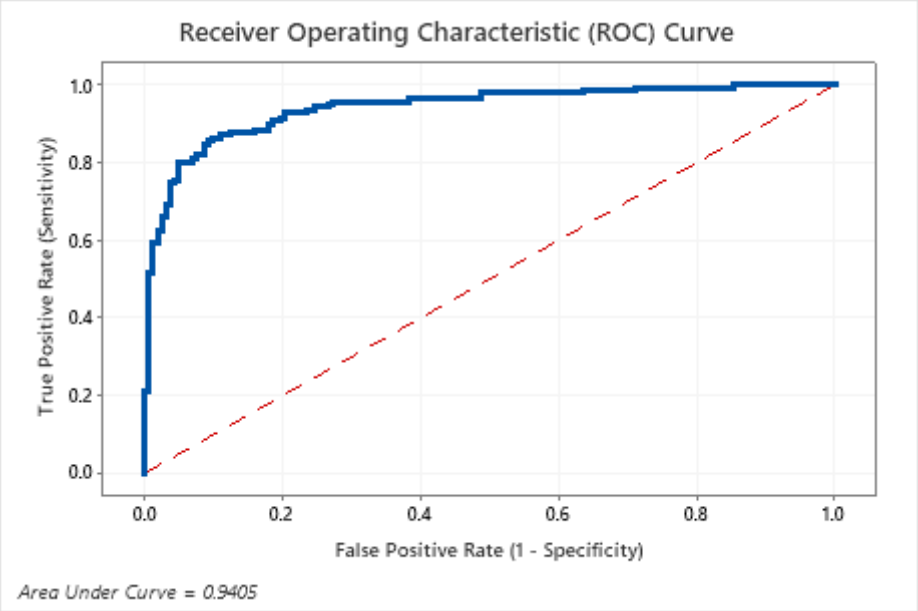
8 8. Summary

Algorithm	Accuracy	Recall	Precision
Optimized kNN	0.834	0.377	0.659
Decision Tree	0.804	0.572	0.514
Random Forest	0.865	0.523	0.736
linear SVM	0.797	0.000	0.000
kernel SVM	0.864	0.444	0.789
Logistic Regression	0.811	0.237	0.582

9 9. Receiver Operating Characteristic (ROC) Curve

[111]:
```
Image(filename='roc.png')
```

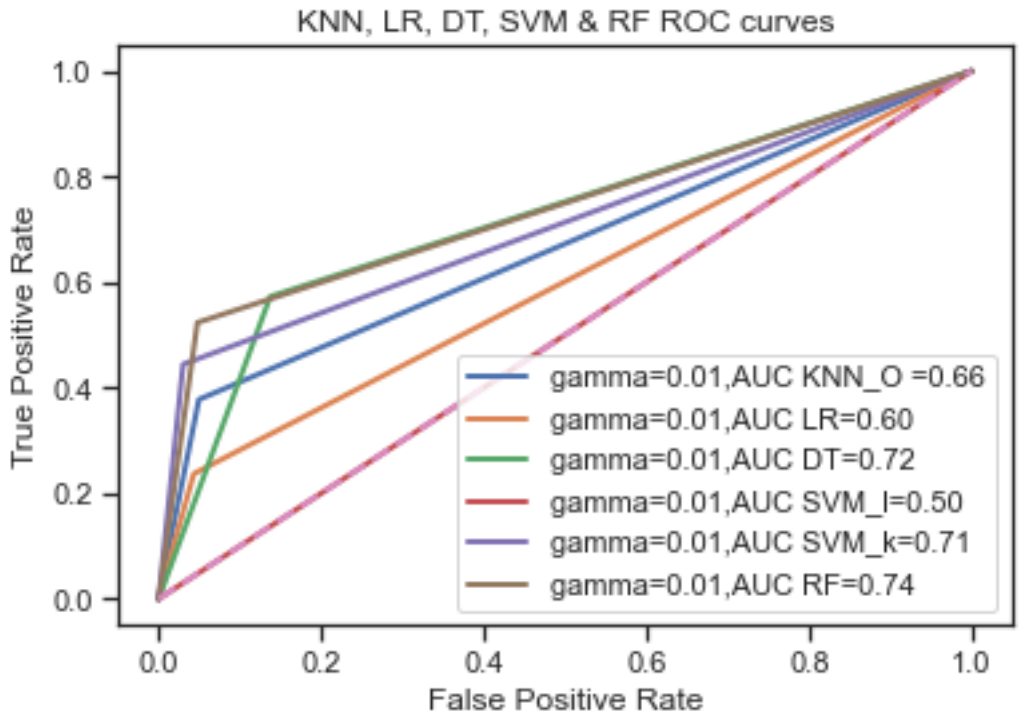
[111]:



[113]:
```
from sklearn import metrics
fpr,tpr,thr = metrics.roc_curve(y_test,y_pred_knn_o)
fpr2,tpr2,thr2 = metrics.roc_curve(y_test,y_predictLR)
fpr3,tpr3,thr3 = metrics.roc_curve(y_test,y_pred_dt)
fpr4,tpr4,thr4 = metrics.roc_curve(y_test,y_predictSVM_l)
```

```
fpr5,tpr5,thr5 = metrics.roc_curve(y_test,y_predictSVM_k)
fpr6,tpr6,thr6 = metrics.roc_curve(y_test,y_pred_rf)
auc = metrics.auc(fpr,tpr)
auc2 = metrics.auc(fpr2,tpr2)
auc3 = metrics.auc(fpr3,tpr3)
auc4 = metrics.auc(fpr4,tpr4)
auc5 = metrics.auc(fpr5,tpr5)
auc6 = metrics.auc(fpr6,tpr6)
plt.plot(fpr,tpr,'-',lw=2,label='gamma=0.01,AUC KNN_O =%.2f'%auc)
plt.plot(fpr2,tpr2,'-',lw=2,label='gamma=0.01,AUC LR=%.2f'%auc2)
plt.plot(fpr3,tpr3,'-',lw=2,label='gamma=0.01,AUC DT=%.2f'%auc3)
plt.plot(fpr4,tpr4,'-',lw=2,label='gamma=0.01,AUC SVM_l=%.2f'%auc4)
plt.plot(fpr5,tpr5,'-',lw=2,label='gamma=0.01,AUC SVM_k=%.2f'%auc5)
plt.plot(fpr6,tpr6,'-',lw=2,label='gamma=0.01,AUC RF=%.2f'%auc6)
plt.plot([0, 1], [0, 1], lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN, LR, DT, SVM & RF ROC curves')
plt.legend(loc='lower right')
```

[113]: <matplotlib.legend.Legend at 0x7fdc1d2fccd0>



Random Forest est le meilleur algorithme

[ ]: