PROJECT 07 : SUPPORT VECTOR MACHINE (SVM) FAMILY OF ALGORITHMS

BULDUK EKER, KACI BOURGUA

06/03/2023

IRONHACK

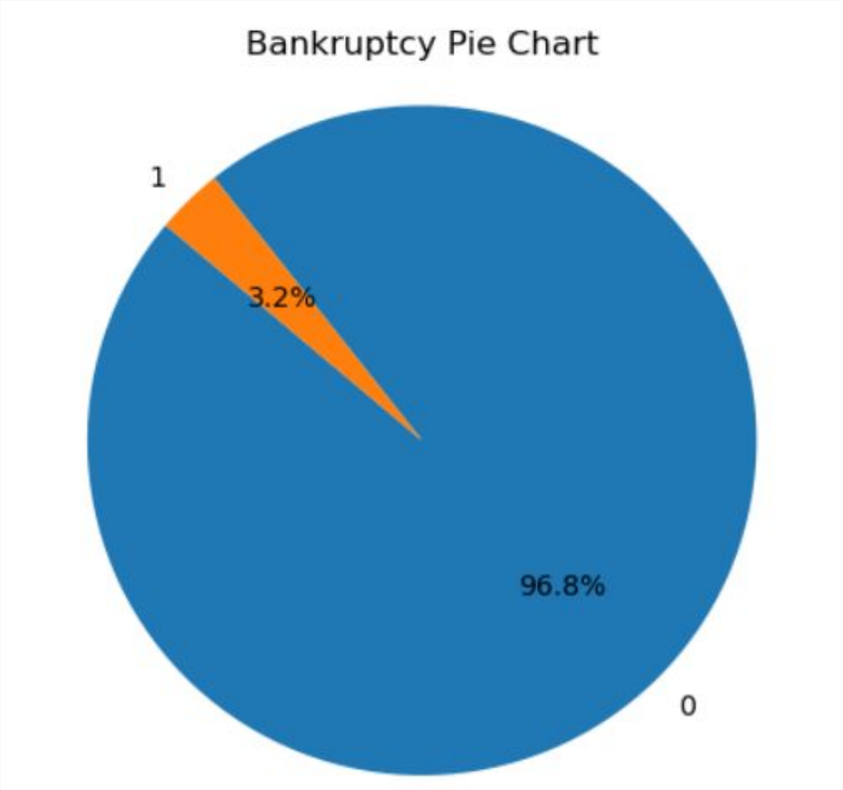
DATA CLEANING

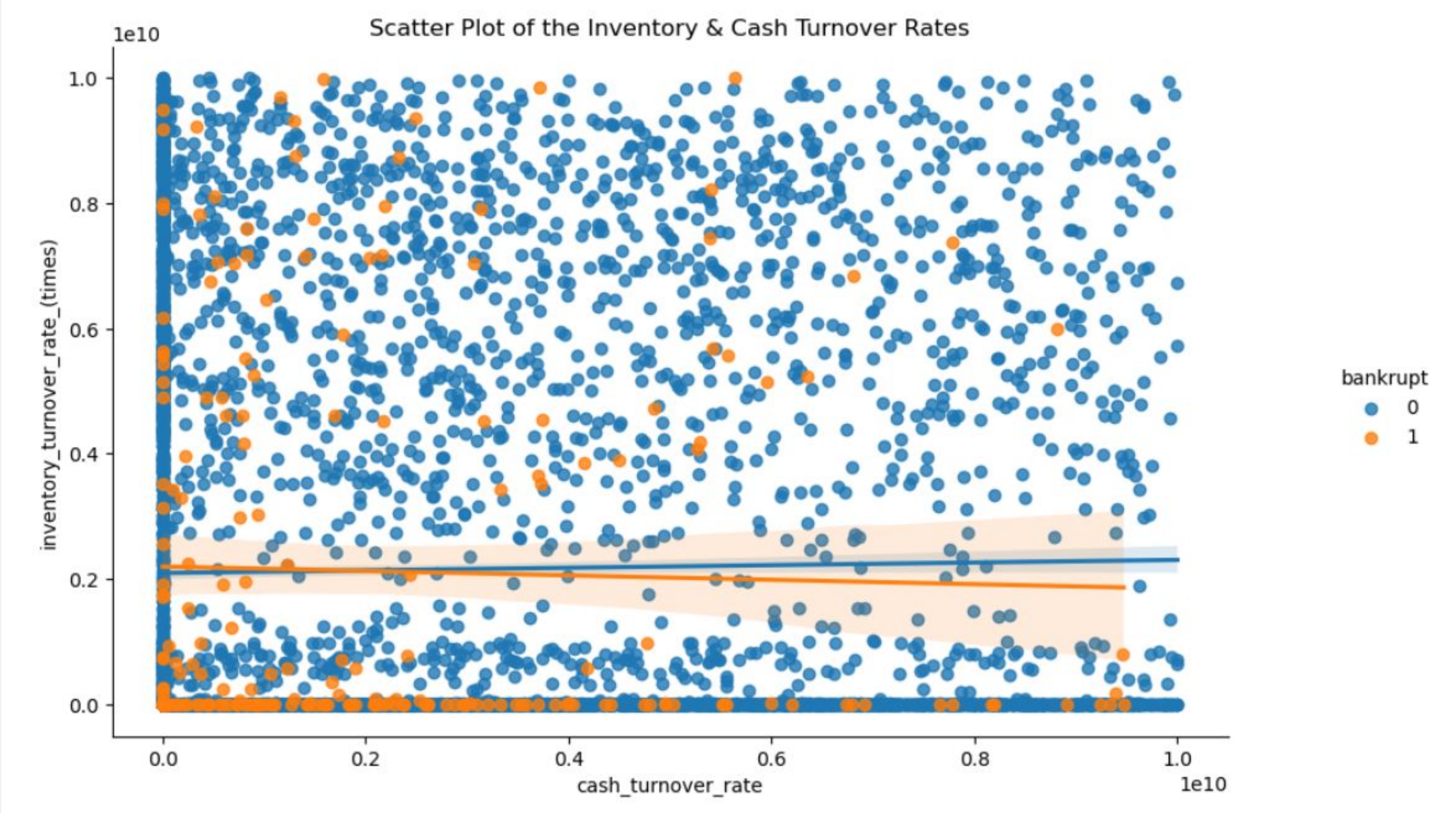
- Column names
- Check for null values (none)
- Check for duplicates
- Check for low variance columns (non)
- Check for collinearity : from 96 to 75 columns
- Check for duplicates : none

DATA EXPLORATION

- Numerical data only
- Target value : bankrupt or not (1-0)
- 96 columns, most of them already scaled

DATA EXPLORATION





DATA PREPARATION

- Splitting into training and testing dataset (80/20)
- Upsampling with SMOTE (Synthetic Minority Oversampling TEchnique)
  ⇒ 96.57% of 0
  ⇒ 3.42% of 1
- Scaled the data using Standard Scaler

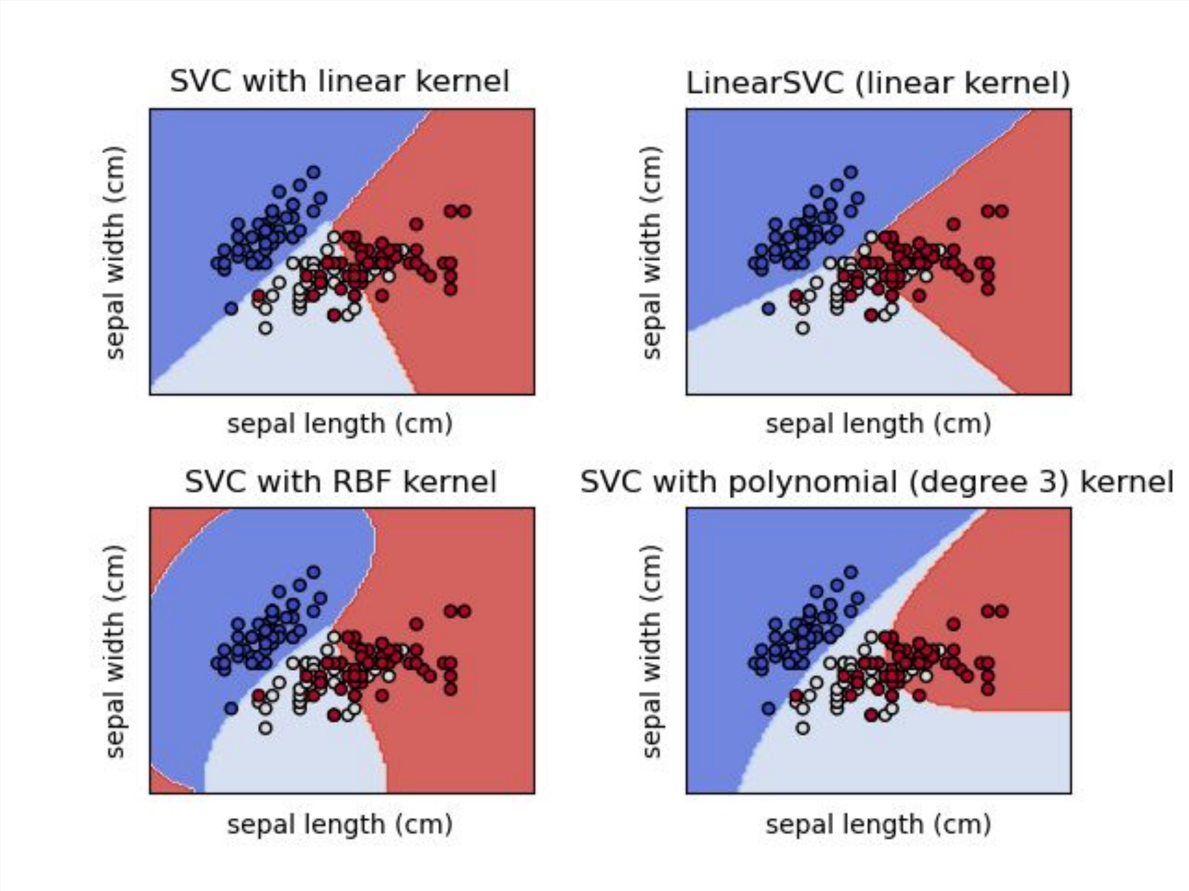
EXPLANATION OF THE SUPPORT VECTOR MACHINE FAMILY OF ALGORITHMS

- Type of ML algorithm used for classification tasks in which the goal is to separate two or more classes of data

- SVM in concept : it tries to draw a line that separates the categories. This line is called a "hyperplane". The goal of the SVM model is to find the best hyperplane that separates the categories as accurately as possible

- How : the SVM model looks at the data points closest to the hyperplane. These data points are called "support vectors". The SVM model tries to find the hyperplane that maximizes the distance between the support vectors and the hyperplane

EXPLANATION OF THE SUPPORT VECTOR MACHINE FAMILY OF ALGORITHMS



- Distance based algorithm that uses different kernel functions to map observations into a higher-dimensional space where it can be more easily separated into different classes (instead of the original feature space like Euclidean/Manhattan)

- The choice of kernel function can have a significant impact on the performance of the SVM model

EXPLANATION OF THE SUPPORT VECTOR MACHINE FAMILY OF ALGORITHMS

Linear Support Vector Classification

- Kernel : linear
- C parameter
- Often used for large datasets and can be faster to train than other SVM models

C-Support Vector Classification

- Kernel : different types like polynomial, radial basis function (RBF)
- C parameter
- Good choice when the number of features is not very high and the data can be separated by a non-linear boundary.

Nu-Support Vector Classification

- Kernel : different types of kernels, but it has a slightly different optimization problem than SVC
- Nu parameter
- Can be more flexible than SVC and can be used when the dataset is small or noisy

Model 1 : C-SVC

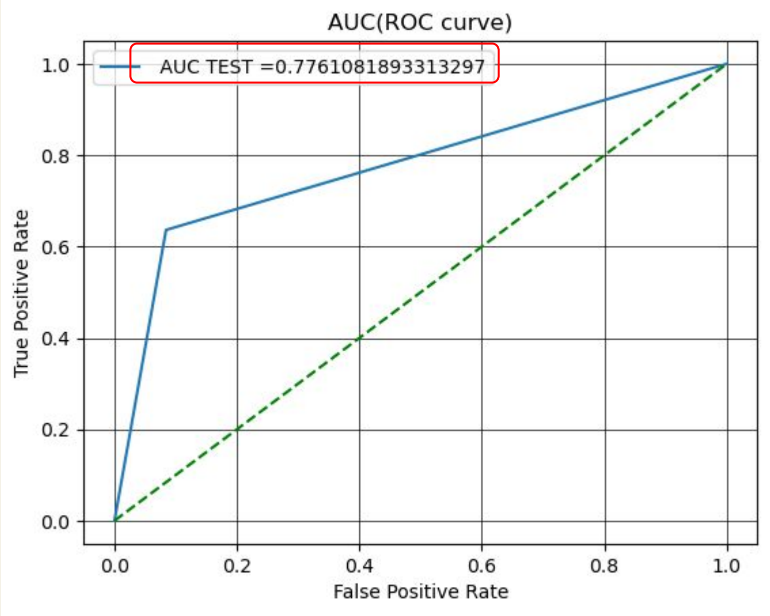
class `sklearn.svm.SVC`(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

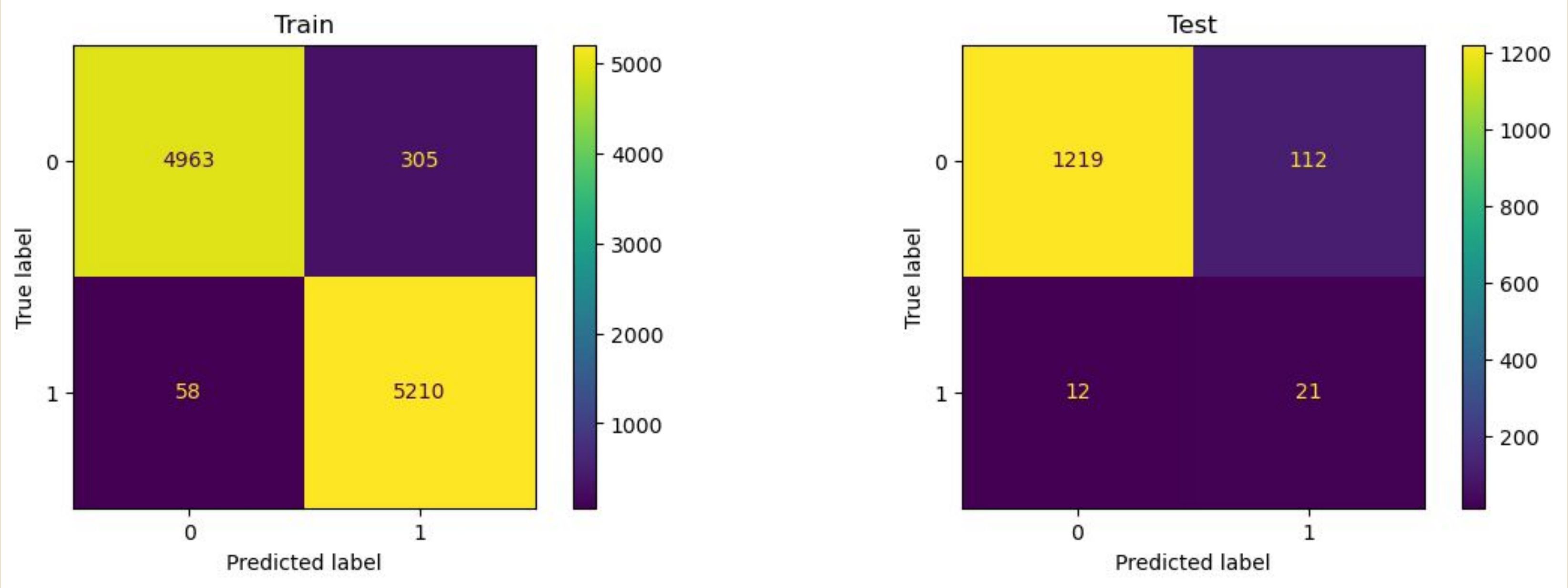
- SVC is a model that can use a linear or non-linear kernel that tries to find the best hyperplane in this higher-dimensional space that separates the different classes of data
- A kernel is function that takes two data points as input and computes a similarity measure between them

C-SVC : Before Hyper-parameter tuning

```
model_svc = SVC()
model_svc.fit(X_train_mod, y_sm)
score = model_svc.score(X_train_mod, y_sm)
print("Score: ", score)
```

```
Score:  0.9655466970387244
```





TRAIN:	precision	recall	f1-score	support
0	0.99	0.94	0.96	5268
1	0.94	0.99	0.97	5268
accuracy			0.97	10536
macro avg	0.97	0.97	0.97	10536
weighted avg	0.97	0.97	0.97	10536

TEST:	precision	recall	f1-score	support
0	0.99	0.92	0.95	1331
1	0.16	0.64	0.25	33
accuracy			0.91	1364
macro avg	0.57	0.78	0.60	1364
weighted avg	0.97	0.91	0.93	1364

Model 2 : Linear SVC

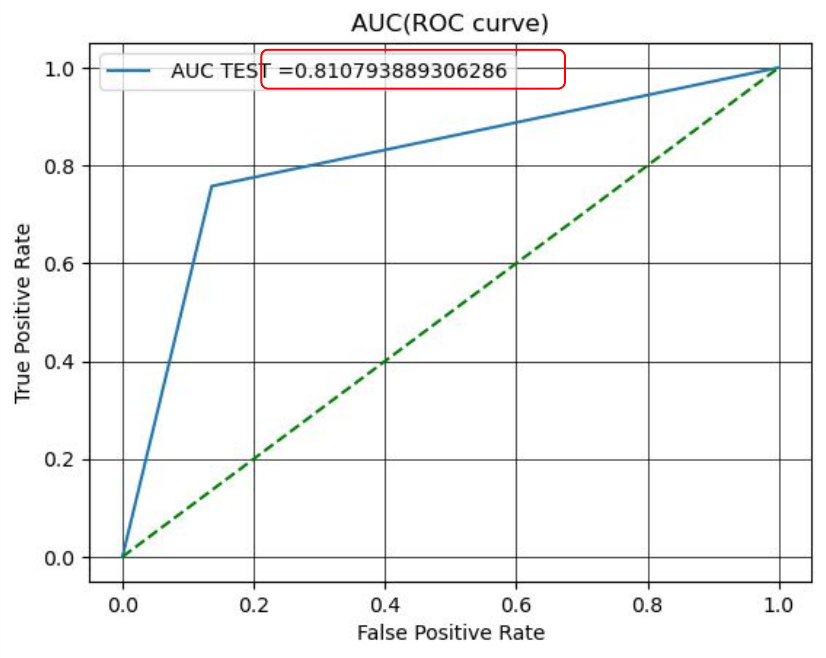
```
class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000) ¶    [source]
```

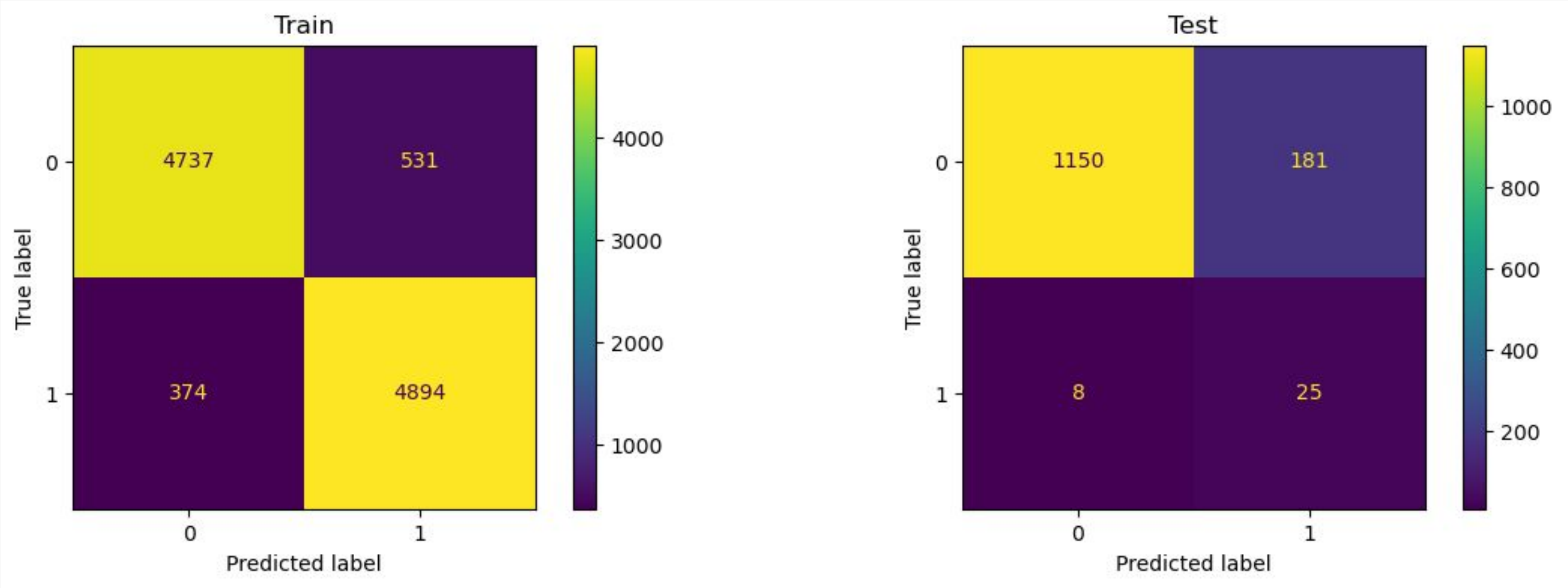
- LinearSVC : Similar to SVC with linear kernel
- With more flexibility in the choice of penalties (a regularization that prevent models from fitting the noise i.e. overfitting) and loss functions.
- It scales better to large numbers of samples

Linear SVC : Before Hyper-parameter tuning

```
lsvc.fit(X_train_mod, y_sm)
score = lsvc.score(X_train_mod, y_sm)
print("Score: ", score)
```

```
Score:  0.9141040242976461
```






```
TRAIN:          precision    recall  f1-score   support

           0       0.93      0.90      0.91      5268
           1       0.90      0.93      0.92      5268

    accuracy                           0.91     10536
   macro avg       0.91      0.91      0.91     10536
weighted avg       0.91      0.91      0.91     10536
```

```
TEST:           precision    recall  f1-score   support

           0       0.99      0.86      0.92      1331
           1       0.12      0.76      0.21        33

    accuracy                           0.86      1364
   macro avg       0.56      0.81      0.57      1364
weighted avg       0.97      0.86      0.91      1364
```

C-SVC & Linear SVC : After Hyper-parameter tuning

- Run on C-SVC as it includes a linear kernel similar to LinearSVC

```
param_grid = {'C': [0.1,1, 10],'kernel':["linear", "poly", "rbf", "sigmoid"], 'gamma': [1,0.1,0.01]}

grid = GridSearchCV(SVC(),param_grid,cv=3,return_train_score=True,refit=True,verbose=2)
grid.fit(X_train_mod,y_sm)
```

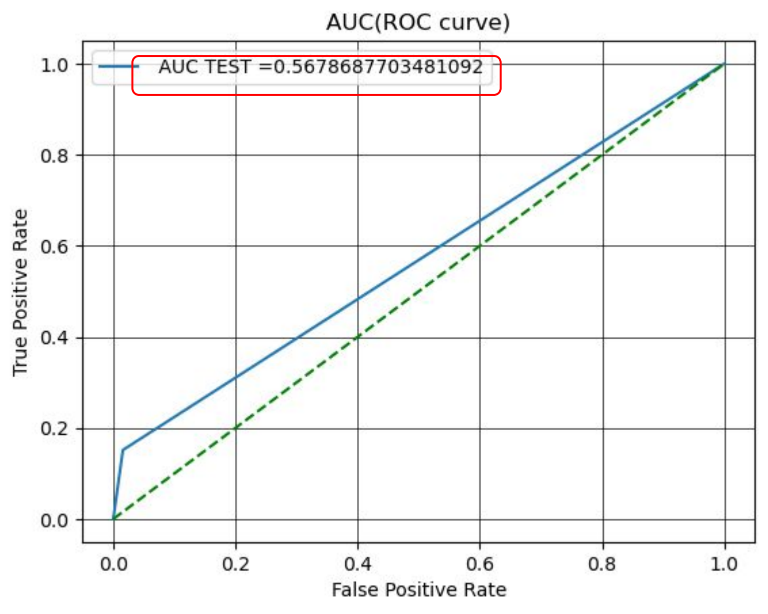
```
best_params = grid.best_params_ #To check the best set of parameters returned
best_params
```

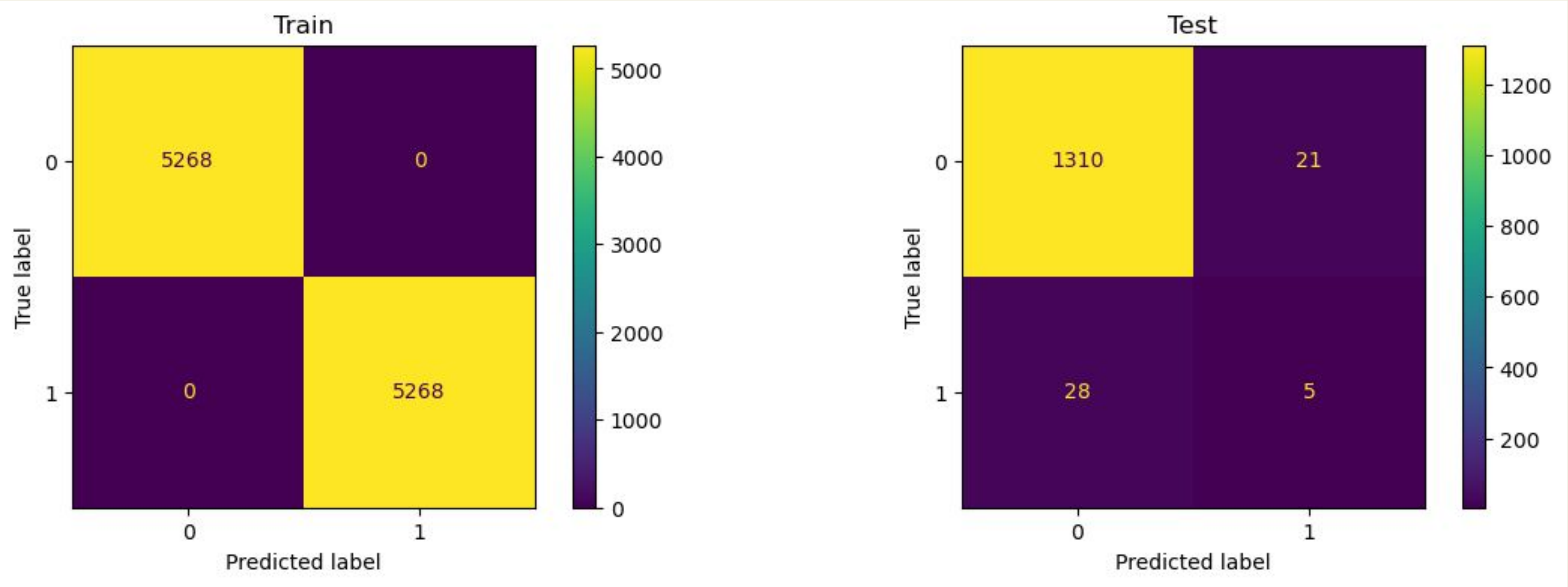
```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

C-SVC & Linear SVC : After Hyper-parameter tuning

```
model_svc_best = SVC(C=10.0,gamma=0.1,kernel= 'rbf')
model_svc_best.fit(X=X_train_mod, y=y_sm)
score = model_svc_best.score(X_train_mod, y_sm)
print("Score: ", score)
```

```
Score:  1.0
```






```
TRAIN:                precision    recall  f1-score   support

           0       1.00      1.00      1.00      5268
           1       1.00      1.00      1.00      5268

    accuracy                           1.00     10536
   macro avg       1.00      1.00      1.00     10536
weighted avg       1.00      1.00      1.00     10536
```

```
TEST:                 precision    recall  f1-score   support

           0       0.98      0.98      0.98      1331
           1       0.19      0.15      0.17        33

    accuracy                           0.96      1364
   macro avg       0.59      0.57      0.58      1364
weighted avg       0.96      0.96      0.96      1364
```

Nu-SVC : Model

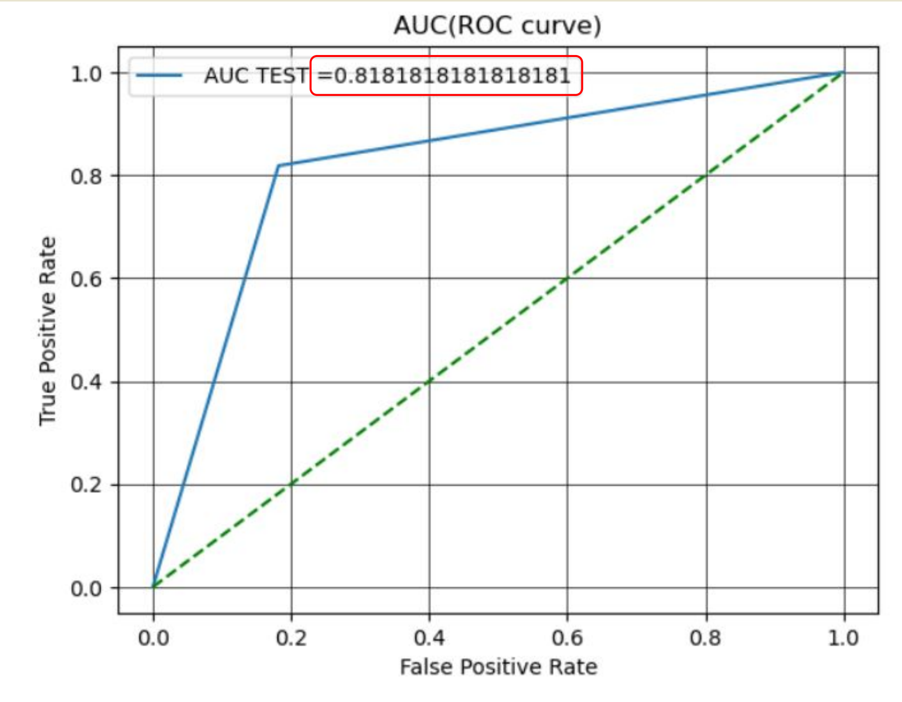
```
class sklearn.svm.NuSVC(*, nu=0.5, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False,
tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False,
random_state=None)
```
[source]

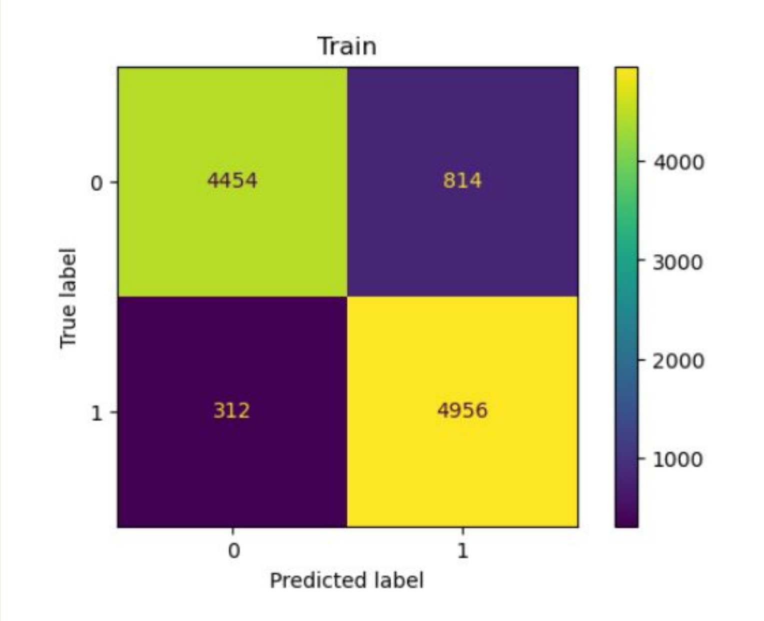
- Similar to SVC but uses a parameter to control the number of support vectors.
- NuSVC is similar to SVC in that it can use different types of kernels, but it has a slightly different optimization problem than SVC. NuSVC can be more flexible than SVC and can be used when the dataset is small or noisy

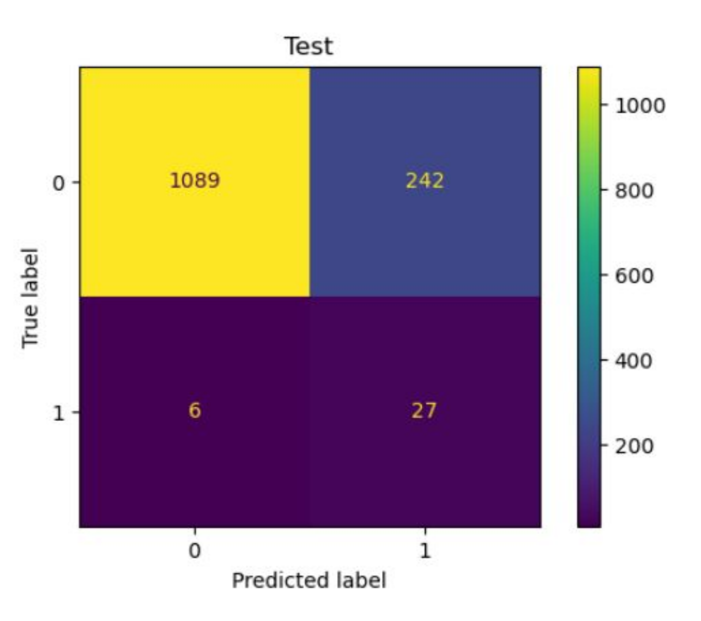
Nu-SVC : Before Hyper-parameter tuning

```python
# Train and test the NuSVC model
nu_svc = NuSVC()
nu_svc.fit(X_train_mod, y_sm)
score = nu_svc.score(X_train_mod, y_sm)
print("Score: ", score)
```

Score: 0.8931283219438116







TRAIN:

	precision	recall	f1-score	support
0	0.93	0.85	0.89	5268
1	0.86	0.94	0.90	5268
accuracy			0.89	10536
macro avg	0.90	0.89	0.89	10536
weighted avg	0.90	0.89	0.89	10536

TEST:

	precision	recall	f1-score	support
0	0.99	0.82	0.90	1331
1	0.10	0.82	0.18	33
accuracy			0.82	1364
macro avg	0.55	0.82	0.54	1364
weighted avg	0.97	0.82	0.88	1364

Nu-SVC : Hyper-parameter tuning

```
param_grid = {'nu': [0,0.5,1],'kernel':["linear", "poly", "rbf", "sigmoid"], 'gamma':["scale", "auto"]}

grid = GridSearchCV(NuSVC(),param_grid,cv=3,return_train_score=True,refit=True,verbose=2)
grid.fit(X_train_mod,y_sm)
```

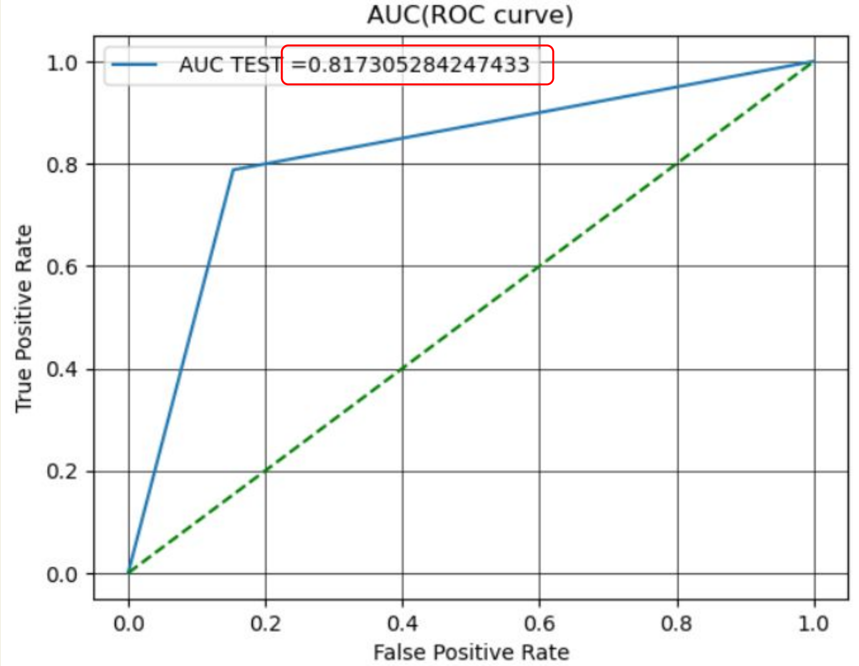
```
best_params_nu = grid.best_params_ #To check the best set of parameters returned
best_params_nu
```

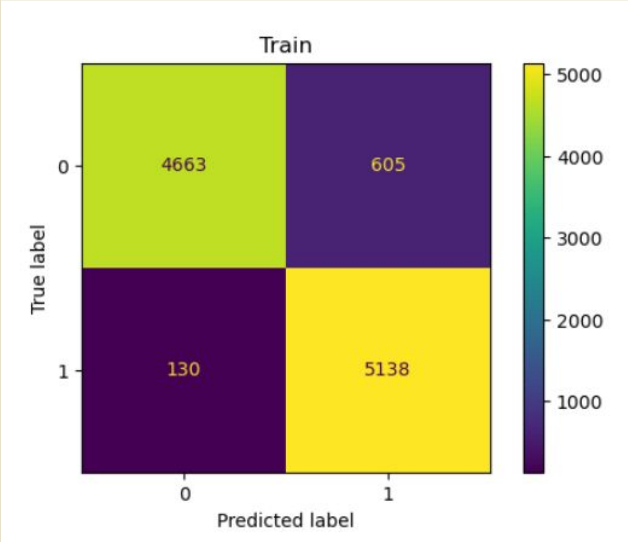
```
{'gamma': 'scale', 'kernel': 'poly', 'nu': 0.5}
```

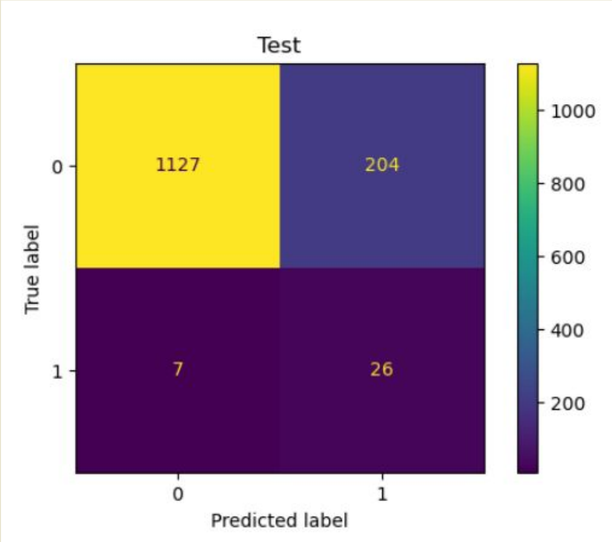
Nu-SVC : After hyper-parameter tuning

```
model_svc_best_nu = NuSVC(gamma='scale', kernel= 'poly', nu=0.5)
model_svc_best_nu.fit(X=X_train_mod, y=y_sm)
score = model_svc_best_nu.score(X_train_mod, y_sm)
print("Score: ", score)
```

Score: 0.9302391799544419







TRAIN:	precision	recall	f1-score	support
0	0.97	0.89	0.93	5268
1	0.89	0.98	0.93	5268
accuracy			0.93	10536
macro avg	0.93	0.93	0.93	10536
weighted avg	0.93	0.93	0.93	10536

TEST:	precision	recall	f1-score	support
0	0.99	0.85	0.91	1331
1	0.11	0.79	0.20	33
accuracy			0.85	1364
macro avg	0.55	0.82	0.56	1364
weighted avg	0.97	0.85	0.90	1364

Conclusions

- Among the 3, C-SVC is the model with better precision, nuSVC is better at recall metric
- Although all models showed limited precision anyway, despite good performance on train sets
- Accuracy is strong, only because the model is good at predicting most of the non-bankrupt rows

- Hyperparameter tuning has an only little improvement in accuracy and affects the recall

- Model seems not adapted to this dataset for predicting bankruptcy

Improvements

- Hyperparameter tuning of the SMOTE
- Feature importance
- Normalize the data BEFORE the SMOTE ? Yes because SMOTE is based on KNN (distance based) so need to normalize before
- Test different normalization methods like PowerTransformer

Thank you for listening !

Do you have any question ?