Data Analysis and Machine-Learning

Chapter 6.4:

Support Vector Machine (4)

Applications Part.2: Logistic Regression and SVM



Now that we have understood the (1) mathematical and algorithmic frameworks of the SVM and (2) specific parameters, let us apply SVM for modelling with actual datasets.

In actual work settings for data scientists and analysts, variables are often provided without specific names or titles for security reasons (due to privacy law, personal information, company privacies, etc.). This is why it is important for data scientists to be able to model solely based on numerical values and statistical estimations, without given backgrounds on the data per se.

In line with such reality, we will be analyzing the real dataset from Kaggle (<https://www.kaggle.com>, Don’t Overfit II) for actual comparative applications of logistic regression and SVM. Once you download the data from Kaggle, you shall see that while there are numerous numbers of variables, all data are provided without variable names, and there might be no immediate insight to be drawn from statistics (there are actually more numbers of variables than the numbers of data).

#Import Data, define features, target

import numpy as np

import pandas as pd

rawData = pd.read\_csv('C:\\Users\\Master\\Desktop\\snb edu\_codes\\5. SVM\\train.csv')

features = pd.DataFrame(data = rawData.loc[:, '0':'299'])

target = pd.DataFrame(data = rawData['target'])

target['target'].unique()

OUTPUT:

array([1., 0.])

target['target'].value\_counts()

OUTPUT:

0.0 183

1.0 67

Name: target, dtype: int64

#Split train and test sets (e.g., 80:20)

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, shuffle=True)

y\_train = y\_train.values.ravel()

y\_test = y\_test.values.ravel()

#Logistic Regression (Base Model)

from sklearn.linear\_model import LogisticRegression

logitModel = LogisticRegression(penalty='none').fit(x\_train, y\_train)

#Initial Evaluation

from sklearn.metrics import roc\_auc\_score

yhat = logitModel.predict(x\_test)

roc\_auc\_score(y\_test, yhat)

OUTPUT:

0.6078098471986417

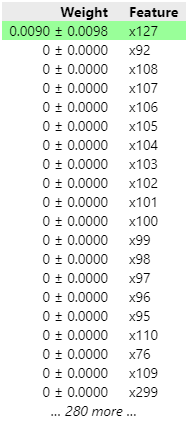
#Permutation Importance

import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(logitModel).fit(x\_train, y\_train)

eli5.show\_weights(perm)



#SHAP

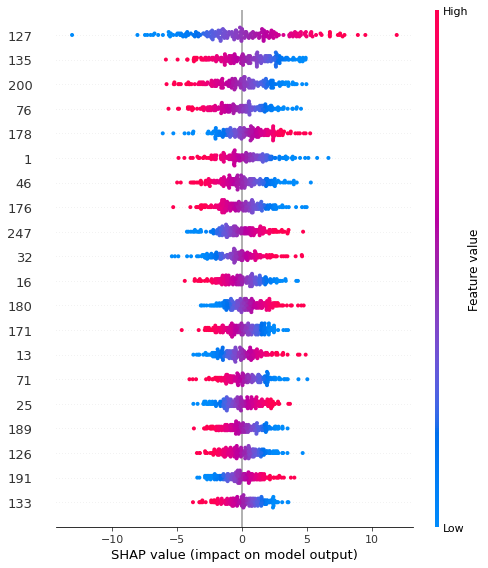
import shap

logitModel = LogisticRegression(penalty='none').fit(x\_train, y\_train)

explainer = shap.LinearExplainer(logitModel, x\_train)

shap\_value = explainer.shap\_values(x\_train)

shap.summary\_plot(shap\_value, x\_train)



As the evaluation results imply, most of the data have no significance at all.

#Feature Selection - RFE

from sklearn.feature\_selection import RFE, RFECV

feature\_selector = RFE(logitModel, n\_features\_to\_select=30).fit(x\_train, y\_train)

#feature\_selector.support\_

x\_train\_selected = x\_train.loc[:, feature\_selector.support\_]

x\_test\_selected = x\_test.loc[:, feature\_selector.support\_]

logitModel.fit(x\_train\_selected, y\_train)

yhat = logitModel.predict(x\_test\_selected)

roc\_auc\_score(y\_test, yhat)

#Evaluation

logitModel.fit(x\_train\_selected, y\_train)

yhat = logitModel.predict(x\_test\_selected)

roc\_auc\_score(y\_test, yhat)

OUTPUT:

0.6341256366723259

#Support Vector Machine – Base Model

from sklearn.svm import SVC

svmModel = SVC(C=10, gamma=10).fit(x\_train, y\_train)

yhat = svmModel.predict(x\_test)

roc\_auc\_score(y\_test, yhat)

yhat = svmModel.predict(x\_test)

roc\_auc\_score(y\_test, yhat)

OUTPUT:

0.5

#Cross Validation

from sklearn.model\_selection import cross\_val\_score, KFold, StratifiedKFold, RepeatedStratifiedKFold

base\_model = SVC(C=0.1, gamma=0.1)

scores = cross\_val\_score(base\_model, x\_train, y\_train, scoring='roc\_auc', cv=10)

display(scores)

display(scores.mean())

OUTPUT:

array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5])

0.5

folds = KFold(n\_splits=10, shuffle=True)

scores = cross\_val\_score(base\_model, x\_train, y\_train, scoring='roc\_auc', cv=folds)

scores.mean()

OUTPUT:

0.5

folds = StratifiedKFold(n\_splits=10, shuffle=True)

scores = cross\_val\_score(base\_model, x\_train, y\_train, scoring='roc\_auc', cv=folds)

scores.mean()

OUTPUT:

0.5

folds = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=5)

scores = cross\_val\_score(base\_model, x\_train, y\_train, cv=folds, scoring='roc\_auc')

scores.mean()

OUTPUT:

0.5

#Parameter Estimation

from sklearn.model\_selection import GridSearchCV

params = {

    'C':[0.001, 0.01,0.1,1,10,100,1000],

    'gamma':[0.001,0.01,0.1,1,10,100,1000]

}

base\_model = SVC()

results = GridSearchCV(base\_model, param\_grid=params, scoring='roc\_auc', cv=10).fit(x\_train, y\_train)

display(results.best\_params\_)

display(results.best\_score\_)

OUTPUT:

{'C': 0.1, 'gamma': 0.001}

0.6648124999999999

#Optmization

folds = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=123)

base\_model = SVC().fit(x\_train, y\_train)

scores = cross\_val\_score(base\_model, x\_train, y\_train, scoring='roc\_auc', cv=folds)

display(scores.mean())

OUTPUT:

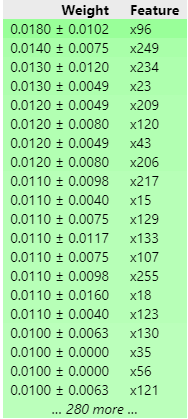
0.6631041666666666

perm = PermutationImportance(base\_model, random\_state=123).fit(x\_train, y\_train)

eli5.show\_weights(perm)

perm = PermutationImportance(base\_model, random\_state=123).fit(x\_train, y\_train)

eli5.show\_weights(perm)



eli5.formatters.as\_dataframe.explain\_weights\_df(perm)

selected\_columns = eli5.formatters.as\_dataframe.explain\_weights\_df(perm).loc[eli5.formatters.as\_dataframe.explain\_weights\_df(perm)['weight']!=0]

top\_features = [i[1:] for i in selected\_columns.feature if 'BIAS' not in i]

x\_train\_selected = x\_train[top\_features]

x\_test\_selected = x\_test[top\_features]

#feature\_selector = RFECV(base\_model, min\_features\_to\_select=30, cv=folds).fit(x\_train\_selected, y\_train)

#Does not work because RFECV only only works when coefficients are available (like linear models). This model is non-linear thus coefficients are not available

svmModel = SVC()

scores = cross\_val\_score(base\_model, x\_train\_selected, y\_train, scoring='roc\_auc', cv=folds)

scores.mean()

OUTPUT:

0.6415

params = {

    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],

    'gamma': [0.001, 0.01, 0.1, 1, 10 , 100, 1000]

}

results = GridSearchCV(svmModel, param\_grid=params, scoring='roc\_auc', cv=folds).fit(x\_train\_selected, y\_train)

results.best\_params\_

OUTPUT:

{'C': 0.1, 'gamma': 0.001}

results.best\_score\_

OUTPUT:

0.6443541666666667

results.best\_estimator\_

OUTPUT:

SVC(C=0.1, gamma=0.001)

best\_model = results.best\_estimator\_

yhat = best\_model.predict(x\_test\_selected)

roc\_auc\_score(y\_test, yhat)

OUTPUT:

0. 5713871328547415

Throughout the optimization process of SVM and comparison to the base model, it can be inferred that the relationship between the data have little or no non-linearity; which means that the data can be explained sufficiently with logistic regression. Let us proceed to pipeline optimization using logistic regression.

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

#base model

model = LogisticRegression(class\_weight='balanced', solver='liblinear')

#RFECV

feature\_selector = RFECV(model, min\_features\_to\_select=30, cv=folds, scoring='roc\_auc').fit(x\_train, y\_train)

x\_train\_selected = x\_train.loc[:, feature\_selector.support\_]

x\_test\_selected = x\_test.loc[:, feature\_selector.support\_]

#help(LogisticRegression)

pipeline = Pipeline([

    ('scale', StandardScaler()),

    ('logit', LogisticRegression())

])

params = {

    'logit\_\_penalty': ['l1','l2'],

    'logit\_\_C':[0.001,0.01,0.1,1,10,100,1000],

    'logit\_\_class\_weight':['balanced',None],

    'logit\_\_solver':['liblinear','lbfgs']

}

results = GridSearchCV(pipeline, param\_grid=params, scoring='roc\_auc', cv=folds, ).fit(x\_train\_selected, y\_train)

display(results.best\_params\_)

display(results.best\_score\_)

OUTPUT:

{'logit\_\_C': 10, 'logit\_\_class\_weight': 'balanced', 'logit\_\_penalty': 'l2', 'logit\_\_solver': 'liblinear'}

0.9973333333333333

pipeline.set\_params(logit\_\_C=10,logit\_\_class\_weight='balanced',logit\_\_penalty='l2',logit\_\_solver='liblinear')

OUTPUT:

Pipeline(steps=[('scale', StandardScaler()), ('logit', LogisticRegression(C=10, class\_weight='balanced', solver='liblinear'))]) pipeline.fit( x\_train\_selected, y\_train )

yhat = pipeline.predict( x\_test\_selected )

display( roc\_auc\_score( y\_test, yhat) )

OUTPUT:

0.75743123938879458