Model 3 Two class SVM

January 12, 2021

[1]:
```
#edit notebook width
 from IPython.display import display, HTML

 display(HTML(data="""
 <style>
     div#notebook-container    { width: 95%; }
     div#menubar-container     { width: 65%; }
     div#maintoolbar-container { width: 99%; }
 </style>
 """))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
#navigate to the right directory
import os
os.chdir("/data/momo/tooling")

#print directory to check the directory
os.getcwd()
```

[2]:
```
'/data/momo/tooling'
```

[3]:
```
#Importing the required libraries
from sklearn.metrics import accuracy_score, recall_score, precision_score
from mpl_toolkits.axes_grid1 import make_axes_locatable
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from _indexers import _generate_headers
from importing_csv import load_csv
from plotting import plot_features
from sklearn.svm import LinearSVC
from preprocessing import PEdata
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sn
import pandas as pd
import numpy as np
import statistics
import pickle
import glob
```

[4]:

```
# create variables that can be changed for testing
Training_composition = [50,50] #percentage of the in-bed / out of bed
test_size = 0.3 #size of the validation dataset
```

[5]:

```
# Read DF_Momo from pickle with all pe variantie values
DF_dataset = pd.read_pickle("/datc/momo/notebooks/Folder Salah/New_Features.
 ↪pkl")

# convert all columns with bedstatus 2 to 1 in DF_dataset
DF_dataset['Bed status'].replace({2: 1}, inplace=True)
```

[6]:

```
#Split the data into a training and test set by percentage
train, valid = train_test_split(DF_dataset, test_size=0.3)
```

[7]:

```
# Create a training dataset with only situation in which Bed status equals 0 in␣
 ↪the training set
DF_Out_of_bed = train[train["Bed status"] == 0 ]

# Create a training dataset with only situation in which Bed status equals 1 in␣
 ↪the training set
DF_In_bed = train[train["Bed status"] == 1 ]

# Count the number of times the Bed status equals 0 in the trainingset
Out_of_bed_count = len(DF_Out_of_bed['Bed status'] == 0)

# Count the number of times the Bed status equals 1 in the trainingset
In_bed_count = len(DF_In_bed['Bed status'] == 0)

# check which dataframe is longer and slice them accordingly to be the exact␣
 ↪same length
if In_bed_count > Out_of_bed_count:
    #get the total length of the dataframes and name the resutl:"Total_rows"
    Total_rows = int(len(DF_Out_of_bed))
    Total_rows1 = int((Total_rows / 100) * Training_composition[0])
    Total_rows2 = int((Total_rows / 100) * Training_composition[1])


elif Out_of_bed_count > In_bed_count :
    #get the total length of the dataframes and name the resutl:"Total_rows"
    Total_rows = int(len(DF_In_bed))
    Total_rows1 = int((Total_rows / 100) * Training_composition[1])
    Total_rows2 = int((Total_rows / 100) * Training_composition[0])
```

```

#Merge the in and out bed situations into a new dataframe
train = pd.concat([DF_In_bed[:Total_rows1], DF_Out_of_bed[:Total_rows2]])
```

[8]:

```
#This code creates the column names for the model
ModelColumns = []
columns = ['avg_column','FSR 15s variance','slope']
for item in columns:
    u = 10
    for i in range(12):
        ModelColumns += [item +' -'+str(u)+'s']
        u = u +10
```

[9]:

```
#Copy the features from the train set and store them in train_X
train_X = train[ModelColumns]



#Copy the features from the train set and store them in train_y
train_y = train['Bed status']

#Copy the features from the test set and store them in test_X
valid_X = valid[ModelColumns]

#Copy the features from the train set and store them in test_y
valid_y = valid['Bed status']
```

[10]:

```
#Convert the training sets to a numpy array and convert all values to int's
train_X = np.array(train_X).astype('int')
train_y = np.array(train_y).astype('int')

#Convert the test sets to a numpy array and convert all values to int's
valid_X = np.array(valid_X).astype('int')
valid_y = np.array(valid_y).astype('int')
```

[11]:

```
#Linear Support vector classifier model

#Initialize and fit the model on training data
model = LinearSVC(dual=False)
model.fit(train_X, train_y)
```

[11]:

```
LinearSVC(dual=False)
```

[12]:

```
#predict the bedstatus on the validation set valid_X
pred_y = model.predict(valid_X)

train_y_pred = model.predict(train_X)
```

[13]:

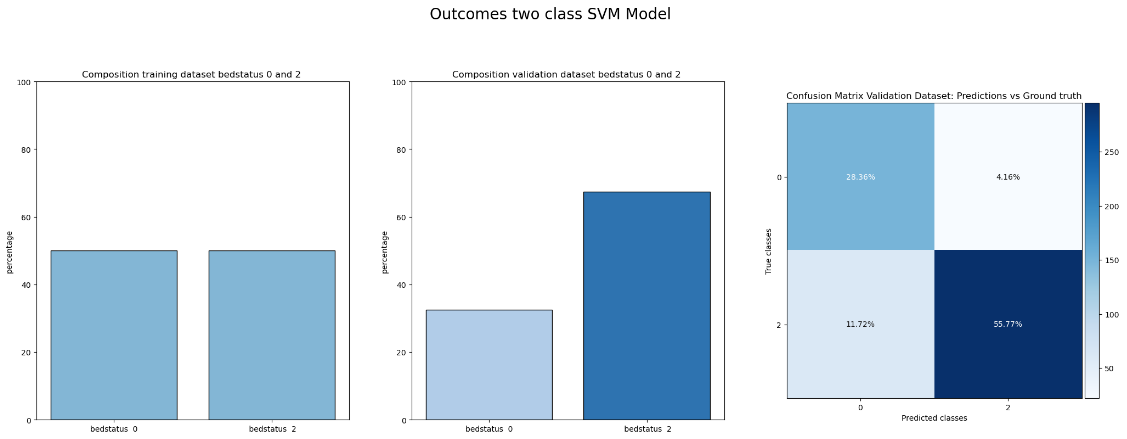
```
#Get the number of true negatives,false negatives, true positives and false␣
 ↪negatives for the validation set
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(valid_y, pred_y)
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
```

[14]:

```
#Group all Plots
fig, axs = plt.subplots(1, 3)

fig.set_figheight(8)
fig.set_figwidth(25)
fig.suptitle('Outcomes two class SVM Model', fontsize=20, y=1.05)

#Plot 1: show the composition of the Dataset used to train the model

#calculate the composition of the training set
train_0 = sum(train['Bed status'] == 0)
train_1 = sum(train['Bed status'] == 1)
train_total = len(train)
Percentage_train_0 = train_0/train_total * 100
Percentage_train_1 = train_1/train_total * 100

# compositions plot training data
All_percentage_train = [Percentage_train_0,Percentage_train_1]
bars_train = ('bedstatus  0', ' bedstatus  2')
y_pos = np.arange(len(bars_train))

if Percentage_train_0 <=10:
    color1= '#FFFFFF'
elif Percentage_train_0 > 10 and Percentage_train_0 <=20 :
    color1= '#F4F9FF'
elif Percentage_train_0 > 20 and Percentage_train_0 <=30 :
    color1= "#D1E2F3"
elif Percentage_train_0 > 30 and Percentage_train_0 <=40 :
    color1= "#B1CCE8"
elif Percentage_train_0 > 40 and Percentage_train_0 <=50 :
    color1= "#83B6D5"
elif Percentage_train_0 > 50 and Percentage_train_0 <=60 :
    color1= "#5094C5"
elif Percentage_train_0 > 60 and Percentage_train_0 <=70 :
    color1= "#2E73B0"
elif Percentage_train_0 > 70 and Percentage_train_0 <=80 :
    color1= "#5094C5"
elif Percentage_train_0 > 80 and Percentage_train_0 <=90 :
    color1= "#0D367F"
elif Percentage_train_0 > 90 and Percentage_train_0 <=100 :
    color1= "#0C1F4E"

if Percentage_train_1 <=10:
    color2='#FFFFFF'
elif Percentage_train_1 > 10 and Percentage_train_1 <=20 :
    color2='#F4F9FF'
elif Percentage_train_1 > 20 and Percentage_train_1 <=30 :
    color2='#D1E2F3'
elif Percentage_train_1 > 30 and Percentage_train_1 <=40 :
    color2='#B1CCE8'
elif Percentage_train_1 > 40 and Percentage_train_1 <=50 :
    color2='#83B6D5'
elif Percentage_train_1 > 50 and Percentage_train_1 <=60 :
    color2='#5094C5'
elif Percentage_train_1 > 60 and Percentage_train_1 <=70 :
    color2='#2E73B0'
elif Percentage_train_1 > 70 and Percentage_train_1 <=80 :
    color2='#5094C5'
elif Percentage_train_1 > 80 and Percentage_train_1 <=90 :
    color2='#0D367F'
elif Percentage_train_1 > 90 and Percentage_train_1 <=100 :
    color2='#0C1F4E'

colors= [color1,color2]






#Plot the training data with custom title
axs[0].bar(y_pos, All_percentage_train, color=colors,ec='black')
axs[0].set_title("Composition training dataset bedstatus 0 and 2 ")

# use the xticks function to add custom labels
axs[0].set_xticks(y_pos)
axs[0].set_xticklabels(bars_train)
axs[0].set_ylabel("percentage")
axs[0].set_ylim(0, 100)


#Plot 2: show the composition of the Dataset used to validate the model

#calculate the composition of the validation set
valid_0 = sum(valid['Bed status'] == 0)
```

```
valid_1 = sum(valid['Bed status'] == 1)
valid_total = len(valid)
Percentage_valid_0 = valid_0/valid_total * 100
Percentage_valid_1 = valid_1/valid_total * 100

# compositions plot validation data
All_percentage_valid = [Percentage_valid_0,Percentage_valid_1]
bars_valid = ('bedstatus  0', ' bedstatus  2')
y_pos = np.arange(len(bars_valid))


if Percentage_valid_0 <=10:
    color1= '#FFFFFF'
elif Percentage_valid_0 > 10 and Percentage_valid_0 <=20 :
    color1= '#F4F9FF'
elif Percentage_valid_0 > 20 and Percentage_valid_0 <=30 :
    color1= "#D1E2F3"
elif Percentage_valid_0 > 30 and Percentage_valid_0 <=40 :
    color1= "#B1CCE8"
elif Percentage_valid_0 > 40 and Percentage_valid_0 <=50 :
    color1= "#83B6D5"
elif Percentage_valid_0 > 50 and Percentage_valid_0 <=60 :
    color1= "#5094C5"
elif Percentage_valid_0 > 60 and Percentage_valid_0 <=70 :
    color1= "#2E73B0"
elif Percentage_valid_0 > 70 and Percentage_valid_0 <=80 :
    color1= "#5094C5"
elif Percentage_valid_0 > 80 and Percentage_valid_0 <=90 :
    color1= "#0D367F"
elif Percentage_valid_0 > 90 and Percentage_valid_0 <=100 :
    color1= "#0C1F4E"

if Percentage_valid_1 <=10:
    color2='#FFFFFF'
elif Percentage_valid_1 > 10 and Percentage_valid_1 <=20 :
    color2='#F4F9FF'
elif Percentage_valid_1 > 20 and Percentage_valid_1 <=30 :
    color2='#D1E2F3'
elif Percentage_valid_1 > 30 and Percentage_valid_1 <=40 :
    color2='#B1CCE8'
elif Percentage_valid_1 > 40 and Percentage_valid_1 <=50 :
    color2='#83B6D5'
elif Percentage_valid_1 > 50 and Percentage_valid_1 <=60 :
    color2='#5094C5'
elif Percentage_valid_1 > 60 and Percentage_valid_1 <=70 :
    color2='#2E73B0'
elif Percentage_valid_1 > 70 and Percentage_valid_1 <=80 :
    color2='#5094C5'
elif Percentage_valid_1 > 80 and Percentage_valid_1 <=90 :
    color2='#0D367F'
elif Percentage_valid_1 > 90 and Percentage_valid_1 <=100 :
    color2='#0C1F4E'

colors= [color1,color2]




# Plot the validation data with custom title
axs[1].bar(y_pos, All_percentage_valid, color=colors,ec='black')
axs[1].set_title("Composition validation dataset bedstatus 0 and 2 ")

# use the xticks function to add custom labels
axs[1].set_xticks(y_pos)
axs[1].set_xticklabels(bars_valid)
axs[1].set_ylabel("percentage")
axs[1].set_ylim(0, 100)






#Plot 3: show the composition of TP,FP,TN,FN in the model's predictions for the validation set
confusion = confusion_matrix(valid_y, pred_y)

CF1 = axs[2].imshow(confusion, cmap='Blues')

axs[2].set_title("Confusion Matrix Validation Dataset: Predictions vs Ground truth")
axs[2].set_xlabel('Predicted classes')
axs[2].set_ylabel('True classes')
axs[2].set_xticks(np.arange(0, 2,step=1))
axs[2].set_yticks(np.arange(0, 2,step=1))
axs[2].set_xticklabels(['0','2'])
axs[2].set_yticklabels(['0','2'])
axs[2].grid(False)
# Add colorbar
divider = make_axes_locatable(axs[2])
cax = divider.append_axes('right', size='5%', pad=0.05)
fig.colorbar(CF1, cax=cax, orientation='vertical')


confusion_percentage = np.array([[0,0],[0,0]],dtype=float)
confusion_percentage[0][0] = (confusion[0][0] / confusion.sum()) * 100
```

```
confusion_percentage[0][1] = (confusion[0][1] / confusion.sum()) * 100
confusion_percentage[1][0] = (confusion[1][0] / confusion.sum()) * 100
confusion_percentage[1][1] = (confusion[1][1] / confusion.sum()) * 100




# Loop over data dimensions and create text annotations.
max_value = np.max(confusion_percentage)
for i in range(2):
    for j in range(2):
        if confusion_percentage[i, j] > (max_value / 2):
            color="w"
        else:
            color="k"
        axs[2].text(j, i, "{:.2f}".format(confusion_percentage[i, j]) + "%",
                       ha="center", va="center", color=color)
```



[15]:
```
#Score the model based on the validation data and print the outcome
print("Model scores based on training data:")
print("")
print("accuracy score: " + str(accuracy_score(train_y, train_y_pred)))
print("recall score: " + str(recall_score(train_y, train_y_pred)))
```

```python
print("precision score: " + str(precision_score(train_y, train_y_pred)))
print("")
print("")
print("")


#Score the model based on the validation data and print the outcome
print("Model scores based on validation data:")
print("")
print("accuracy score: " + str(accuracy_score(valid_y, pred_y)))
print("recall score: " + str(recall_score(valid_y,␣
 ↪pred_y,pos_label=0,average='binary')))
print("precision score: " + str(precision_score(valid_y,␣
 ↪pred_y,pos_label=0,average='binary')))
```

```
Model scores based on training data:

accuracy score: 0.9043715846994536
recall score: 0.8797814207650273
precision score: 0.9252873563218391



Model scores based on validation data:

accuracy score: 0.8412098298676749
recall score: 0.872093023255814
precision score: 0.7075471698113207
```