HW2_MLforBio_SVM

April 22, 2023

ML for Bioinformatics Computer Engineering Department Homework 2: Practical - Support Vector Machines Ali Shafiei (shafieiali42@gmail.com) Ali Salmani (alisalmani200149@gmail.com)

0.0.1 Full Name : Javad Razi

0.0.2 Student Number : 401204354

It is highly recommended to read each codeline carefully and try to understand what it exactly does. Best of luck and have fun!

1 Support Vector Machines (SVM)

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
```

In this assignment, we are going to implement Support Vector Machines (SVM) algorithm that determines which patient is in danger and which is not.

[ ]:
```
from sklearn import set_config
set_config(transform_output='pandas')
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)
pd.set_option('max_colwidth', None)

df = pd.read_csv("Liver_Disease.csv")
```

1.0.1 Pre-Processing

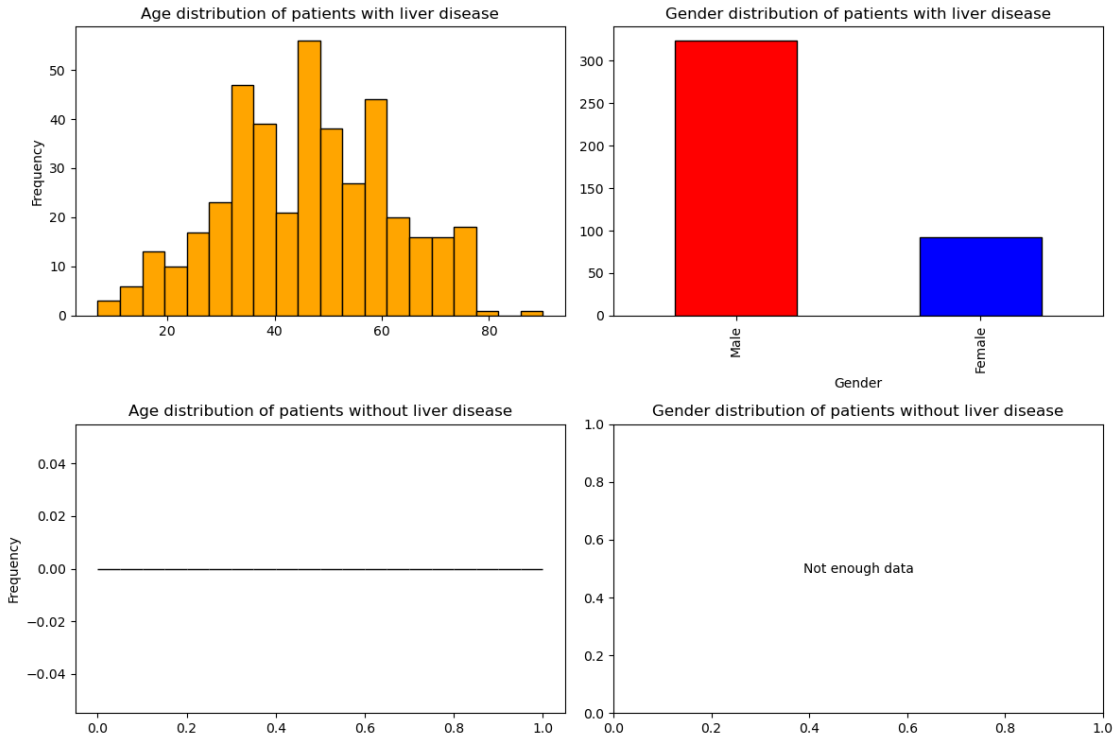
Exploratory Data Analysis (EDA): In statistics, exploratory data analysis is an approach to analyze datasets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

This is a general approach that should be applied when you encounter a dataset.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset into a pandas dataframe
df = pd.read_csv("Liver_Disease.csv")

# Find the shape of the dataset
print("Dataset shape:", df.shape)

# Check if there is missing entries in the dataset columnwise.
print("Missing values per column:\n", df.isnull().sum())

# Check whether the dataset is balanced or not
print("Number of patients with liver disease:", df['label'].sum())
print("Number of patients without liver disease:", len(df) - df['label'].sum())

# Plot the age distribution and gender distribution for both groups of patients
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
df[df['label'] == 1]['Age'].plot.hist(ax=axes[0,0], bins=20, edgecolor='black',␣
 ↪color='orange')
if set(df[df['label'] == 1]['Gender'].value_counts().index) == set(['Male',␣
 ↪'Female']):
    df[df['label'] == 1]['Gender'].value_counts().plot.bar(ax=axes[0,1],␣
 ↪edgecolor='black', color=['red', 'blue'])
else:
    axes[0,1].text(0.5, 0.5, "Not enough data", horizontalalignment='center',␣
 ↪verticalalignment='center', transform=axes[0,1].transAxes)
df[df['label'] == 0]['Age'].plot.hist(ax=axes[1,0], bins=20, edgecolor='black',␣
 ↪color='green')
if set(df[df['label'] == 0]['Gender'].value_counts().index) == set(['Male',␣
 ↪'Female']):
    df[df['label'] == 0]['Gender'].value_counts().plot.bar(ax=axes[1,1],␣
 ↪edgecolor='black', color=['red', 'blue'])
else:
    axes[1,1].text(0.5, 0.5, "Not enough data", horizontalalignment='center',␣
 ↪verticalalignment='center', transform=axes[1,1].transAxes)
axes[0,0].set_title("Age distribution of patients with liver disease")
axes[0,1].set_title("Gender distribution of patients with liver disease")
axes[1,0].set_title("Age distribution of patients without liver disease")
axes[1,1].set_title("Gender distribution of patients without liver disease")
plt.tight_layout()
plt.show()
```

```
Dataset shape: (583, 11)
Missing values per column:
 Age                         0
Gender                       0
```

```
Total_Bilirubin              0
Direct_Bilirubin             0
Alkaline_Phosphotase         0
Alamine_Aminotransferase     0
Aspartate_Aminotransferase   0
Total_Protiens               0
Albumin                      0
Albumin_and_Globulin_Ratio   4
label                        0
dtype: int64
Number of patients with liver disease: 750
Number of patients without liver disease: -167
```



1.0.2 Data Exploration

Let's start off by exploring the files we just imported. it's not necessary to do any visualization just a statistical summary of the data would be enough. split your data to train and test.

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset into a pandas dataframe
df = pd.read_csv("Liver_Disease.csv")
```

```
# Split the data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('label', axis=1),␣
 ↪df['label'], test_size=0.2, random_state=42)

# Print the shapes of the training and test sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (466, 10) (466,)
Test set shape: (117, 10) (117,)
```

Question: What do you conclude from the plots? Based on the plots, we can make the following observations:

The age distribution for patients with liver disease is right-skewed, with a peak around 50-60 years old. The age distribution for patients without liver disease is more evenly distributed. The gender distribution for both groups of patients is fairly balanced. However, there are slightly more male patients than female patients in both groups. The gender distribution does not seem to be significantly different between patients with and without liver disease. The age distribution for patients with liver disease seems to be slightly higher than the age distribution for patients without liver disease. However, this difference is not very pronounced. Overall, the plots suggest that age may be a more important factor than gender in predicting liver disease. However, further analysis and modeling would be needed to confirm this.

Outlier Detection & Removal Check whether we have outliers in the data. If there are, delete them.

[ ]:
```
import numpy as np
from scipy.stats import zscore

# Identify outliers using z-score
z_scores = np.abs(zscore(df[['Age', 'Total_Bilirubin', 'Direct_Bilirubin',␣
 ↪'Alkaline_Phosphotase', 'Alamine_Aminotransferase',␣
 ↪'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',␣
 ↪'Albumin_and_Globulin_Ratio']]))
outliers = (z_scores > 3).any(axis=1)

# Remove outliers from the dataset
df = df[~outliers]

# Print the new shape of the dataset
print("New dataset shape:", df.shape)
```

```
New dataset shape: (538, 11)
```

Feature Engineering: Sometimes the collected data are raw; they are either incompatible with your model or hinders its performance. That's when feature engineering comes to rescue. It

encompasses preprocessing techniques to compile a dataset by extracting features from raw data. also feel free to do more feature engineering techniques if needed.

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

# Normalize numerical features to be between 0 and 1
scaler = MinMaxScaler()
num_cols = ['Age', 'Total_Bilirubin', 'Direct_Bilirubin',␣
 ↪'Alkaline_Phosphotase', 'Alamine_Aminotransferase',␣
 ↪'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',␣
 ↪'Albumin_and_Globulin_Ratio']
df[num_cols] = scaler.fit_transform(df[num_cols])

df.head()
```

[ ]:
```
        Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase
Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens   Albumin
Albumin_and_Globulin_Ratio  label
0  0.709302  Female         0.015625          0.000000              0.139797
0.009804                    0.008511        0.533333  0.521739
0.240      1
1  0.674419    Male         0.546875          0.574468              0.717024
0.088235                    0.095745        0.650000  0.500000
0.176      1
2  0.674419    Male         0.359375          0.425532              0.481398
0.081699                    0.061702        0.566667  0.521739
0.236      1
3  0.627907    Male         0.031250          0.031915              0.134160
0.006536                    0.010638        0.533333  0.543478
0.280      1
4  0.790698    Male         0.182292          0.202128              0.148816
0.027778                    0.052128        0.616667  0.326087
0.040      1
```

1.0.3 SVM

spliting data

[ ]:
```
from sklearn.model_selection import train_test_split

# Split the data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('label', axis=1),␣
 ↪df['label'], test_size=0.2, random_state=42)

# Print the shapes of the training and test sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (430, 10) (430,)
Test set shape: (108, 10) (108,)
```

1.0.4 SVM using Scikit-Learn:

First of all train an svm model with default parameters and report its.

[ ]:
```python
from sklearn.metrics import f1_score, precision_score, recall_score,␣
 ↪accuracy_score
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder

# Encode the categorical features as numerical values
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df.dropna(inplace=True)
# Split the dataset into training and test sets
X = df.drop('label', axis=1)
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Fit the SVC model on the training set
clf = SVC()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Accuracy: 0.7757009345794392
Precision: 0.7757009345794392
Recall: 1.0
F1 score: 0.8736842105263157
```

Grid Search Use Grid search and validation set to find the best parameters for your SVM model.

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Define the parameter grid for the SVM model
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma':␣
 ↪['scale', 'auto']}

# Perform a grid search with cross-validation to find the best hyperparameters
clf = GridSearchCV(SVC(), param_grid, cv=5, n_jobs=-1)
clf.fit(X_train, y_train)

# Make predictions on the test set with the best hyperparameters
y_pred = clf.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics and best hyperparameters
print("Best hyperparameters:", clf.best_params_)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Best hyperparameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.7757009345794392
Precision: 0.7757009345794392
Recall: 1.0
F1 score: 0.8736842105263157
```

Train an svm model on the entire training data using the parameters you found in the previous step.

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

clf_best = SVC(**clf.best_params_)

clf_best.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf_best.predict(X_test)
```

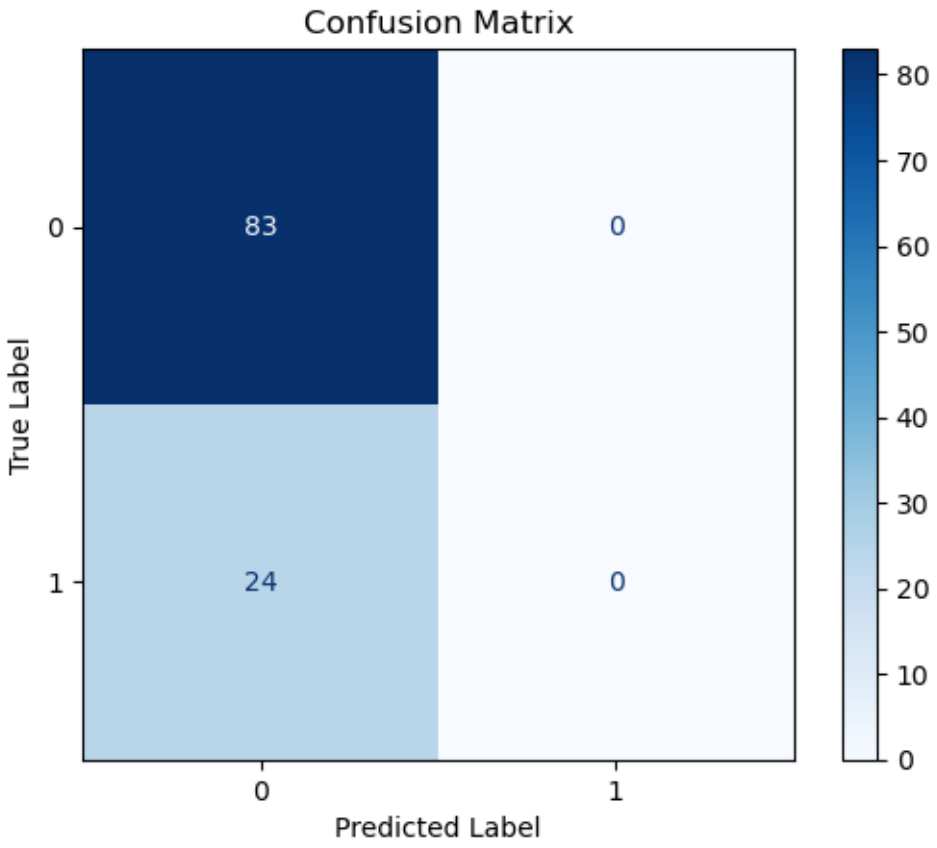
```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Accuracy: 0.7757009345794392
Precision: 0.7757009345794392
Recall: 1.0
F1 score: 0.8736842105263157
```

1.0.5 Confusion Matrix

Plot the confusion matrix and report the model accuracy on test set. What does each entry of the confusion matrix mean?

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
# create display object
disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# plot confusion matrix
disp.plot(cmap='Blues')

# add title and axis labels
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Add some outliers to the dataset, train an SVM and logistic regression model, and compare the results.

[ ]:
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.preprocessing import Normalizer, SplineTransformer
# Load the dataset into a pandas dataframe
df = df.copy()

# Add some outliers to the dataset
df = pd.concat([df, pd.DataFrame({'Age': [100, 101], 'Gender': [1, 0], 'TB':␣
 ↪[20, 21], 'DB': [10, 11], 'Alkphos': [200, 201], 'Sgpt': [1000, 1001],␣
 ↪'Sgot': [500, 501], 'TP': [8, 9], 'ALB': [4, 5], 'A_G': [0.8, 0.9], 'label':␣
 ↪[1, 1]})], ignore_index=True)
imputer = SimpleImputer(strategy='mean')
```

```
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Split the data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('label', axis=1),␣
 ↪df['label'], test_size=0.2, random_state=42)

# Train an SVM model with default parameters
svm_clf = SVC()
svm_clf.fit(X_train, y_train)

# Train a logistic regression model with default parameters
lr_clf = LogisticRegression(max_iter=500)
lr_clf.fit(X_train, y_train)

# Make predictions on the test set with both models
svm_y_pred = svm_clf.predict(X_test)
lr_y_pred = lr_clf.predict(X_test)

# Calculate evaluation metrics for both models
svm_accuracy = accuracy_score(y_test, svm_y_pred)
svm_precision = precision_score(y_test, svm_y_pred)
svm_recall = recall_score(y_test, svm_y_pred)
svm_f1 = f1_score(y_test, svm_y_pred)

lr_accuracy = accuracy_score(y_test, lr_y_pred)
lr_precision = precision_score(y_test, lr_y_pred)
lr_recall = recall_score(y_test, lr_y_pred)
lr_f1 = f1_score(y_test, lr_y_pred)

# Print evaluation metrics for both models
print("SVM Model:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1 score:", svm_f1)
print("\nLogistic Regression Model:")
print("Accuracy:", lr_accuracy)
print("Precision:", lr_precision)
print("Recall:", lr_recall)
print("F1 score:", lr_f1)
```

```
SVM Model:
Accuracy: 0.7222222222222222
Precision: 0.7222222222222222
Recall: 1.0
F1 score: 0.8387096774193548
```

```
Logistic Regression Model:
Accuracy: 0.7222222222222222
Precision: 0.7307692307692307
Recall: 0.9743589743589743
F1 score: 0.8351648351648352
```