Machine Learning

Assignment #4 Support Vector Machines

Submitted by:	Sec.	B.N.
Alaa Allah Essam Abdrabo	1	13

- # This assignment is composed of 2 Problems.
- # Part1 :
  - ## The Difference between the dataset using (without normalization) and those using(normalization):
    - 3 types of normalization were tried in this code

    1. Using built in function that scales values in the range [0, 1]:

```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()      # normalizatin with built in function
x_train_normalized = scale.fit_transform(x_train)
x_test_normalized = scale.transform(x_test)
```

    - That results in :

```
Average accuracy without normalization : 0.7546875
Average accuracy with normalization: 0.7553125
```

    2. scales values to have mean 0 and standard deviation 1:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()             # standadization
x_train_normalized = sc.fit_transform(x_train)
x_test_normalized = sc.transform(x_test)
```

    - That results in :

```
Average accuracy without normalization : 0.7521875
Average accuracy with normalization: 0.7540625000000001
```

3. normalization implementation from scratch based on min & max of train and normalize test data with same min & max of train :

```
#implementation of normalization

x_train_normalized= np.zeros((x_train.shape[0],x_train.shape[1]))
x_test_normalized = np.zeros((x_test.shape[0],x_test.shape[1]))
for j in range(x_train.shape[1]):
    train_feature=x_train.iloc[:,j]
    norm_train_feature = (train_feature - np.min( train_feature)) / (np.max( train_feature)-np.min( train_feature))
    x_train_normalized[:,j] = norm_train_feature
    test_feature=x_test.iloc[:,j]
    norm_test_feature=(test_feature - np.min( train_feature)) / (np.max( train_feature)-np.min( train_feature))
    x_test_normalized [:, j] = norm_test_feature
```

- That results in :

```
Average accuracy without normalization : 0.76125
Average accuracy with normalization: 0.7621875
```

The averaged accuracy over the ten trails:

3 types of normalization were tried in this code

1. Using built in function that scales values in the range [0, 1]:

```
Average accuracy without normalization : 0.7546875
Average accuracy with normalization: 0.7553125
```

2. scales values to have mean 0 and standard deviation 1:

```
Average accuracy without normalization : 0.7521875
Average accuracy with normalization: 0.7540625000000001
```

3. normalization implementation from scratch based on min & max of train and normalize test data with same min & max of train :

```
Average accuracy without normalization : 0.76125
Average accuracy with normalization: 0.7621875
```

The difference in the averaged accuracy of (normalized) and (not-normalized)

we notice that the average accuracy in case of using normalization is higher than without normalization

4. preprocessing steps to the data

No preprocessing was needed for the given data except for normalization as the data

- had no missing values
- no text as all data points are numeric

before normalization

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1
1	2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1
2	4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0
3	1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1
4	1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	1
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
795	4	9	2	23	2	2	2	4	2	22	3	1	1	1	1	0	0	1	0	1	0	0	0	1
796	1	18	2	75	5	5	3	4	2	51	3	1	2	2	1	0	1	1	0	0	0	0	0	1
797	4	12	4	13	1	2	2	4	2	22	3	2	1	1	1	0	0	1	0	1	0	0	1	0
798	4	24	3	7	5	5	4	4	3	54	3	2	1	2	1	1	0	1	0	0	1	0	0	1
799	2	9	2	15	5	2	3	2	1	35	3	1	1	1	1	1	0	1	0	0	1	1	0	0

after normaliztion the range of each feature is between [0,1]

```
x_train_normalized: array([[1.        , 0.19642857, 0.75      , ..., 0.        , 0.        ,
  > special variables
  > [0:480] : [array([1.        , 0...        ]), array([0.        , 0...        ]), array([0.33333333, 0...
  > dtype: dtype('float64')
    max: 1.0
    min: 0.0
```

- # Part2 :

- ## Implement, from scratch, linear SVM model using Gradient descent as an optimization function
- ## Required functions

1.parameters used in this algorithm

- C : is the hyperparameter "Regularization Constant" that determines to what extent the soft margine would be(15.0)
- B : is the beta in hyperplane equation [h(x)=B1X1+B2X2+....+b] has number of values according to number of features
- b : is the bias in previous equation
- Learning rate: 0.001
- number of iterations: 500

```python
def __init__(self, C=1.0):

    self.C = C
    self.B = None
    self.b = None
```

2.Fit function

```python
def fit(self, X, y, LR=0.001, iterations=500):

    # Initialize  hyperplane equation parameters B(beta) and b(bias)
    space_dimension=X.shape[1]
    self.B = np.random.randn(space_dimension)      # B is the beta in hyperplane equation [h(x)=B1*X1+B2*X2+....+b] has number of values according to number of features
    self.b = 0                                     # b is the bias in previous equation

    #cost_arr = []

    for i in range(iterations):
        decision=X.dot(self.B) + self.b
        self.sign=np.sign(decision)
        margin = y * decision                # margine equation y(B1*X1+B2*X2+....+B)=1&-1 according to the label 1 &-1  ==== or zero in data point lies on that plane

       # Gradient descent
        #cost= 0.5* self.B.dot(self.B) + self.C * np.sum(np.maximum(0, 1 - margin))     # cost function
        #cost_arr.append(cost)
        #print(cost)

        wrong_class = np.where(margin < 1)[0]
        d_B = self.B - self.C * y[wrong_class].dot(X[wrong_class])         # derivative of cost function to beta
        self.B = self.B - LR * d_B                                         # updated beta
        d_b = - self.C * np.sum(y[wrong_class])                            # derivative of cost function to bias
        self.b = self.b - LR * d_b                                         # updated bias
    self.support_vectors_train = np.where(margin <= 1)[0]                  # used in plotting
```

3. predict function

```python
def predict(self,X,y):

    prediction= np.sign(self.hyperplane(X))
    margin_test = y * self.hyperplane(X)      # for plot
    self.support_vectors_test = np.where(margin_test <= 1)[0]   # for plot
    return np.mean(y ==  prediction)
```

- ## Loading Data

- data was loaded from seaborn package

- For binary classification only two classes of data were used so, the last 100 points were extracted being of 2 classes

- In this algorithm equations were designed according to classes with codes of -1 & 1 then all 0 class were just encoded to be -1 and the other class was already of code 1

-"species" column was dropped as it had classes written in text

- This data has 4 features but only 2 were taken as mentioned in the statement to be easily drawn in 2d graph

- preprocessing step was applied to the data which is standardization

- Splitting data into training and testing data with 60% training

```
#=======================================Loading and preparing the data for binary classification in 2d ====================

data= sns.load_dataset("iris")   # columns : sepal_length,sepal_width,petal_length,petal_width,species
#print(data.shape)  # (150,5)
data= data.tail(100)            # to use the data in binary classification: use only last 100 points as they contain only two classes
#print(data.shape)  #(100,5)
encode = preprocessing.LabelEncoder()
labels= encode.fit_transform(data["species"])
labels[labels == 0] = -1        # replacing 0 labels with -1 to work with the equations that is based on that
#print(labels.shape)            # (100,) only 1&0
data= data.drop(["species"], axis=1)
data=data.iloc[:,2:4]
data=np.asarray(data)
# preprocessing: Standardize the data.
scale= StandardScaler()
data= scale.fit_transform(data)

x_train,x_test,y_train,y_test=train_test_split(data,labels,test_size=0.4,random_state=42)
```

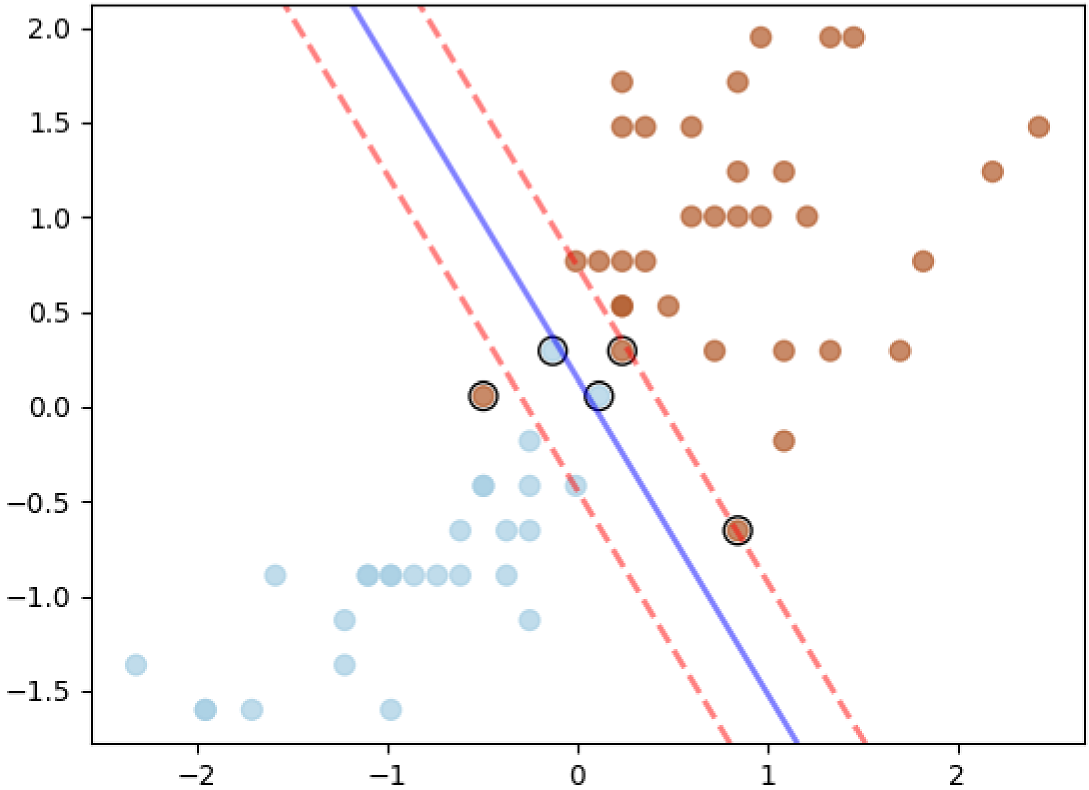
Accuracy

Accuracy from implemented algorithm & sklearn

```
sklearn accuracy using svm 0.875
accuaracy of implemented algorithm: 0.875
```

Plotting the 2 features

1. using train data



1. using train data