

Introduction to Machine Learning

*Thesis submitted in partial fulfillment of the
requirements for the degree*

of

Master of Technology(D.E)

by

**Arun Kumar
MIT2020116**

Under the supervision of

Dr.K.P Mishra



**DEPARTMENT OF INFORMATION TECHNOLOGY (DATA ENGINEERING)
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD**

© *Arun Kumar*
All rights reserved

DECLARATION

Introduction of Machine Learning

Author

Arun Kumar

Student ID

MIT2020116

Supervisor

Dr.K.P Mishra

I declare that this thesis entitled is Introduction of Machine Learning the result of my own work except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Arun Kumar

MIT2020116

Department of Information Technology(D.E)
Indian Institute of Information Technology,
Allahabad

Date: March 27,2021

ASSIGNMENT-2: Introduction to Machine Learning

Q1. Create an ANN with one hidden layer and do classification on the datasets given in the link.

([https://archive.ics.uci.edu/ml/datasets/Statlog+\(Landsat+Satellite\)](https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)))

(<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+Diagnostic>)

(<https://github.com/EpistasisLab/pmlb/tree/master/datasets/iris>)

(<https://github.com/EpistasisLab/pmlb/tree/master/datasets/titanic>)

i) Plot a graph of accuracy vs. the number of hidden units.(64,128,256,512)

ii) Plot a graph of accuracy vs. activation function.(Relu,logistic sigmoid,tanh,leaky Relu)

iii) Plot a graph comparing the following three loss functions vs accuracy

a)Multi-Class Cross-Entropy Loss

b)Sparse Multiclass Cross-Entropy Loss

c)Kullback Leibler Divergence Loss

A-NN:

```
1
2 #!/usr/bin/env python
3 # coding: utf-8
4
5 # In[1]:
6
7
8 #Q-1. Create an ANN with one hidden layer and do
   classification on
9   the datasets given in the link:
10
11
12 # In[2]:
13
14
15 import numpy as np
16 import pandas as pd
17 import seaborn as sns
18 import matplotlib.pyplot as plt
19 from sklearn.model_selection import train_test_split
20 from sklearn.metrics import confusion_matrix
21 from sklearn.preprocessing import LabelEncoder
22 from sklearn.preprocessing import StandardScaler
23
```

```

24
25
26 #Check observation one by one different dataset and see
    observations.
27
28 data = pd.read_csv("https://raw.githubusercontent.com/psantul/
    Dataset/main/data.csv", encoding='latin-1')
29 #data = pd.read_csv("https://raw.githubusercontent.com/psantul
    /Dataset/main/titanic.csv", delimiter = ',', encoding='latin
    -1')
30 print(data.shape)
31 data.head()
32
33
34 # In[3]:
35
36
37 x = data.iloc[:,3:].values
38 y = data.iloc[:,1].values
39
40
41 # In[4]:
42
43
44 encoder = LabelEncoder()
45 y = encoder.fit_transform(y)
46
47
48 # In[5]:
49
50
51 X_train, X_test, y_train, y_test = train_test_split(x, y,
    test_size = 0.2, random_state = 0)
52
53
54 # In[6]:
55
56
57 sc = StandardScaler()
58 X_train = sc.fit_transform(X_train)
59 X_test = sc.fit_transform(X_test)
60
61
62 # In[7]:
63

```

```

64
65 class NeuralNetwork:
66
67     def __init__(self,X, y, X_test, y_test, hidden_nodes=1,
68         learning_rate=0.1, epochs=5000):
69
70         #data
71         self.y = y[:,None]
72         self.X = X
73
74         self.X_test = X_test
75         self.y_test = y_test
76
77         #parameters
78         np.random.seed(4)
79         self.input_nodes = len(X[0])
80         self.hidden_nodes = hidden_nodes
81         self.output_nodes = self.y.shape[1]
82         self.learning_rate = learning_rate
83
84         #init weights
85         self.w1 = 2*np.random.random((self.input_nodes, self.
86             hidden_nodes)) - 1
87         self.w2 = 2*np.random.random((self.hidden_nodes, self.
88             output_nodes)) - 1
89
90         self.train(epochs)
91         self.test()
92
93     def sigmoid(self,X):
94         return (1/(1+np.exp(-X)))
95
96     def sigmoid_prime(self,X):
97         return X * (1 - X)
98
99     def train(self, epochs):
100
101         for e in range(epochs):
102
103             #FORWARD PROPAGATION
104
105             #hidden layer
106             # W1(398,30) X(30,12)
107             l1 = self.sigmoid(np.dot(self.X, self.w1))

```

```

106
107         #output layer
108         #l1(398,12) W2(12,1)
109         l2 = self.sigmoid(np.dot(l1, self.w2))
110
111         # BACKPROPAGATION
112
113         #calculate how far off our prediciton was
114         error = self.y-l2
115
116         #calculate how far off each layer is
117         l2_delta = error * self.sigmoid_prime(l2)
118         l1_delta = l2_delta.dot(self.w2.T) * self.
            sigmoid_prime(l1)
119
120         #update weights with our newly found error values
121         self.w2 = np.add(self.w2, l1.T.dot(l2_delta) *
            self.learning_rate)
122         self.w1 = np.add(self.w1, self.X.T.dot(l1_delta) *
            self.learning_rate)
123
124         #print('Error:', (abs(error)).mean())
125
126     def test(self):
127         correct = 0
128         pred_list = []
129
130         #replicate feedforward network for testing
131         l1 = self.sigmoid(np.dot(self.X_test, self.w1))
132         l2 = self.sigmoid(np.dot(l1, self.w2))
133
134         #loop through all of the outputs of layer 2
135         for i in range(len(l2)):
136             if l2[i] >= 0.5:
137                 pred = 1
138             else:
139                 pred = 0
140
141             if pred == self.y_test[i]:
142                 correct += 1
143
144             pred_list.append(pred)
145         print(pred_list)
146         print(y_test)
147         correct = np.sum(pred_list == y_test)

```

```
148         print("%d_out_of_%d_predictions_correct" % (correct,
149               len(pred_list)))
150         print("Test_Accuracy:", ((correct/len(y_test))*100),
151               '%')
152
153 # In[8]:
154
155
156 nn = NeuralNetwork(X_train, y_train, X_test, y_test)
```


- i) Plot a graph of accuracy vs. the number of hidden units.(64,128,256,512)
- ii) Plot a graph of accuracy vs. activation function.(Relu,logistic sigmoid,tanh,leaky Relu)
- iii) Plot a graph comparing the following three loss functions vs accuracy

```

1
2  #!/usr/bin/env python
3  # coding: utf-8
4
5
6
7  #i) Plot a graph of accuracy vs. the number of hidden
8  #units.(64,128,256,512)
9  #ii) Plot a graph of accuracy vs. activation function.
10  #(Relu,logistic sigmoid,tanh,leaky Relu)
11  #iii) Plot a graph comparing the following three
12  #loss functions vs accuracy
13
14
15  import numpy as np
16  import pandas as pd
17  import seaborn as sns
18  import matplotlib.pyplot as plt
19  from sklearn.model_selection import train_test_split
20  from sklearn.metrics import confusion_matrix
21  from sklearn.preprocessing import LabelEncoder
22  from sklearn.preprocessing import StandardScaler
23
24
25  Check observation one by one different dataset remove
26  comment part and see observations
27
28
29  data = pd.read_csv("https://raw.githubusercontent.com
30  _/psantul/Dataset/main/data.csv", encoding='latin-1')
31  #data = pd.read_csv("https://raw.githubusercontent.com
32  _/psantul/Dataset/main/titanic.csv",delimiter=';',
33  encoding='latin-1')
34  print(data.shape)
35  data.head()
36
37  x=data.iloc[:,3:].values
38  y=data.iloc[:,1].values

```

```

39
40 encoder = LabelEncoder()
41 y = encoder.fit_transform(y)
42
43 X_train, X_test, y_train, y_test = train_test_split(x, y,
44 test_size = 0.2, random_state = 0)
45
46 print(X_train.shape)
47 print(y_train.shape)
48 print(X_test.shape)
49 print(y_test.shape)
50
51 np.unique(y_train)
52
53 nn_input_dim = X_train.shape[1]
54 nn_output_dim = len(np.unique(y_train))
55 lr = 0.01
56
57 # helper function to calculate total loss on the dataset
58 def calculate_loss(model, X, y):
59     num_examples = X.shape[0]
60     W1 = model['W1']
61     b1 = model['b1']
62     W2 = model['W2']
63     b2 = model['b2']
64     # forward propagation to calculate out predictions
65     z1 = X.dot(W1) + b1
66     a1 = np.tanh(z1)
67     z2 = a1.dot(W2) + b2
68     exp_scores = np.exp(z2)
69     probs = exp_scores / np.sum(exp_scores, axis=1,
70 keepdims=True)
71     # Calculating the loss
72     correct_logprobs = -np.log(probs[range(num_examples), y])
73     data_loss = np.sum(correct_logprobs)
74     return 1. / num_examples * data_loss
75
76 def predict(model, x):
77     W1 = model['W1']
78     b1 = model['b1']
79     W2 = model['W2']
80     b2 = model['b2']
81
82
83     # Forward propagation

```

```

84     zz1=xx.dot(W1)+b1
85     aa1=np.tanh(z1)
86     zz2=aa1.dot(W2)+b2
87     exp_scores=np.exp(z2)
88     probs=exp_scores/np.sum(exp_scores,axis=1,
89     keepdims=True)
90     return np.argmax(probs,axis=1)
91
92     #(Relu,logistic,sigmoid,tanh,leaky_Relu)
93
94     def relu(x):
95         xx[x<0]=0
96         return x
97
98     def logsig(x):
99         return 1/(1+np.exp(-x))
100
101     def tanh(x):
102         return np.tanh(x)
103
104     def leaky_relu(x):
105         xx[x<0]*=0.01
106         return x
107
108     def build_model(X,y,nn_hdim,num_passes=10000,
109     print_loss=False):
110
111         num_examples=X.shape[0]
112         np.random.seed(0)
113         W1=np.random.rand(nn_input_dim,nn_hdim)/
114         np.sqrt(nn_input_dim)
115         b1=np.zeros((1,nn_hdim))
116         W2=np.random.rand(nn_hdim,nn_output_dim)/
117         np.sqrt(nn_hdim)
118         b2=np.zeros((1,nn_output_dim))
119         model={}
120         losses=[]
121
122         #Gradient_descent.for_each_batch
123         for i in range(0,num_passes):
124
125
126             #Forward_propagation
127             z1=X.dot(W1)+b1
128             a1=tanh(z1)

```

```

129 z2=a1.dot(W2)+b2
130 exp_scores=np.exp(z2)
131 probs=exp_scores/np.sum(np.array(exp_scores),
132 axis=1,keepdims=True)
133
134 #Backpropagation
135 delta3=np.array(probs)
136 delta3[range(num_examples),y]-=1
137 dW2=(a1.T).dot(delta3)
138 db2=np.sum(delta3,axis=0,keepdims=True)
139 delta2=delta3.dot(W2.T)*(1-np.power(a1,2))
140 dW1=np.dot(X.T,delta2)
141 db1=np.sum(delta2,axis=0)
142
143 #Gradient descent parameter update
144 W1+=-lr*dW1
145 b1+=-lr*db1
146 W2+=-lr*dW2
147 b2+=-lr*db2
148
149
150 #Assign new parameters to the model
151 model={'W1':W1,'b1':b1,'W2':W2,'b2':b2}
152
153 if print_loss and i%1000==0:
154     loss=calculate_loss(model,X,y)
155     losses.append(loss)
156     print("Loss after iteration %i: %f"%(i,loss))
157
158 return model,losses
159
160 x=[]
161 y=[]
162
163 for nodes in [64,128,256,512]:
164     print('\nfor {} number of nodes'.format(nodes))
165     model,loss=build_model((X_train),(y_train),nodes,
166     print_loss=True)
167     preds=predict(model,X_test)
168     acc=((preds==y_test).sum()/len(y_test))*100
169     x.append(nodes)
170     y.append(acc)
171
172 plt.xlabel('number of nodes')
173 plt.ylabel('accuracy')

```

```
174 plt.title('using {} activation function'.format('leaky_relu'))
175 #place of relu changes logsig, tanh, leaky_relu:
176 Different loss result shown in graph and accuracy
177
178 plt.plot(x, y)
```

Output: (64,128,256,512)

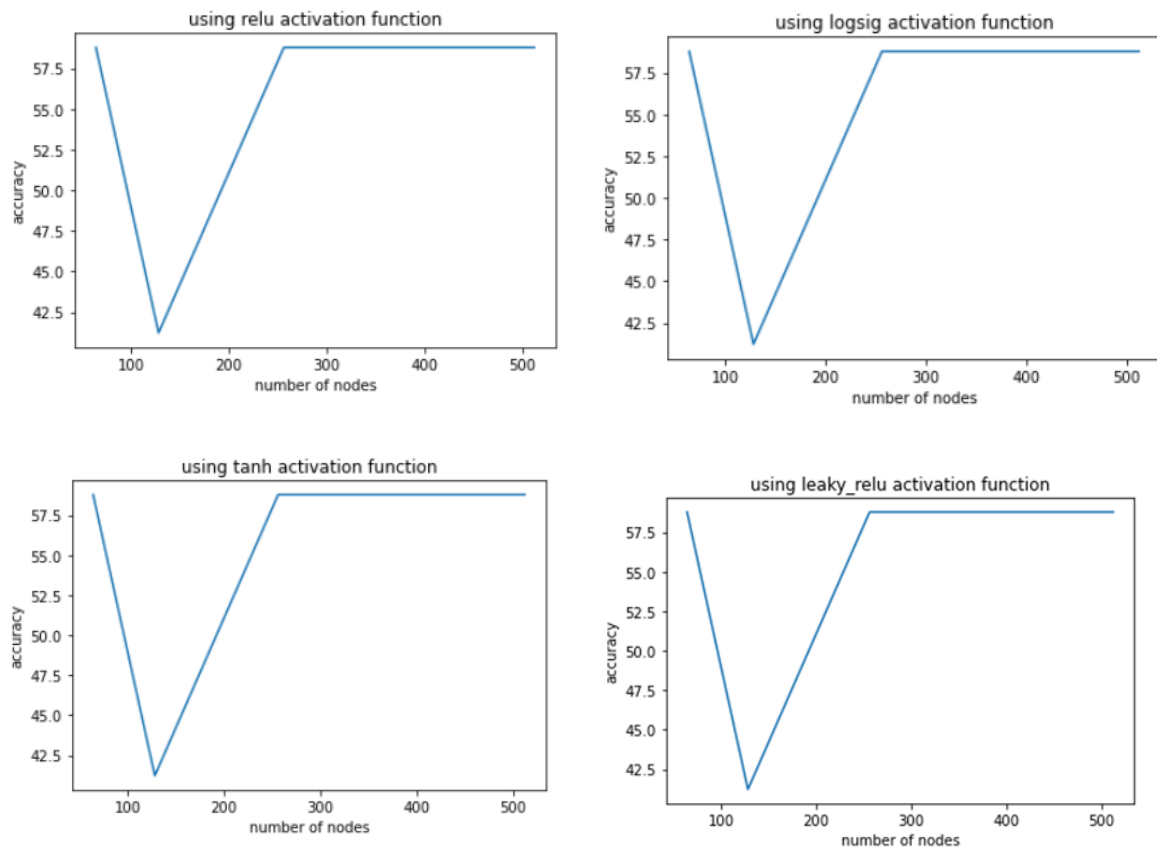
[illegible]

Fig: Observation of Dateset-1: <https://raw.githubusercontent.com/psantul/Dataset/main/data.csv>

Output: (64,128,256,512)

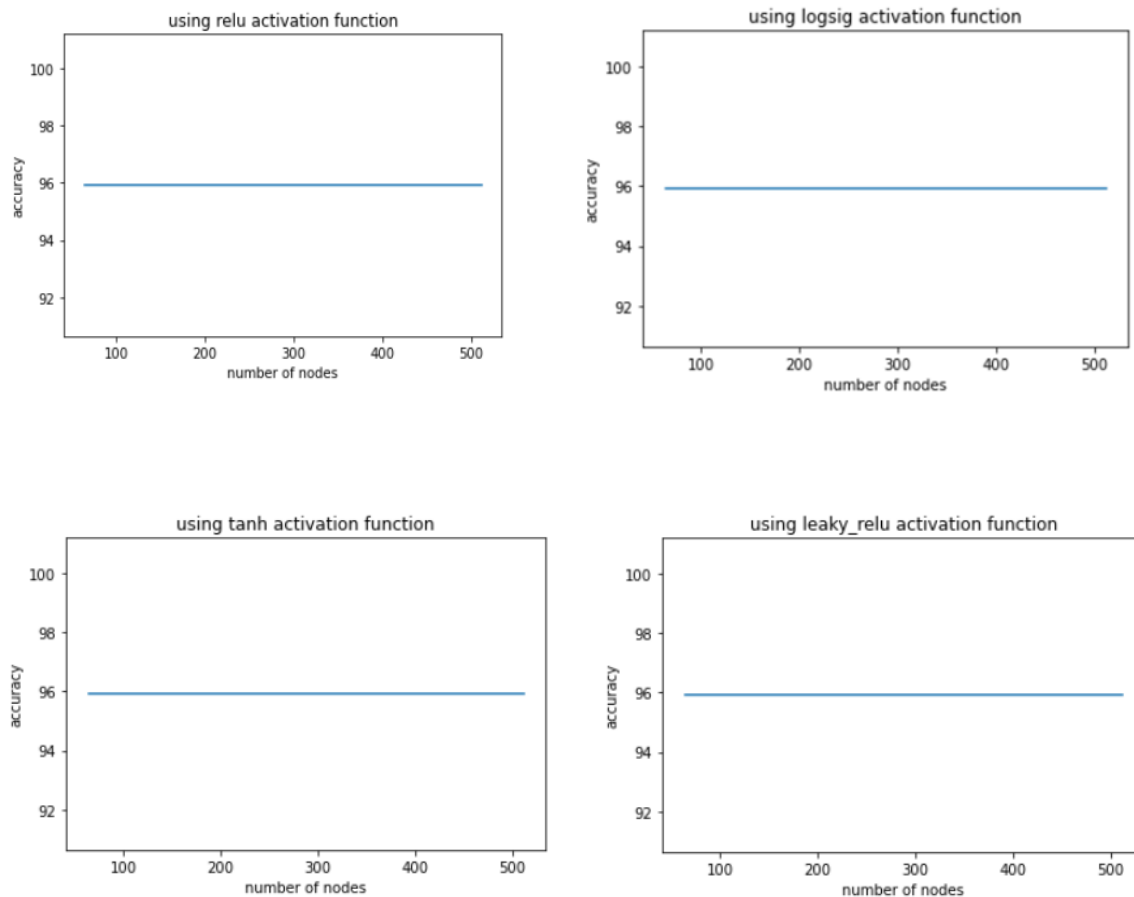
[illegible]

Fig: Observation of dataset 2:<https://raw.githubusercontent.com/psantul/Dataset/main/titanic.csv>

- iii) Plot a graph comparing the following three loss functions vs accuracy
- a) Multi-Class Cross-Entropy Loss
 - b) Sparse Multiclass Cross-Entropy Loss
 - c) Kullback Leibler Divergence Loss

```
1
2 # mlp for regression with mse loss function
3 from sklearn.datasets import make_regression
4 from sklearn.preprocessing import StandardScaler
5 from keras.models import Sequential
6 from keras.layers import Dense
7 from keras.optimizers import SGD
8 from matplotlib import pyplot
9 from sklearn.preprocessing import LabelEncoder
10 import pandas as pd
11
12 data = pd.read_csv("https://raw.githubusercontent.com/psantul/
13 Dataset/main/titanic.csv", delimiter = ',', encoding='latin-1')
14 print(data.shape)
15 data.head()
16
17 X = data.iloc[:,3:].values
18 y = data.iloc[:,1].values
19
20 encoder = LabelEncoder()
21 y = encoder.fit_transform(y)
22
23 # split into train and test
24 from sklearn.model_selection import train_test_split
25 X_train, X_test, y_train, y_test = train_test_split(X, y,
26 test_size = 0.2, random_state = 0)
27
28 import tensorflow as tf
29 from tensorflow.keras.models import Sequential
30
31 # define model
32 model = Sequential()
33 model.add(Dense(25, input_dim=1, activation='relu',
34 kernel_initializer='he_uniform'))
35 model.add(Dense(1, activation='linear'))
36 opt = SGD(lr=0.01, momentum=0.9)
37 model.compile(loss='mean_squared_error', optimizer=opt)
38
```



```

39 X_test.shape, y_test.shape
40
41 X.shape, y.shape
42
43 # fit model
44 history = model.fit(X_train, y_train,
45 validation_data=(X_test, y_test), epochs=100, verbose=0)
46
47 # evaluate the model
48 train_mse = model.evaluate(X_train, y_train, verbose=0)
49 test_mse = model.evaluate(X_test, y_test, verbose=0)
50
51 print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
52 # plot loss during training
53 pyplot.title('Loss / Mean Squared Error')
54 pyplot.plot(history.history['loss'], label='train')
55 pyplot.plot(history.history['val_loss'], label='test')
56 pyplot.legend()
57 pyplot.show()
58
59 """#Kullback Leibler Divergence Loss"""
60
61 from keras.utils import to_categorical
62 # one hot encode output variable
63 y = to_categorical(y)
64
65 # define model
66 model = Sequential()
67 model.add(Dense(50, input_dim=1, activation='relu',
68 kernel_initializer='he_uniform'))
69 model.add(Dense(3, activation='softmax'))
70
71 # compile model
72 opt = SGD(lr=0.01, momentum=0.9)
73 model.compile(loss='kullback_leibler_divergence',
74 optimizer=opt, metrics=['accuracy'])
75
76 # fit model
77 history = model.fit(X_train, y_train,
78 validation_data= (X_test, y_test), epochs=100, verbose=0)
79
80 # evaluate the model
81 _, train_acc = model.evaluate(X_train, y_train, verbose=0)
82 _, test_acc = model.evaluate(X_test, y_test, verbose=0)
83 print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

```

```

84
85 # plot loss during training
86 pyplot.figure(figsize=(10,8))
87 pyplot.subplot(211)
88 pyplot.title('Loss')
89 pyplot.plot(history.history['loss'], label='train')
90 pyplot.plot(history.history['val_loss'], label='test')
91 pyplot.legend()
92
93 # plot accuracy during training
94 pyplot.subplot(212)
95 pyplot.title('Accuracy')
96 pyplot.plot(history.history['accuracy'], label='train')
97 pyplot.plot(history.history['val_accuracy'], label='test')
98 pyplot.legend()
99 pyplot.show()
100
101 """#Sparse Multiclass Cross-Entropy Loss"""
102
103 # define model
104 model = Sequential()
105 model.add(Dense(50, input_dim=1,activation='relu',
106 kernel_initializer='he_uniform'))
107 model.add(Dense(3, activation='softmax'))
108 # compile model
109 opt = SGD(lr=0.01, momentum=0.9)
110 model.compile(loss='sparse_categorical_crossentropy',
111 optimizer=opt, metrics=['accuracy'])
112 # fit model
113 history = model.fit(X_train, y_train,
114 validation_data=(X_test, y_test), epochs=100, verbose=0)
115 # evaluate the model
116 _, train_acc = model.evaluate(X_train, y_train, verbose=0)
117 _, test_acc = model.evaluate(X_test, y_test, verbose=0)
118 print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
119 # plot loss during training
120 pyplot.subplot(211)
121 pyplot.title('Loss')
122 pyplot.plot(history.history['loss'], label='train')
123 pyplot.plot(history.history['val_loss'], label='test')
124 pyplot.legend()
125 # plot accuracy during training
126 pyplot.subplot(212)
127 pyplot.title('Accuracy')
128 pyplot.plot(history.history['accuracy'], label='train')

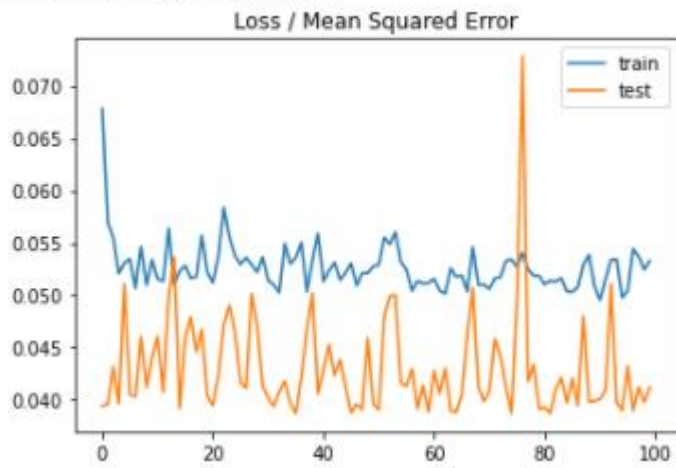
```

```
129 pyplot.plot(history.history['val_accuracy'], label='test')
130 pyplot.legend()
131 pyplot.show()
```

Output:

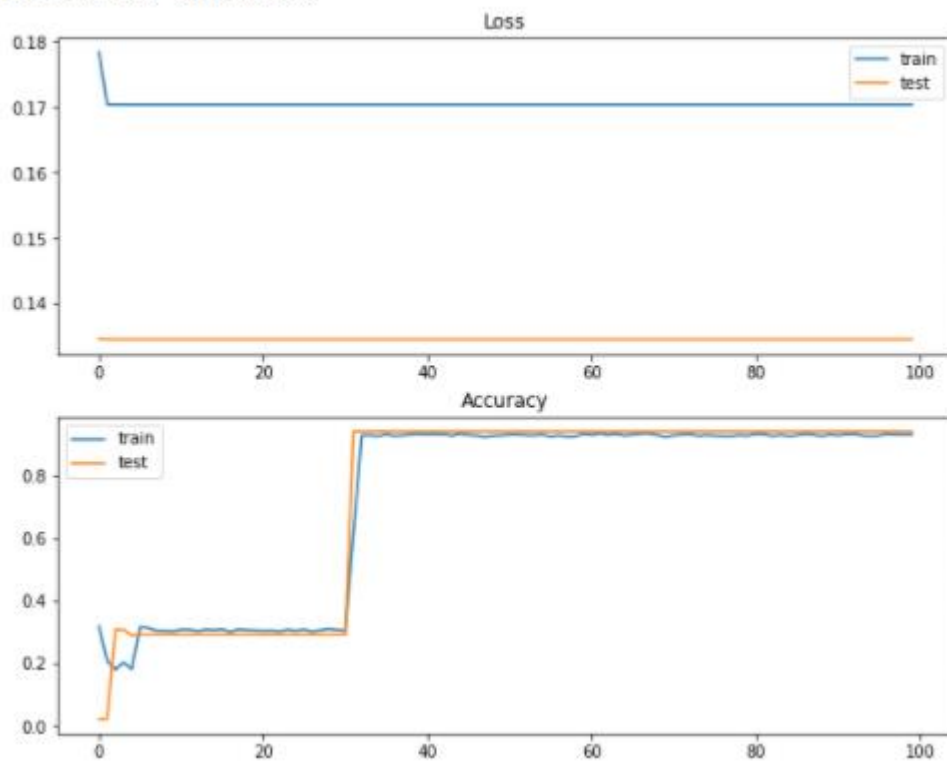
a) Multi-Class Cross-Entropy Loss:

☞ Train: 0.051, Test: 0.041



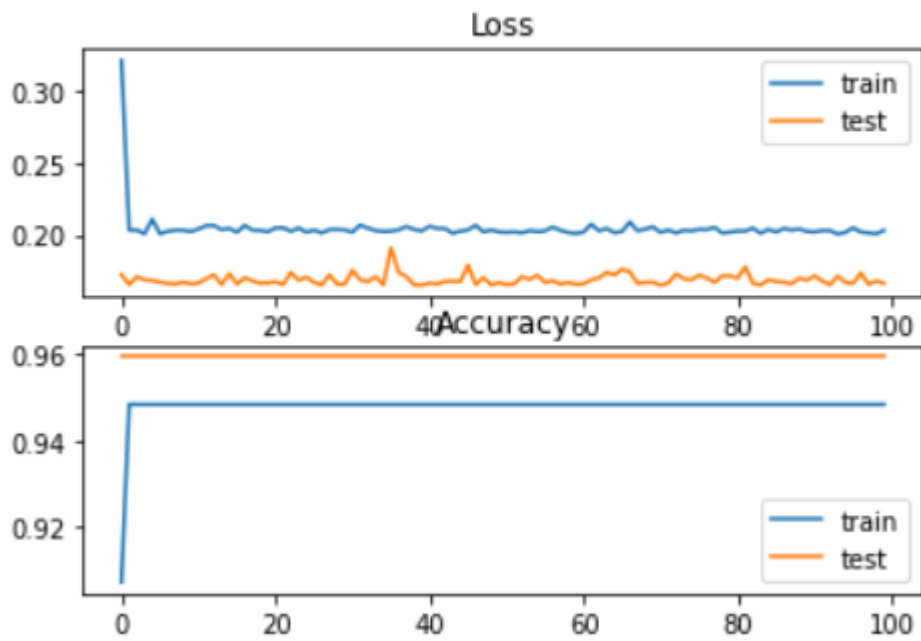
b) Sparse Multiclass Cross-Entropy Loss:

☞ Train: 0.935, Test: 0.943



c) Kullback Leibler Divergence Loss:

Train: 0.948, Test: 0.959



Q2.Implement SVM from scratch on the above datasets and plot a graph for the given kernel functions:

- i)Linear kernel vs accuracy
- ii)Polynomial kernel vs accuracy
- iii)Gaussian RBF kernel vs accuracy

```
1
2  #!/usr/bin/env python
3  # coding: utf-8
4
5  # In[1]:
6
7
8  import pandas as pd
9  from matplotlib.pyplot import *
10 import matplotlib.pyplot as plt
11
12
13 # In[2]:
14
15
16 from sklearn.metrics import roc_auc_score
17 from sklearn.metrics import roc_curve
18 from sklearn.preprocessing import LabelEncoder
19
20
21 # In[3]:
22
23
24 import numpy as np
25 from numpy import linalg
26 import cvxopt
27 import cvxopt.solvers
28
29 def linear_kernel(x1, x2):
30     return np.dot(x1, x2)
31
32 def polynomial_kernel(x, y, p=3):
33     return (1 + np.dot(x, y)) ** p
34
35 def gaussian_kernel(x, y, sigma=5.0):
36     return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))
37
38 class SVM(object):
```

```

39
40     def __init__(self, kernel=linear_kernel, C=None):
41         self.kernel = kernel
42         self.C = C
43         if self.C is not None: self.C = float(self.C)
44
45     def fit(self, X, y):
46         n_samples, n_features = X.shape
47
48         # Gram matrix
49         K = np.zeros((n_samples, n_samples))
50         for i in range(n_samples):
51             for j in range(n_samples):
52                 K[i,j] = self.kernel(X[i], X[j])
53
54         P = cvxopt.matrix(np.outer(y,y) * K)
55         q = cvxopt.matrix(np.ones(n_samples) * -1)
56         A = cvxopt.matrix(y, (1,n_samples))
57         b = cvxopt.matrix(0.0)
58
59         if self.C is None:
60             G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1)
61                               )
62             h = cvxopt.matrix(np.zeros(n_samples))
63         else:
64             tmp1 = np.diag(np.ones(n_samples) * -1)
65             tmp2 = np.identity(n_samples)
66             G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
67             tmp1 = np.zeros(n_samples)
68             tmp2 = np.ones(n_samples) * self.C
69             h = cvxopt.matrix(np.hstack((tmp1, tmp2)))
70
71         # solve QP problem
72         solution = cvxopt.solvers.qp(P, q, G, h, A, b)
73
74         # Lagrange multipliers
75         a = np.ravel(solution['x'])
76
77         # Support vectors have non zero lagrange multipliers
78         sv = a > 1e-5
79         ind = np.arange(len(a))[sv]
80         self.a = a[sv]
81         self.sv = X[sv]
82         self.sv_y = y[sv]
83         print("%d support vectors out of %d points" % (len(

```

```

        self.a), n_samples))
83
84     # Intercept
85     self.b = 0
86     for n in range(len(self.a)):
87         self.b += self.sv_y[n]
88         self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv
            ])
89     self.b /= len(self.a)
90
91     # Weight vector
92     if self.kernel == linear_kernel:
93         self.w = np.zeros(n_features)
94         for n in range(len(self.a)):
95             self.w += self.a[n] * self.sv_y[n] *
96                 self.sv[n]
97     else:
98         self.w = None
99
100     def project(self, X):
101         if self.w is not None:
102             return np.dot(X, self.w) + self.b
103         else:
104             y_predict = np.zeros(len(X))
105             for i in range(len(X)):
106                 s = 0
107                 for a, sv_y, sv in zip(self.a, self.sv_y,
108                     self.sv):
109                     s += a * sv_y * self.kernel(X[i], sv)
110             y_predict[i] = s
111             return y_predict + self.b
112
113     def predict(self, X):
114         return np.sign(self.project(X))
115
116 if __name__ == "__main__":
117     import pylab as pl
118
119     def plot_margin(X1_train, X2_train, clf):
120         def f(x, w, b, c=0):
121             return (-w[0] * x - b + c) / w[1]
122
123         pl.plot(X1_train[:,0], X1_train[:,1], "ro")
124         pl.plot(X2_train[:,0], X2_train[:,1], "bo")
125         pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

```



```

126
127
128     a0 = -4; a1 = f(a0, clf.w, clf.b)
129     b0 = 4; b1 = f(b0, clf.w, clf.b)
130     pl.plot([a0,b0], [a1,b1], "k")
131
132
133     a0 = -4; a1 = f(a0, clf.w, clf.b, 1)
134     b0 = 4; b1 = f(b0, clf.w, clf.b, 1)
135     pl.plot([a0,b0], [a1,b1], "k--")
136
137
138     a0 = -4; a1 = f(a0, clf.w, clf.b, -1)
139     b0 = 4; b1 = f(b0, clf.w, clf.b, -1)
140     pl.plot([a0,b0], [a1,b1], "k--")
141
142     pl.axis("tight")
143     pl.show()
144
145
146 def accuracy_metric(actual, predicted):
147     correct = 0
148     for i in range(len(actual)):
149         if actual[i] == predicted[i]:
150             correct += 1
151     return correct / float(len(actual)) * 100.0
152
153 def plot_contour(X1_train, X2_train, clf):
154     pl.plot(X1_train[:,0], X1_train[:,1], "ro")
155     pl.plot(X2_train[:,0], X2_train[:,1], "bo")
156     pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")
157
158     X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace
159                          (-6,6,50))
160     X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1),
161                                             np.ravel(X2))])
162     Z = clf.project(X).reshape(X1.shape)
163     pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1,
164               origin='lower')
165     pl.contour(X1, X2, Z + 1, [0.0], colors='red',
166               linewidths=1, origin='lower')
167     pl.contour(X1, X2, Z - 1, [0.0], colors='green',
168               linewidths=1, origin='lower')
169
170     pl.axis("tight")

```

```

167         pl.show()
168
169     def test_linear():
170
171         dataset = pd.read_csv("https://raw.githubusercontent.com
172         /psantul/Dataset/main/titanic.csv", delimiter = ',',
173         encoding='latin-1')
174
175         x = dataset.iloc[:,0:2].values
176         y = dataset.iloc[:, -1].values
177         print(x)
178
179         from sklearn.model_selection import train_test_split
180         X_train, X_test, y_train, y_test = train_test_split(x,
181         y, test_size = 0.15, random_state=1)
182
183         clf = SVM()
184         clf.fit(X_train, y_train)
185
186         y_predict = clf.predict(X_test)
187         #print(y_predict)
188         correct = np.sum(y_predict == y_test)
189         print("%d out of %d predictions correct" % (correct,
190         len(y_predict)))
191         accuracy = accuracy_metric(y_test, y_predict)
192         print("Accuracy of Model is:")
193         print(accuracy)
194
195         plot_margin(X_train[y_train==1], X_train[y_train==-1],
196         clf)
197
198         roc_auc4 = roc_auc_score(y_test, y_predict)
199         fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
200         plt.figure()
201         plt.plot(fpr4, tpr4, label='Support Vector Machines
202         Classifier (area=%0.2f)' % roc_auc4)
203         plt.plot([0, 1], [0, 1], 'r--')
204         plt.xlim([0.0, 1.0])
205         plt.ylim([0.0, 1.0])
206         plt.xlabel('False Positive Rate')
207         plt.ylabel('True Positive Rate')
208         plt.title('ROC for SVM with Linear Kernel')
209         plt.legend(loc="lower right")

```

```

207     plt.savefig('Log_ROC_SVM')
208     plt.show()
209
210
211
212
213     def test_non_linear():
214
215         dataset = pd.read_csv("https://raw.githubusercontent.com
216         /psantul/Dataset/main/titanic.csv", delimiter = ',',
217         encoding='latin-1')
218
219         x = dataset.iloc[:,0:2].values
220         y = dataset.iloc[:, -1].values
221
222         from sklearn.model_selection import train_test_split
223         X_train, X_test, y_train, y_test = train_test_split(x,
224         y, test_size = 0.15, random_state=1)
225
226         clf = SVM(polynomial_kernel)
227         clf.fit(X_train, y_train)
228
229         y_predict = clf.predict(X_test)
230         correct = np.sum(y_predict == y_test)
231         #print(y_predict)
232         print("%d out of %d predictions correct" % (correct,
233         len(y_predict)))
234
235         accuracy = accuracy_metric(y_test, y_predict)
236         print(accuracy)
237
238         plot_contour(X_train[y_train==1], X_train[y_train
239         ==-1],
240         clf)
241
242         roc_auc4 = roc_auc_score(y_test, y_predict)
243         fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
244         plt.figure()
245         plt.plot(fpr4, tpr4, label='Support Vector Machines
246         Classifier (area=%0.2f)' % roc_auc4)
247         plt.plot([0, 1], [0, 1], 'r--')
248         plt.xlim([0.0, 1.0])
249         plt.ylim([0.0, 1.0])

```

```

246     plt.xlabel('False_Positive_Rate')
247     plt.ylabel('True_Positive_Rate')
248     plt.title('ROC_for_SVM_with_Polynomial_Kernel')
249     plt.legend(loc="lower_right")
250     plt.savefig('Log_ROC_SVM')
251     plt.show()
252
253
254     def test_soft():
255
256         dataset = pd.read_csv("https://raw.githubusercontent.com
257         /psantul/Dataset/main/titanic.csv", delimiter = ',',
           encoding='latin-1')
258
259         x = dataset.iloc[:,1:3].values
260         y = dataset.iloc[:, -1].values
261
262         from sklearn.model_selection import train_test_split
263         X_train, X_test, y_train, y_test = train_test_split(x,
           y, test_size = 0.15, random_state=1)
264
265         clf = SVM(C=1000.1)
266         clf.fit(X_train, y_train)
267
268         y_predict = clf.predict(X_test)
269         #print(y_predict)
270         correct = np.sum(y_predict == y_test)
271         print("%d out of %d predictions correct" % (correct,
           len(y_predict)))
272         #print("Accuracy: %.3f (%.3f)" % (y_test.mean(),
           y_predict.std()))
273         accuracy = accuracy_metric(y_test, y_predict)
274         print(accuracy)
275
276         plot_contour(X_train[y_train==1], X_train[y_train
           ==-1],
277         clf)
278
279         roc_auc4 = roc_auc_score(y_test, y_predict)
280         fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
281         plt.figure()
282         plt.plot(fpr4, tpr4, label='Support_Vector_Machines_
           Classifier(area=%0.2f)' % roc_auc4)
283         plt.plot([0, 1], [0, 1], 'r--')

```

```
284     plt.xlim([0.0, 1.0])
285     plt.ylim([0.0, 1.0])
286     plt.xlabel('False Positive Rate')
287     plt.ylabel('True Positive Rate')
288     plt.title('ROC for SVM with Gaussian Kernel')
289     plt.legend(loc="lower right")
290     plt.savefig('Log_ROC_SVM')
291     plt.show()
292
293
294     test_linear()
295     test_non_linear()
296     test_soft()
```

Output:

Linear Kernel: Accuracy of Model is: 64.65%

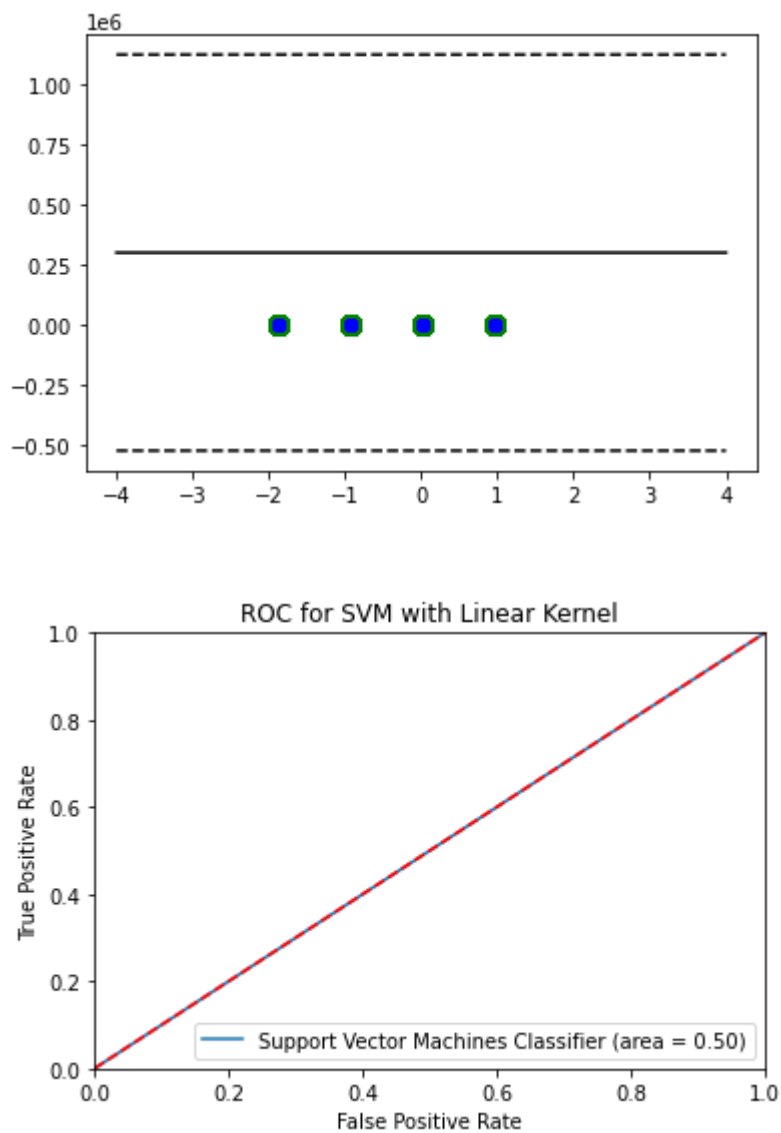


Fig: Observation SVM Linear Kernel Result

Polynomial Kernel: Accuracy of Model is: 65.25%

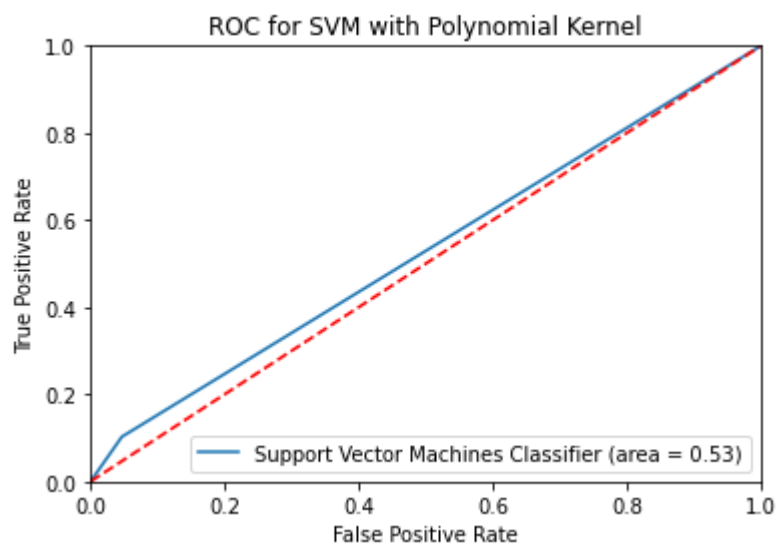
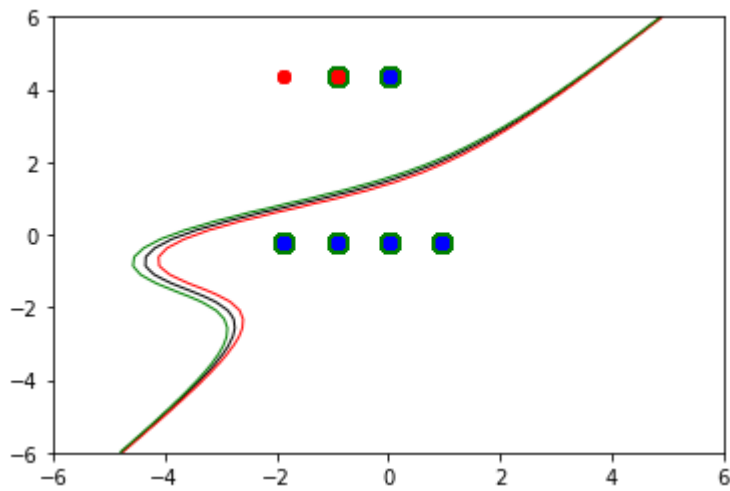


Fig: Observation SVM Polynomial Kernel Result

Gaussian Kernel: Accuracy of Model is: 77.94%

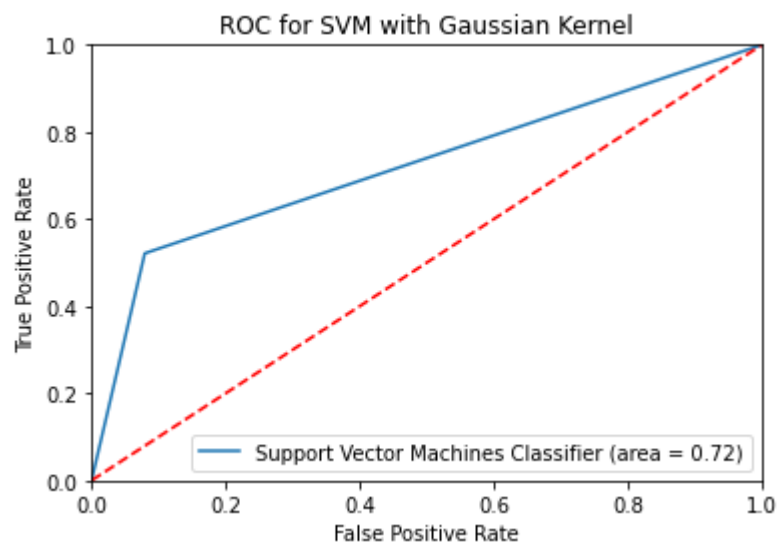
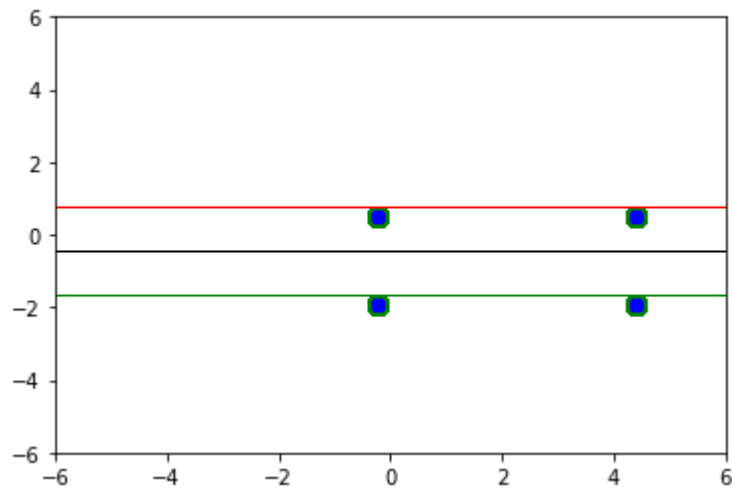


Fig: Observation SVM Gaussian Kernel Result