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

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/Breast+Cancer+ Wiscon sin+ 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:

```

#!/usr/bin/env python
# coding: utf-8

# In[1]:


#Q-1. Create an ANN with one hidden layer and do classification on
    the datasets given in the link:


# In[2]:


import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

```

```


#Check observation one by one different dataset and see observations.

data = pd.read_csv("https://raw.githubusercontent.com/psantul/Dataset/main/data.csv", encoding='latin-1')
#data = pd.read_csv("https://raw.githubusercontent.com/psantul/Dataset/main/titanic.csv",delimiter = ',', encoding='latin-1')
print(data.shape)
data.head()


# In[3]:


x = data.iloc[:,3:].values
y = data.iloc[:,1].values


# In[4]:


encoder = LabelEncoder()
y = encoder.fit_transform(y)


# In[5]:


X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)


# In[6]:


sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)


# In[7]:

```

```python

class NeuralNetwork:

    def __init__(self,X, y, X_test, y_test, hidden_nodes=1, learning_rate=0.1, epochs=5000):

        #data
        self.y = y[:,None]
        self.X = X

        self.X_test = X_test
        self.y_test = y_test


        #parameters
        np.random.seed(4)
        self.input_nodes = len(X[0])
        self.hidden_nodes = hidden_nodes
        self.output_nodes = self.y.shape[1]
        self.learning_rate = learning_rate

        #init weights
        self.w1 = 2*np.random.random((self.input_nodes, self.hidden_nodes)) - 1
        self.w2 = 2*np.random.random((self.hidden_nodes, self.output_nodes)) - 1

        self.train(epochs)
        self.test()

    def sigmoid(self,X):
        return (1/(1+np.exp(-X)))

    def sigmoid_prime(self,X):
        return X * (1 - X)

    def train(self, epochs):

        for e in range(epochs):

            #FORWARD PROPAGATION

            #hidden layer
            # W1(398,30) X(30,12)
            l1 = self.sigmoid(np.dot(self.X, self.w1))
```

```

            #output layer
            #l1(398,12) W2(12,1)
            l2 = self.sigmoid(np.dot(l1, self.w2))

            # BACKPROPAGATION

            #calculate how far off our prediciton was
            error = self.y-l2

            #calculate how far off each layer is
            l2_delta = error * self.sigmoid_prime(l2)
            l1_delta = l2_delta.dot(self.w2.T) * self.sigmoid_prime(l1)

            #update weights with our newly found error values
            self.w2 = np.add(self.w2, l1.T.dot(l2_delta) * self.learning_rate)
            self.w1 = np.add(self.w1, self.X.T.dot(l1_delta) * self.learning_rate)

        #print('Error:', (abs(error)).mean())

    def test(self):
        correct = 0
        pred_list = []

        #replicate feedforward network for testing
        l1 = self.sigmoid(np.dot(self.X_test, self.w1))
        l2 = self.sigmoid(np.dot(l1, self.w2))

        #loop through all of the outputs of layer 2
        for i in range(len(l2)):
            if l2[i] >= 0.5:
                pred = 1
            else:
                pred = 0

            if pred == self.y_test[i]:
                correct += 1

            pred_list.append(pred)
        print(pred_list)
        print(y_test)
        correct = np.sum(pred_list == y_test)
```

```
        print("%d out of %d predictions correct" % (correct, len(pred_list)))

        print("Test Accuracy: ", ((correct/len(y_test))*100),'%')


# In[8]:


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

```

#!/usr/bin/env python
# coding: utf-8



#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


import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler


 Check observation one by one different dataset remove
 comment part and see observations


 data = pd.read_csv("https://raw.githubusercontent.com
␣/psantul/Dataset/main/data.csv", encoding='latin-1')
#data = pd.read_csv("https://raw.githubusercontent.com
/psantul/Dataset/main/titanic.csv",delimiter␣=␣',',
encoding='latin-1')
print(data.shape)
data.head()

x␣=␣data.iloc[:,3:].values
y␣=␣data.iloc[:,1].values
```

```

encoder = LabelEncoder()
y = encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size = 0.2, random_state = 0)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

np.unique(y_train)

nn_input_dim = X_train.shape[1]
nn_output_dim = len(np.unique(y_train))
lr = 0.01

# helper function to calculate total loss on the dataset
def calculate_loss(model, X, y):
  num_examples = X.shape[0]
  W1 = model['W1']
  b1 = model['b1']
  W2 = model['W2']
  b2 = model['b2']
  # forward propagation to calculate out predictions
  z1 = X.dot(W1) + b1
  a1 = np.tanh(z1)
  z2 = a1.dot(W2) + b2
  exp_scores = np.exp(z2)
  probs = exp_scores / np.sum(exp_scores, axis=1,
  keepdims=True)
  # Calculating the loss
  corect_logprobs = -np.log(probs[range(num_examples), y])
  data_loss = np.sum(corect_logprobs)
  return 1. / num_examples * data_loss

def predict(model, x):
  W1 = model['W1']
  b1 = model['b1']
  W2 = model['W2']
  b2 = model['b2']


  # Forward propagation
```

```
  z1 = x.dot(W1) + b1
  a1 = np.tanh(z1)
  z2 = a1.dot(W2) + b2
  exp_scores = np.exp(z2)
  probs = exp_scores / np.sum(exp_scores, axis=1,
  keepdims=True)
  return np.argmax(probs, axis=1)

# (Relu,logistic sigmoid,tanh,leaky Relu)

def relu(x):
  x[x<0]=0
  return x

def logsig(x):
  return 1/(1 + np.exp(-x))

def tanh(x):
  return np.tanh(x)

def leaky_relu(x):
  x[x<0] *= 0.01
  return x

def build_model(X, y, nn_hdim, num_passes = 10000,
print_loss=False):

  num_examples = X.shape[0]
  np.random.seed(0)
  W1 = np.random.rand(nn_input_dim, nn_hdim) /
  np.sqrt(nn_input_dim)
  b1 = np.zeros((1, nn_hdim))
  W2 = np.random.rand(nn_hdim, nn_output_dim) /
  np.sqrt(nn_hdim)
  b2 = np.zeros((1, nn_output_dim))
  model = {}
  losses = []

  # Gradient descent. for each batch
  for i in range(0, num_passes):


    # Forward propagation
    z1 = X.dot(W1) + b1
    a1 = tanh(z1)
```

```
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(np.array(exp_scores),
    axis=1, keepdims=True)

    # Backpropagation
    delta3 = np.array(probs)
    delta3[range(num_examples), y] -= 1
    dW2 = (a1.T).dot(delta3)
    db2 = np.sum(delta3, axis=0, keepdims=True)
    delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
    dW1 = np.dot(X.T, delta2)
    db1 = np.sum(delta2, axis=0)

    # Gradient descent parameter update
    W1 += -lr * dW1
    b1 += -lr * db1
    W2 += -lr * dW2
    b2 += -lr * db2


    # Assign new parameters to the model
    model = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

    if print_loss and i % 1000 == 0:
      loss = calculate_loss(model, X, y)
      losses.append(loss)
      print("Loss after iteration %i: %f" % (i, loss))

  return model, losses

x = []
y = []

for nodes in [64, 128, 256, 512]:
  print('\n for {} number of nodes'.format(nodes))
  model, loss = build_model((X_train),(y_train), nodes,
  print_loss=True)
  preds = predict(model, X_test)
  acc = ((preds == y_test).sum() / len(y_test)) * 100
  x.append(nodes)
  y.append(acc)

plt.xlabel('number of nodes')
plt.ylabel('accuracy')
```

```
plt.title('using {} activation function'.format('leaky_relu'))
#place of relu changes logsig,tanh,leaky_relu:
Different loss result shown in graph and accuracy

plt.plot(x, y)
```

Output: (64,128,256,512)

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 1 1 0]
67 out of 114 predictions correct
Test Accuracy:  58.77192982456141 %
```

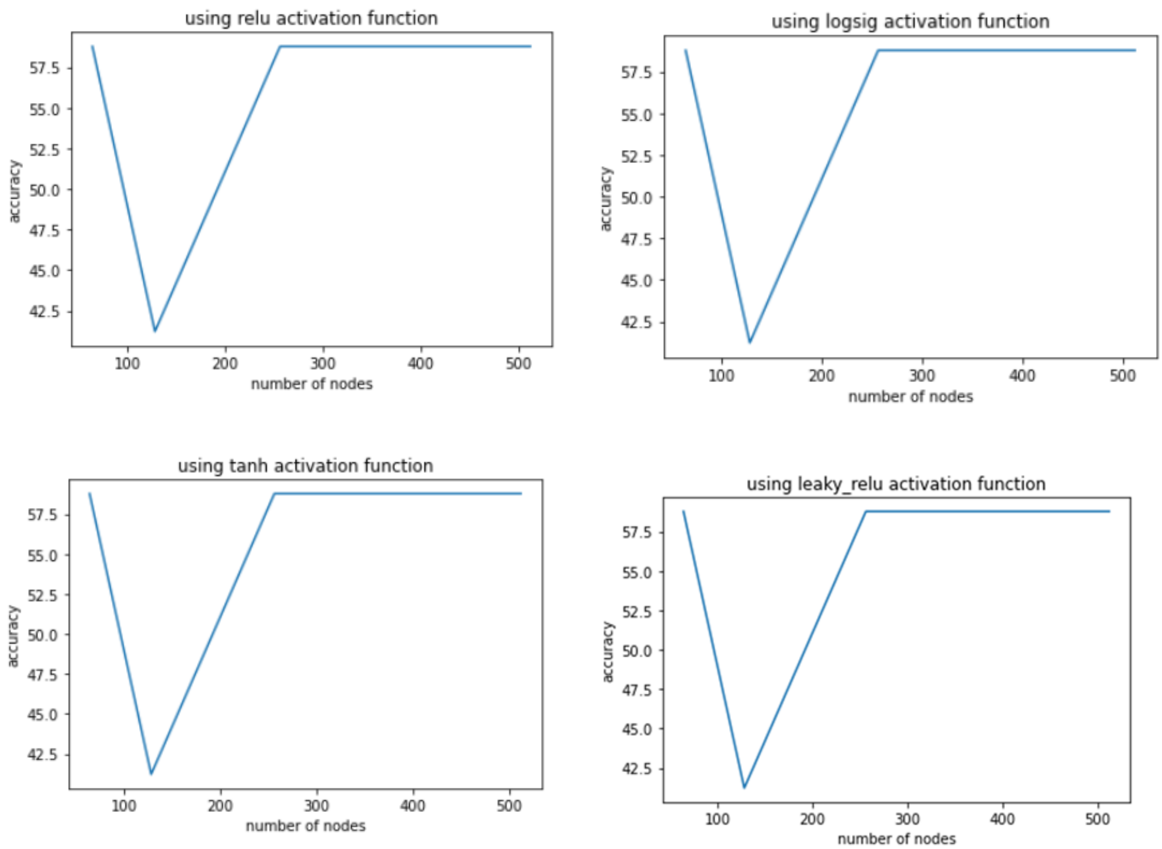
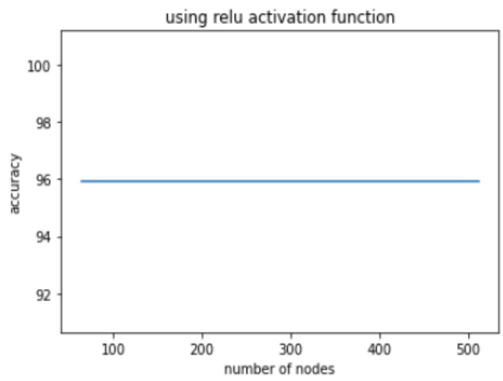
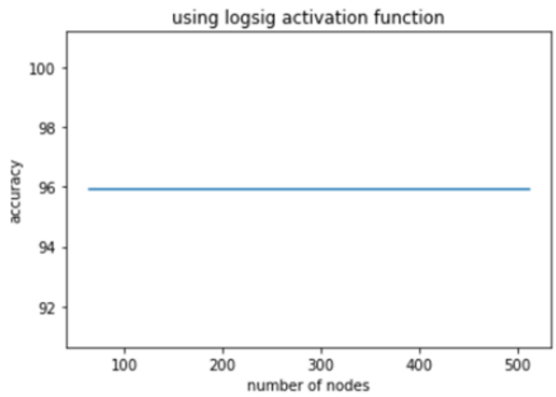


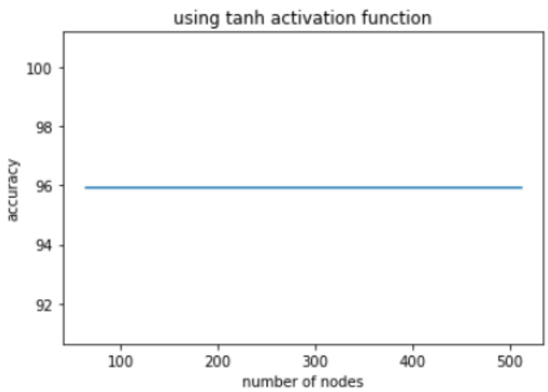
Fig: Observation of Dateset-1: https://raw.githubusercontent.com/psantul/Dataset/main/data.csv

Output: (64,128,256,512)

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
423 out of 441 predictions correct
Test Accuracy:  95.91836734693877 %
```







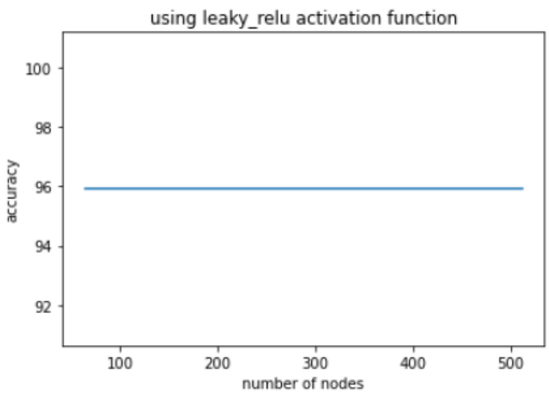


Fig: Observation of dataset 2:https://raw.githubusercontent.com/psantul/Dataset/main/titanic.csv

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

```

# mlp for regression with mse loss function
from sklearn.datasets import make_regression
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from matplotlib import pyplot
from sklearn.preprocessing import LabelEncoder
import pandas as pd

data = pd.read_csv("https://raw.githubusercontent.com/psantul/
Dataset/main/titanic.csv",delimiter = ',', encoding='latin-1')
print(data.shape)
data.head()

X = data.iloc[:,3:].values
y = data.iloc[:,1].values

encoder = LabelEncoder()
y = encoder.fit_transform(y)

# split into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)

import tensorflow as tf
from tensorflow.keras.models import Sequential

# define model
model = Sequential()
model.add(Dense(25, input_dim=1, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(1, activation='linear'))
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='mean_squared_error', optimizer=opt)

```

```
X_test.shape, y_test.shape

X.shape, y.shape

# fit model
history = model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=100, verbose=0)

# evaluate the model
train_mse = model.evaluate(X_train, y_train, verbose=0)
test_mse = model.evaluate(X_test, y_test, verbose=0)

print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
pyplot.title('Loss / Mean Squared Error')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

"""#Kullback Leibler Divergence Loss"""

from keras.utils import to_categorical
# one hot encode output variable
y = to_categorical(y)

# define model
model = Sequential()
model.add(Dense(50, input_dim=1, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(3, activation='softmax'))

# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='kullback_leibler_divergence',
optimizer=opt, metrics=['accuracy'])

# fit model
history = model.fit(X_train, y_train,
validation_data= (X_test, y_test), epochs=100, verbose=0)

# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```

# plot loss during training
pyplot.figure(figsize=(10,8))
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()

# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()

"""#Sparse Multiclass Cross-Entropy Loss"""

# define model
model = Sequential()
model.add(Dense(50, input_dim=1,activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(3, activation='softmax'))
# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='sparse_categorical_crossentropy',
optimizer=opt, metrics=['accuracy'])
# fit model
history = model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=100, verbose=0)
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
```

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

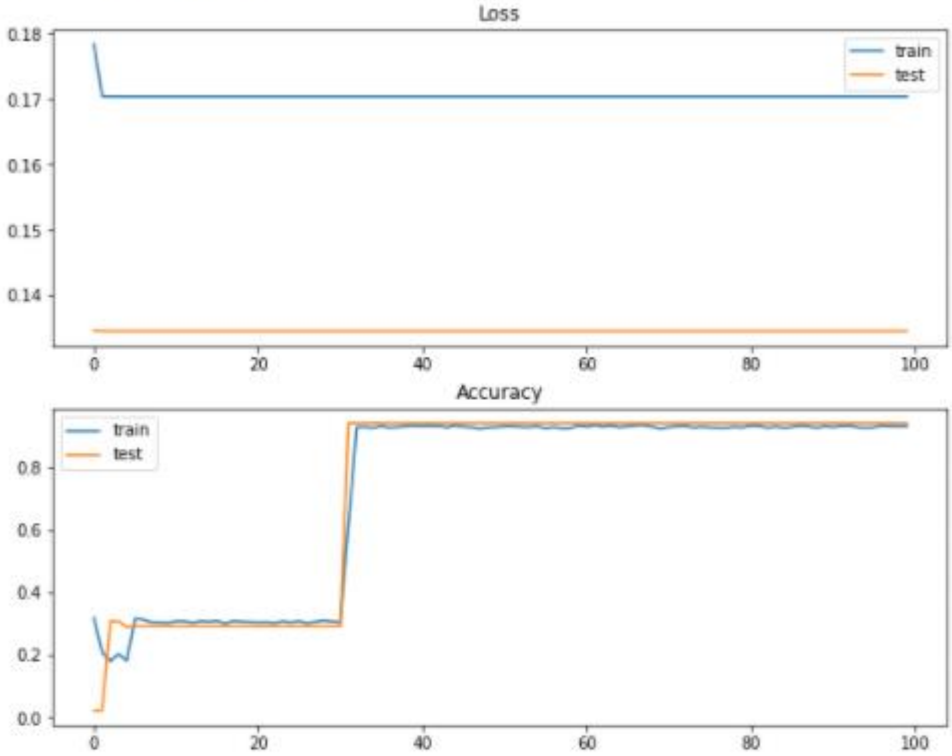
Output:

a) Multi-Class Cross-Entropy Loss:

```
Train: 0.051, Test: 0.041
```

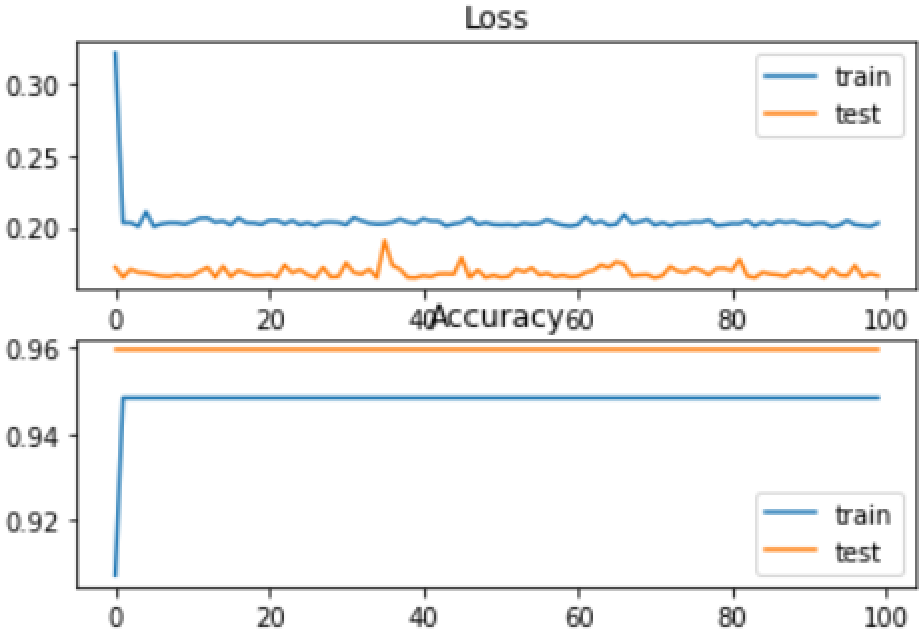
b) Sparse Multiclass Cross-Entropy Loss:

```
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

```

#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
from matplotlib.pyplot import *
import matplotlib.pyplot as plt


# In[2]:


from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.preprocessing import LabelEncoder


# In[3]:


import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers

def linear_kernel(x1, x2):
    return np.dot(x1, x2)

def polynomial_kernel(x, y, p=3):
    return (1 + np.dot(x, y)) ** p

def gaussian_kernel(x, y, sigma=5.0):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

class SVM(object):
```

```

    def __init__(self, kernel=linear_kernel, C=None):
        self.kernel = kernel
        self.C = C
        if self.C is not None: self.C = float(self.C)

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Gram matrix
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = self.kernel(X[i], X[j])

        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
        print("%d support vectors out of %d points" % (len(
```

```
           self.a), n_samples))

        # Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
        self.b /= len(self.a)

        # Weight vector
        if self.kernel == linear_kernel:
            self.w = np.zeros(n_features)
            for n in range(len(self.a)):
                self.w += self.a[n] * self.sv_y[n] *
                self.sv[n]
        else:
            self.w = None

    def project(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a, sv_y, sv in zip(self.a, self.sv_y,
                self.sv):
                    s += a * sv_y * self.kernel(X[i], sv)
                y_predict[i] = s
            return y_predict + self.b

    def predict(self, X):
        return np.sign(self.project(X))

if __name__ == "__main__":
    import pylab as pl

    def plot_margin(X1_train, X2_train, clf):
        def f(x, w, b, c=0):
            return (-w[0] * x - b + c) / w[1]

        pl.plot(X1_train[:,0], X1_train[:,1], "ro")
        pl.plot(X2_train[:,0], X2_train[:,1], "bo")
        pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")
```

```python


    a0 = -4; a1 = f(a0, clf.w, clf.b)
    b0 = 4; b1 = f(b0, clf.w, clf.b)
    pl.plot([a0,b0], [a1,b1], "k")


    a0 = -4; a1 = f(a0, clf.w, clf.b, 1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, 1)
    pl.plot([a0,b0], [a1,b1], "k--")


    a0 = -4; a1 = f(a0, clf.w, clf.b, -1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, -1)
    pl.plot([a0,b0], [a1,b1], "k--")

    pl.axis("tight")
    pl.show()


def accuracy_metric(actual, predicted):
  correct = 0
  for i in range(len(actual)):
    if actual[i] == predicted[i]:
      correct += 1
  return correct / float(len(actual)) * 100.0

def plot_contour(X1_train, X2_train, clf):
    pl.plot(X1_train[:,0], X1_train[:,1], "ro")
    pl.plot(X2_train[:,0], X2_train[:,1], "bo")
    pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

    X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace(-6,6,50))
    X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
    Z = clf.project(X).reshape(X1.shape)
    pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')
    pl.contour(X1, X2, Z + 1, [0.0], colors='red',
    linewidths=1, origin='lower')
    pl.contour(X1, X2, Z - 1, [0.0], colors='green', linewidths=1, origin='lower')

    pl.axis("tight")
```

```
        pl.show()

    def test_linear():

        dataset = pd.read_csv("https://raw.githubusercontent.com
        /psantul/Dataset/main/titanic.csv",delimiter = ',',
    encoding='latin-1')

        x = dataset.iloc[:,0:2].values
        y = dataset.iloc[:,-1].values
        print(x)

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(x,
            y,test_size = 0.15, random_state=1)

        clf = SVM()
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
        #print(y_predict)
        correct = np.sum(y_predict == y_test)
        print("%d out of %d predictions correct" % (correct,
            len(y_predict)))
        accuracy = accuracy_metric(y_test, y_predict)
        print("Accuracy of Model is : ")
        print(accuracy)

        plot_margin(X_train[y_train==1], X_train[y_train==-1],
        clf)


        roc_auc4 = roc_auc_score(y_test, y_predict)
        fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
        plt.figure()
        plt.plot(fpr4, tpr4, label='Support Vector Machines 
            Classifier (area = %0.2f)' % roc_auc4)
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for SVM with Linear Kernel')
        plt.legend(loc="lower right")
```

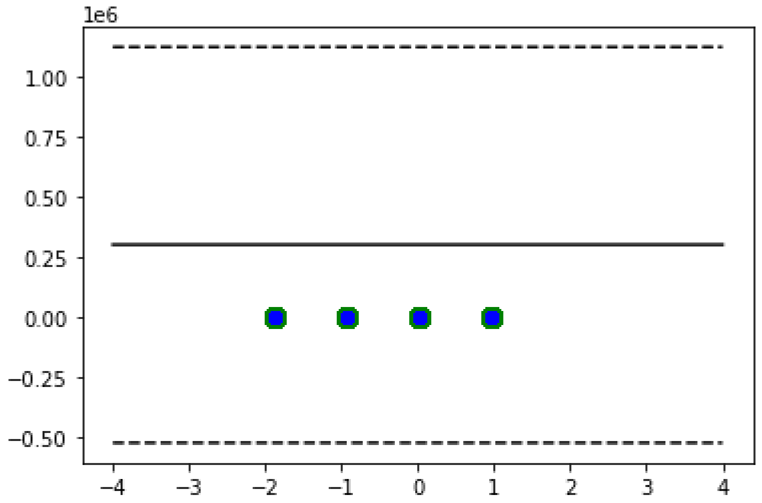
```
        plt.savefig('Log_ROC_SVM')
        plt.show()




    def test_non_linear():

        dataset = pd.read_csv("https://raw.githubusercontent.
           com
␣␣␣␣␣␣␣␣/psantul/Dataset/main/titanic.csv",delimiter = ',',
    encoding='latin-1')

        x = dataset.iloc[:,0:2].values
        y = dataset.iloc[:,-1].values

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(x,
           y,test_size = 0.15, random_state=1)

        clf = SVM(polynomial_kernel)
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
        correct = np.sum(y_predict == y_test)
        #print(y_predict)
        print("%d␣out␣of␣%d␣predictions␣correct" % (correct,
           len(y_predict)))

        accuracy = accuracy_metric(y_test, y_predict)
        print(accuracy)

        plot_contour(X_train[y_train==1], X_train[y_train
           ==-1],
        clf)


        roc_auc4 = roc_auc_score(y_test, y_predict)
        fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
        plt.figure()
        plt.plot(fpr4, tpr4, label='Support␣Vector␣Machines␣
           Classifier␣(area␣=␣%0.2f)' % roc_auc4)
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
```

```python
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for SVM with Polynomial Kernel')
        plt.legend(loc="lower right")
        plt.savefig('Log_ROC_SVM')
        plt.show()


    def test_soft():

        dataset = pd.read_csv("https://raw.githubusercontent.com
         /psantul/Dataset/main/titanic.csv",delimiter = ',', encoding='latin-1')

        x = dataset.iloc[:,1:3].values
        y = dataset.iloc[:,-1].values

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(x, y,test_size = 0.15, random_state=1)

        clf = SVM(C=1000.1)
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
        #print(y_predict)
        correct = np.sum(y_predict == y_test)
        print("%d out of %d predictions correct" % (correct, len(y_predict)))
        #print("Accuracy: %.3f (%.3f)" % (y_test.mean(), y_predict.std()))
        accuracy = accuracy_metric(y_test, y_predict)
        print(accuracy)

        plot_contour(X_train[y_train==1], X_train[y_train==-1],
        clf)

        roc_auc4 = roc_auc_score(y_test, y_predict)
        fpr4, tpr4, thresholds4 = roc_curve(y_test, y_predict)
        plt.figure()
        plt.plot(fpr4, tpr4, label='Support Vector Machines Classifier (area = %0.2f)' % roc_auc4)
        plt.plot([0, 1], [0, 1],'r--')
```

```
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for SVM with Gaussian Kernel')
        plt.legend(loc="lower right")
        plt.savefig('Log_ROC_SVM')
        plt.show()


    test_linear()
    test_non_linear()
    test_soft()
```

Output:

Linear Kernel: Accuracy of Model is: 64.65%



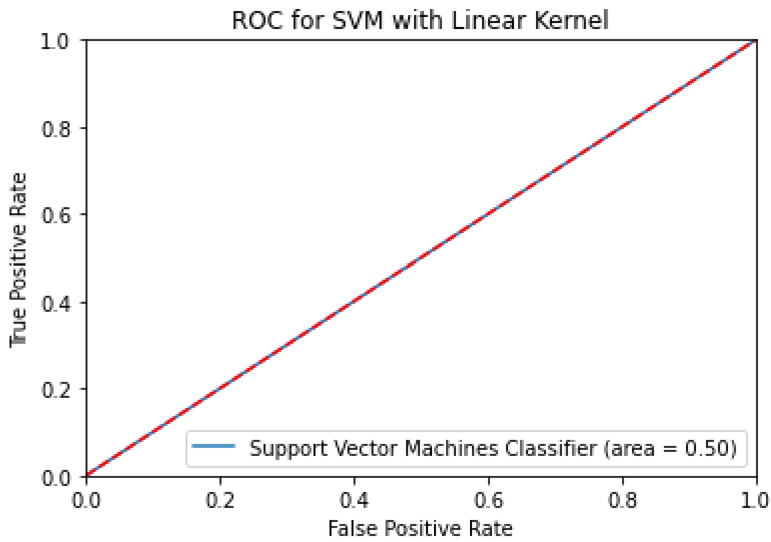
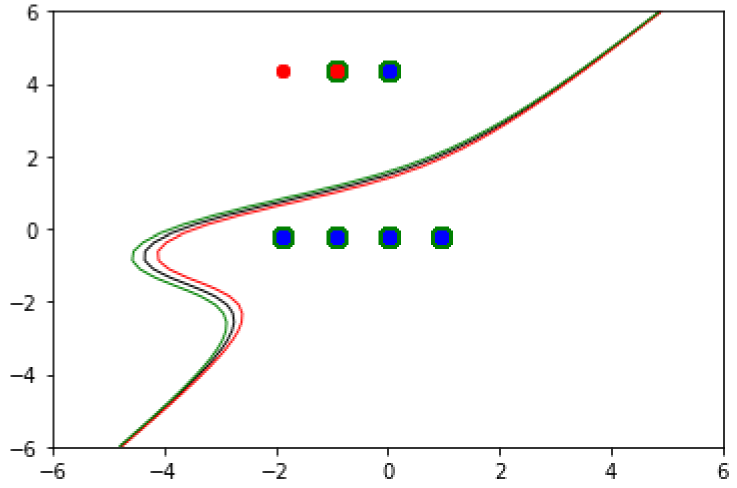


Fig: Observation SVM Linear Kernel Result

Polynomial Kernel: Accuracy of Model is: 65.25%



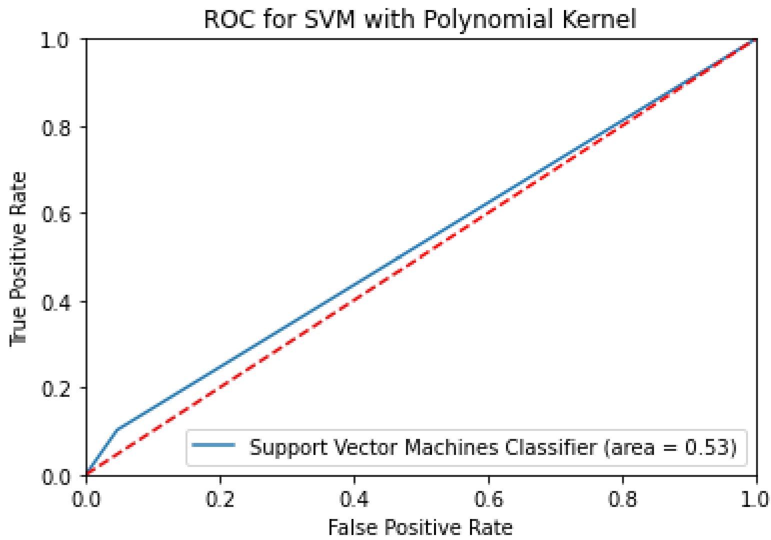
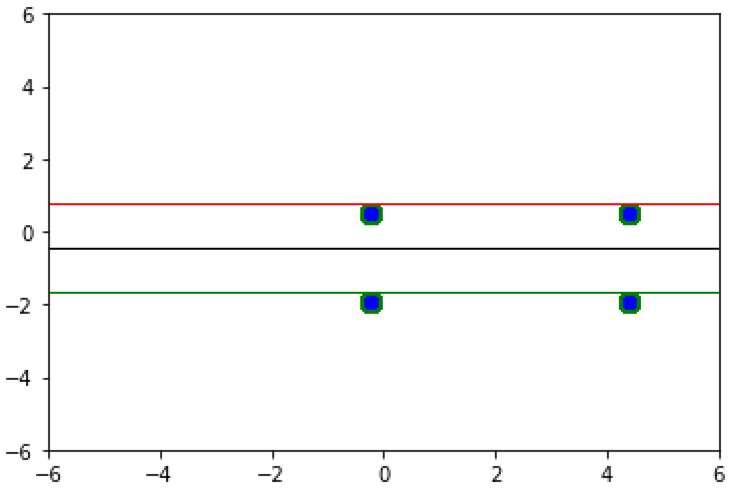


Fig: Observation SVM Polynomial Kernel Result

Gaussian Kernel: Accuracy of Model is: 77.94%



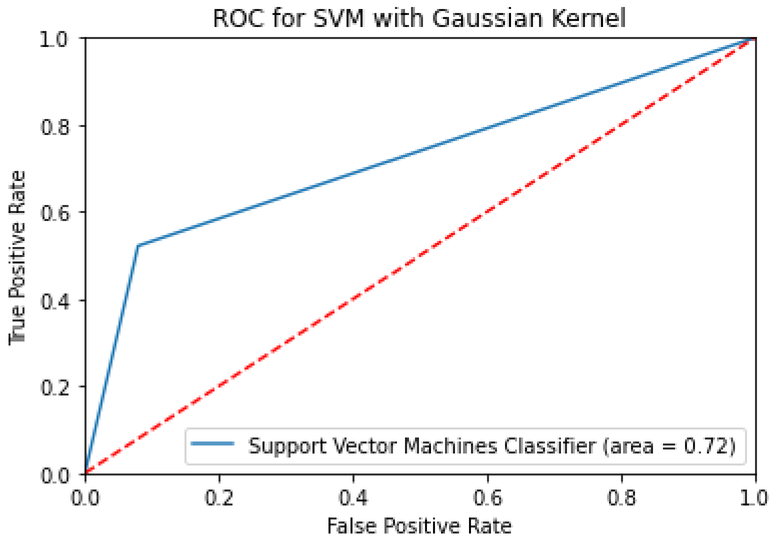


Fig: Observation SVM Gaussian Kernel Result