CS 509: Pattern Recognition

SM ZOBAED-C00300901
Assignment-2

April 28, 2019

1. Data Generator Building: I generate dataset that we used for clustering. To this end, I concatenate uniform distributions of two data and attach a label. With that I further concatenate a Gaussian data distribution and attach a different label for them. Thus I generate 'L' shape data that is close to our lecture note.

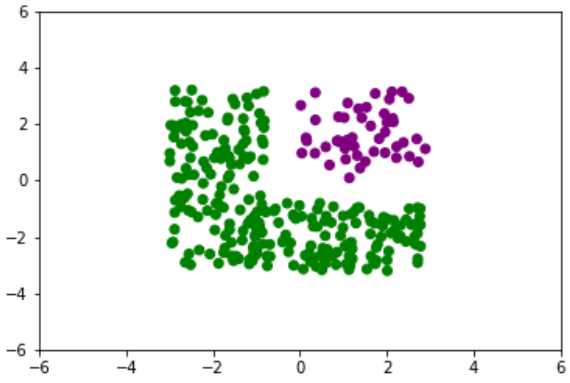

Figure 1: Dataset

2. I partition the data into training and test. I keep 80% data as training, 20% data as test data and saved training data, training data labels, test data, and test data labels in four different *.npy* files.

Listing 1: Insert code directly in your document

```
train_data, test_data, train_label, test_label = train_test_split(tot_data,
tot_data_label, test_size=0.20, random_state=50)
```

3. We plot the whole dataset by fitting through *perceptron* algorithm and draw decision boundary to understand the linearly separability of the dataset. We represent it in Figure 2. From through which it visible that data is not linearly separable.

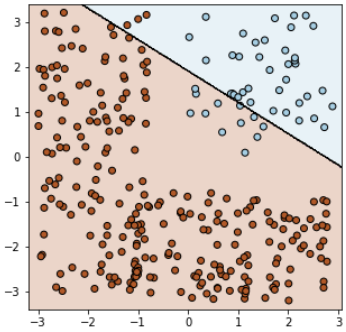

Figure 2: Verifying linearly separability

4. I have used four classifiers to understand which is the best performer. Their name, generated decision boundaries, and accuracy are mentioned below.

   - Linear SVC (Figure 3) Obtained accuracy: 96.67%

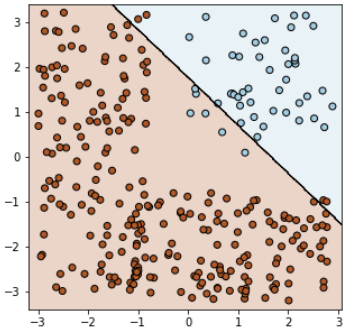

Figure 3: Linear SVC

   - Logistic Regression(Figure 4) Obtained accuracy: 95.5%

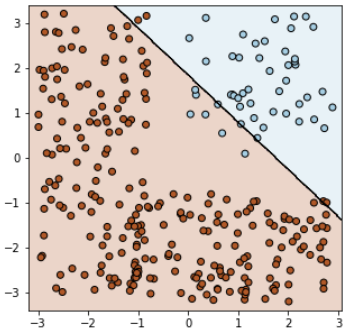

Figure 4: Logistic Regression

- SGD Classifier (Figure 5) Obtained accuracy: 88.33%

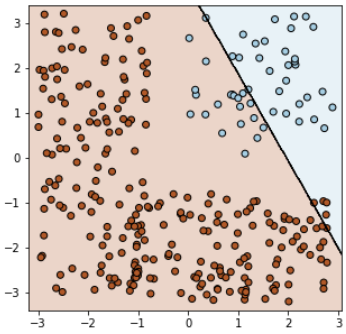

Figure 5: SGD Classifier

- : Perceptron (Figure 6) Obtained accuracy: 93.33%

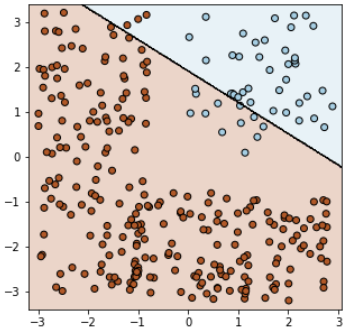

Figure 6: Perceptron

Comparing the obtained accuracy of the aforementioned four different classifiers (Figure 3, 4, 5, 6), we conclude that *Lienar SVC* classifier is the best for the considered dataset.

5. (a) Perceptron using sklearn: I call perceptron classifier from scikit learn and train the classifier with training data. Then I used my test data to predict their labels and finally obtain accuracy score by comparing the predicted labels with actual labels.

Listing 2: Code Snippet: Perceptron.

```
print('Perceptron')
X=np.load('train_data.npy')
y=np.load('train_data_label.npy')
c_all = Perceptron(tol=1e-3, random_state=0)
c_all.fit(X, y)

X_test=np.load('test_data.npy')
y_test=np.load('test_data_label.npy')

print("Accuracy:  ",c_all.score(X_test, y_test))
```

(b) Pseudo-inverse linear classifier using Python code

Listing 3: Code Snippet: Pseudo-inverse Linear Classifier.

```
# input values
x = np.load('train_data.npy')
# observed values
z = np.load('train_data_label.npy')

# data matrix
xdata = np.concatenate((x.reshape(x.shape[0], 2), np.ones(
        x.shape[0]).reshape(x.shape[0], 1)), axis=1)
# pseudo-inverse
xtx = np.matmul(xdata.transpose(), xdata)
xtxinv = np.linalg.inv(xtx)
xinv = np.matmul(xtxinv, xdata.transpose())
w = np.matmul(xinv, z)
print("Weight vector : ", w)

score=accuracy()
print ("psuedo inverse:" ,score)
```

(c) Batch Gradient Descent using python code

Batch gradient descent classifier updates weights using the whole training set in each iteration. I write a class named 'GradientDescent.py'. Then I create instance of the class and use fit function to train the classifier and calculate accuracy score.

Listing 4: Code Snippet: Class file Gradient Descent.

```
class BatchGradientDescent():
    def __init__(self, eta, n_iter):
        self.eta=eta
        self.n_iter=n_iter

    #def calculate_accuracy():




    def fit(self, X, y):
        w = np.random.randn(X.shape[1]) # initialize weights
        N = X.shape[0] #Data Length
        for i in range(self.n_iter):
            error = np.matmul(X, w) - y
            gradient = 2.0 * np.matmul(X.transpose(), error) / N
            w = w - self.eta * gradient
        print("Calculated Weight : ", w)
        testData = np.load('test_data.npy')
        testLabel = np.load('test_data_label.npy')

        tdata = np.concatenate(
            (testData.reshape(testData.shape[0], 2),
            np.ones(testData.shape[0]).
            reshape(testData.shape[0], 1)), axis=1)
```

```
cnt_right_prediction = 0
cnt_wrong_prediction = 0

for idx, point in enumerate(tdata):
    predicted_label = np.matmul(point, w)
    actual_label = testLabel[idx]

    if (actual_label<0 and predicted_label>0) or
    (actual_label>0 and predicted_label<0):
        cnt_wrong_prediction +=1
    else:
        cnt_right_prediction +=1


accuracy = (cnt_right_prediction)/(
cnt_right_prediction+cnt_wrong_prediction )
print ("Total Wrong Predictions: ", cnt_wrong_prediction )
print ("Total Right Predictions: ", cnt_right_prediction )
print ("Accuracy: ", accuracy)
```

I use 500 iterations and .1 as learning factor.

Listing 5: Code Snippet: Batch Gradient Descent.

```
from GradientDescent import BatchGradientDescent
x = np.load('train_data.npy')
y=np.load('train_data_label.npy')
X= np.concatenate( (x.reshape(x.shape[0],2), np.ones(x.shape[0]).
reshape(x.shape[0],1)), axis=1  )

cl2= BatchGradientDescent(.1, 500) #Learning factor, number of iterations.
cl2.fit(X,y)
```

(d) Mini Batch Gradient Descent using python code

Utilizing this classifier, weight is updated by using a randomly selected subset of the training set at each iteration.

Like the previous one, I write a class for mini batch gradient desecent. Then I create instance of the class and use fit function to train the classifier and calculate accuracy score.

Listing 6: Code Snippet: Mini Batch Gradient Descent .

```
from GradientDescent import miniBatchGradientDescent
x = np.load('train_data.npy')
y=np.load('train_data_label.npy')
X= np.concatenate( (x.reshape(x.shape[0],2),
    np.ones(x.shape[0]).reshape(x.shape[0],1)), axis=1  )
if len(sys.argv) == 1:
    batch_size = X.shape[0]//10          # 10 batches
else:
    batch_size = int(sys.argv[1])

print("Batch Size: ", batch_size)
cl3= miniBatchGradientDescent(.1, 100)
cl3.fit(X,y)
```

(e) Mini Batch Gradient Descent using tensorflow

Listing 7: Code Snippet: Mini Batch Gradient Descent using tf .

```
x = np.load("train_data.npy")
zdata = np.load("train_data_label.npy")
xdata= np.concatenate( (x.reshape(x.shape[0],2),
    np.ones(x.shape[0]).reshape(x.shape[0],1)), axis=1  )
N, p = xdata.shape
 # no length yet in placeholder
X = tf.placeholder(tf.float32, shape=(None, p), name="X")
z = tf.placeholder(tf.float32, shape=(None, 1), name="z")

# minibatch gradient descent solution
if len(sys.argv) == 1:
    batch_size = xdata.shape[0] // 10 # 10 batches
else:
    batch_size = int(sys.argv[1])
print("Batch size: ", batch_size)
eta = 0.1
n_iterations = 100
Xt = tf.transpose(X)
weights = tf.Variable(tf.random_uniform([p, 1], -1.0, 1.0),
name="weights")
y = tf.matmul(X, weights, name="predictions")
error = y - z
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = tf.gradients(mse, [weights])[0]
training_step = tf.assign(weights, weights - eta * gradients)
init = tf.global_variables_initializer()
from datetime import datetime
# current time as string
ctime = datetime.utcnow().strftime("%Y%m%d%H%M%S")
root_logdir = "tf_logs"
logdir = "{}/run-{}/".format(root_logdir, "x")


# create a node that evaluates the mse value and
#write to binary log string
mse_summary = tf.summary.scalar('mse', mse)
# create logfile writer for summaries
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

with tf.Session() as session:
    session.run(init)
    for epoch in range(n_iterations):
        mini_batches = create_mini_batches(xdata, zdata.reshape(-1,1), batch_size)
        for mini_batch in mini_batches:
            x_mini, z_mini = mini_batch
            session.run(training_step, feed_dict={X:x_mini,
                z:z_mini.reshape(-1, 1)})
            if epoch % 10 == 0: # write summary
                summary = session.run(mse_summary,
                    feed_dict={X:x_mini, z:z_mini.reshape(-1,1)})
                step = epoch
```

```
                file_writer.add_summary(summary, step)

        weightvalues = weights.eval()
        print(weightvalues)

accura = accuracy()
print("Accuracy of the model : ", accura)
file_writer.close()
```

6. Showing the plot of performance over iteration and computation graph:

Figure 7: Performance over iteration graph

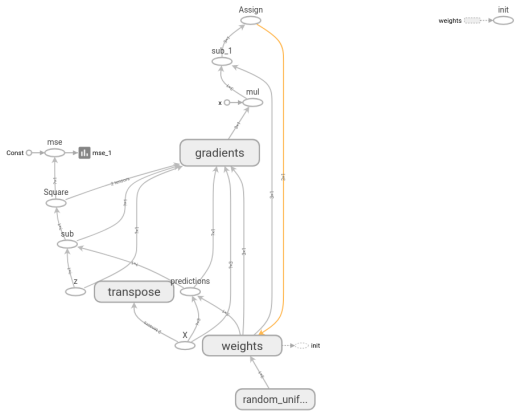

Figure 8: Computation graph

7. **Result Discussion:**

Classifier- Accuracy

Perceptron- 0.9333

Linear SVC- 0.9666

Psuedo Inverse Linear Classifier- .95

Logistic Regression- 0.955

SGDClassifier- 0.8833

Batch Gradient Descent- 0.95

Mini Batch Gradient Descent- 0.916

Mini Batch Gradient Descent (tf)- 0.966

From the accuracy, it is found that pseudo-Linear SVC and mini batch gradient descent with tensor flow classifier provide the highest accuracy. To measure these accuracies, I run each classifier at least 10 times and consider the most occurred ones.