Introduction

logistic regression is another strategy acquired by machine gaining from the field of measurements. It is the go-to strategy for order issues. Despite the name "logistic regression" this isn't a calculation for regression issues (where the assignment is to anticipate a genuine esteemed yield).

Logistic Regression is somewhat like Linear Regression as in both have the objective of evaluating the qualities for the parameters/coefficients, so the toward the finish of the preparation of the machine learning model we got a capacity that best portray the connection between the known information and the yield esteems.

Dissimilar to linear regression, the expectation for the yield is changed utilizing a non-linear capacity called the logistic capacity (a few varieties this name is: sigmoid capacity and logit work). I'm not heading off to the subtle elements of the math behind logistic regression as it's not compulsory to comprehend and have the capacity to apply this calculation, however it isn't so much that muddled and, in the event, that you know and comprehend linear regression you will think that it's simple to get this one.

Considering how the model is found out, the expectations made by logistic regression can likewise be utilized as the likelihood of a given information occasion having a place with class 0 or class 1. Logistic regression measures the connection between the straight out ward variable (target class: yes/no, spam/not spam, positive/negative,…) and at least one free factors by evaluating probabilities.

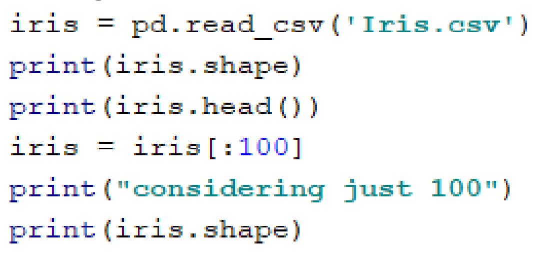
Objectives

- To implement the Logistic regression with new data set which is not used in class
- To show the graph in TensorBoard
- Changing the hyperparameter and comparing the result

Approaches/Methods

Initially, I have considered iris data for logistic regression.

1. Reading the dataset.



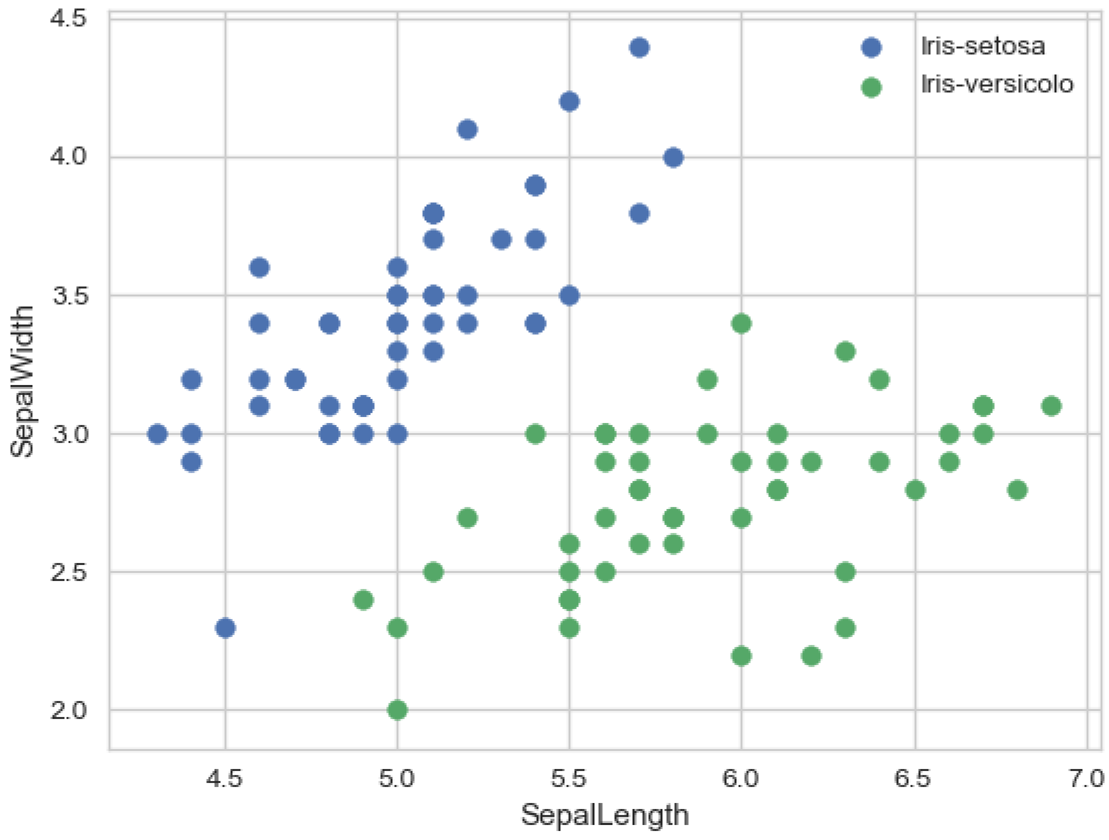
```
iris = pd.read_csv('Iris.csv')
print(iris.shape)
print(iris.head())
iris = iris[:100]
print("considering just 100")
print(iris.shape)
```

After the file is read, to do the binary classification, I’m considering the first 100 rows from the data.

Iris-setosa species is linearly distinguishable from the other two, however the other two are not linearly detachable from each other. To keep the species balance Iris-setosa and Iris-versicolor are chosen.

2. Numerical processing of the data.
   Replacing Iris-setosa as 0, Iris-versicolor as 1.

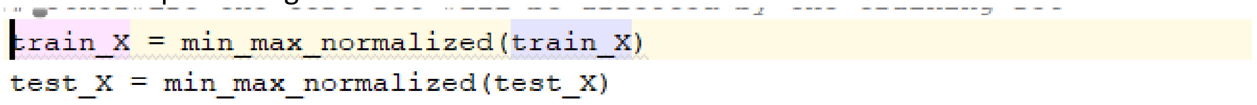
```
iris.Species = iris.Species.replace(to_replace=['Iris-setosa', 'Iris-versicolor'], value=[0, 1])
```



3. Splitting the data.
   Training set: 80% and test set: 20%

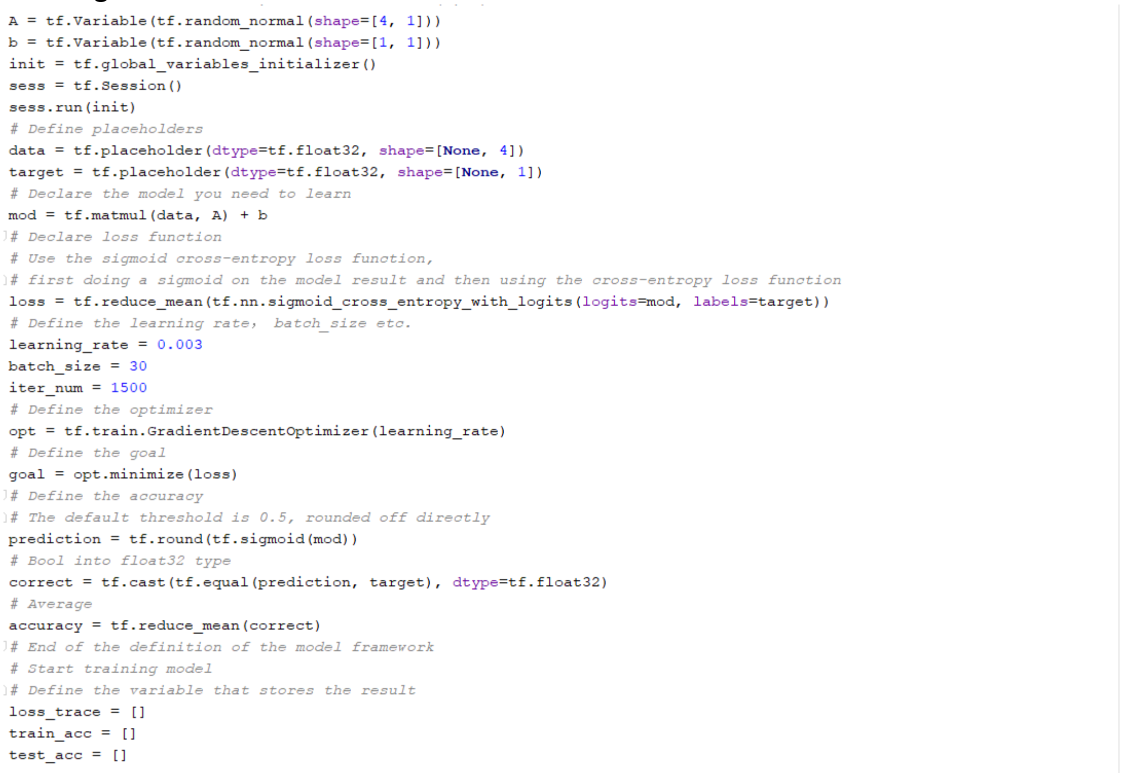
```
train_index = np.random.choice(len(X), round(len(X) * 0.8), replace=False)
# diff set
test_index = np.array(list(set(range(len(X))) - set(train_index)))
train_X = X[train_index]
train_y = y[train_index]
test_X = X[test_index]
test_y = y[test_index]
```

4. Normalized processing



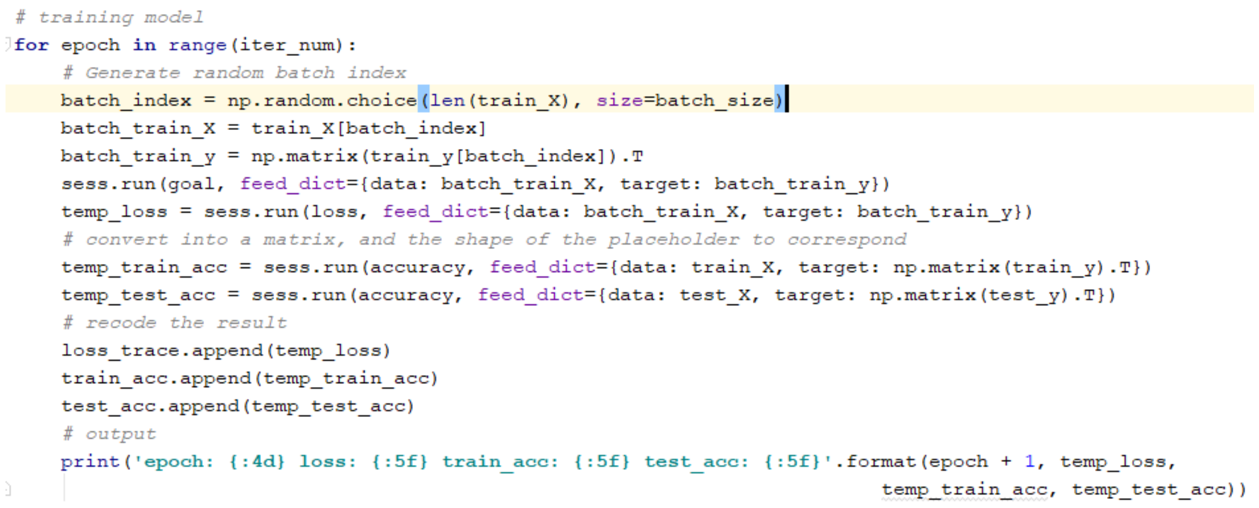
```
train_X = min_max_normalized(train_X)
test_X = min_max_normalized(test_X)
```

5. Building the model for the framework.



```
A = tf.Variable(tf.random_normal(shape=[4, 1]))
b = tf.Variable(tf.random_normal(shape=[1, 1]))
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
# Define placeholders
data = tf.placeholder(dtype=tf.float32, shape=[None, 4])
target = tf.placeholder(dtype=tf.float32, shape=[None, 1])
# Declare the model you need to learn
mod = tf.matmul(data, A) + b
# Declare loss function
# Use the sigmoid cross-entropy loss function,
# first doing a sigmoid on the model result and then using the cross-entropy loss function
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=mod, labels=target))
# Define the learning rate， batch_size etc.
learning_rate = 0.003
batch_size = 30
iter_num = 1500
# Define the optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate)
# Define the goal
goal = opt.minimize(loss)
# Define the accuracy
# The default threshold is 0.5, rounded off directly
prediction = tf.round(tf.sigmoid(mod))
# Bool into float32 type
correct = tf.cast(tf.equal(prediction, target), dtype=tf.float32)
# Average
accuracy = tf.reduce_mean(correct)
# End of the definition of the model framework
# Start training model
# Define the variable that stores the result
loss_trace = []
train_acc = []
test_acc = []
```

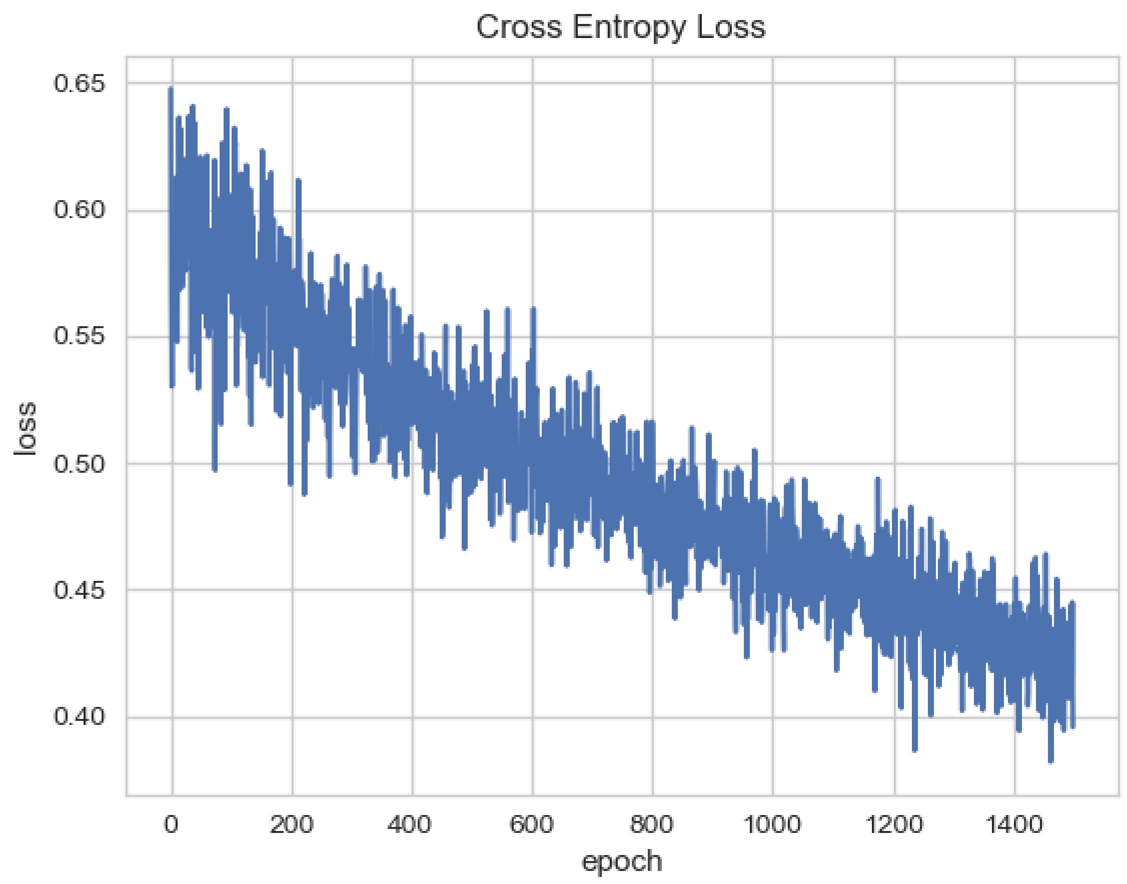
6. Training the model.



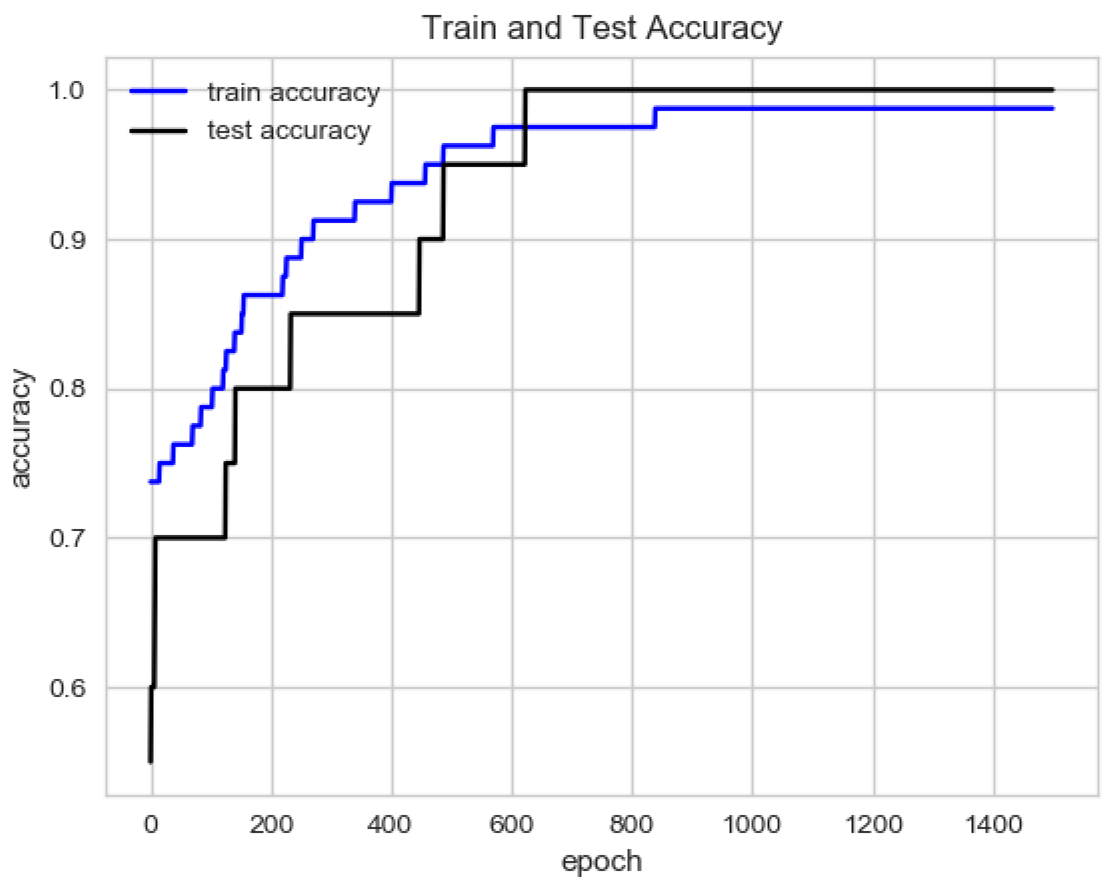
```
# training model
for epoch in range(iter_num):
    # Generate random batch index
    batch_index = np.random.choice(len(train_X), size=batch_size)
    batch_train_X = train_X[batch_index]
    batch_train_y = np.matrix(train_y[batch_index]).T
    sess.run(goal, feed_dict={data: batch_train_X, target: batch_train_y})
    temp_loss = sess.run(loss, feed_dict={data: batch_train_X, target: batch_train_y})
    # convert into a matrix, and the shape of the placeholder to correspond
    temp_train_acc = sess.run(accuracy, feed_dict={data: train_X, target: np.matrix(train_y).T})
    temp_test_acc = sess.run(accuracy, feed_dict={data: test_X, target: np.matrix(test_y).T})
    # recode the result
    loss_trace.append(temp_loss)
    train_acc.append(temp_train_acc)
    test_acc.append(temp_test_acc)
    # output
    print('epoch: {:4d} loss: {:5f} train_acc: {:5f} test_acc: {:5f}'.format(epoch + 1, temp_loss,
                                                                              temp_train_acc, temp_test_acc))
```

7. Visualization.

```
plt.plot(loss_trace)
plt.title('Cross Entropy Loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



```
# accuracy
plt.plot(train_acc, 'b-', label='train accuracy')
plt.plot(test_acc, 'k-', label='test accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Train and Test Accuracy')
plt.legend(loc='best')
plt.show()
```



Datasets

https://github.com/harshasaranam/deep-learning/blob/master/Assignment-01/documentation/Iris.csv

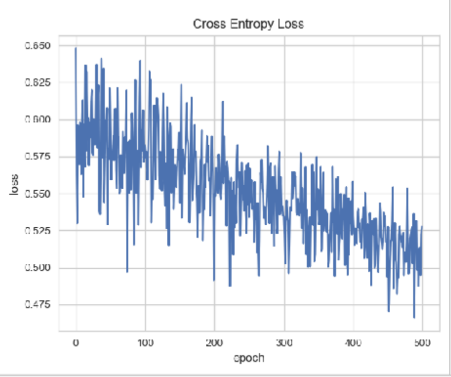
Parameters

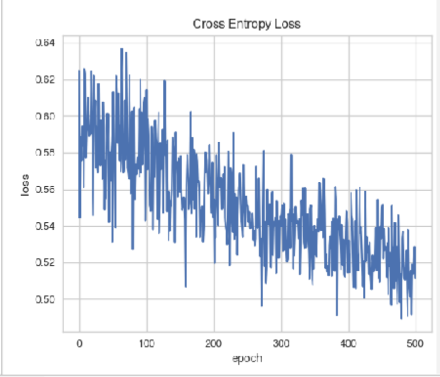
Here, I'm considering learning rate and batch size as my parameters to differentiate it. Learning rate, batch size.

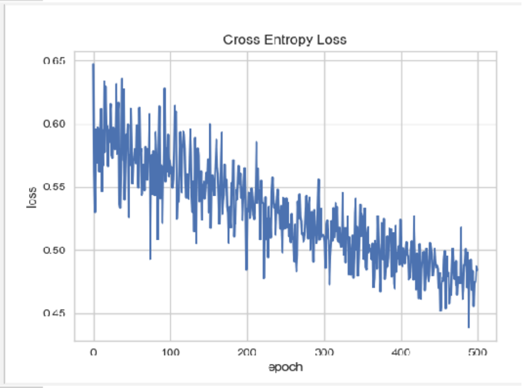
Use-cases for changing parameters

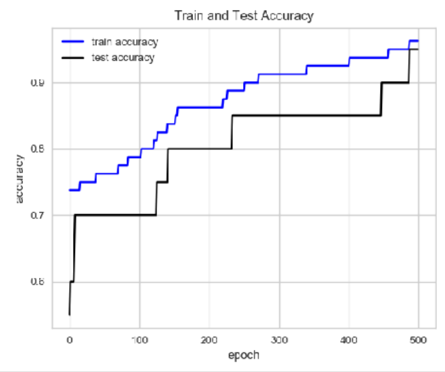
- learning_rate = 0.003
  batch_size = 30
- learning_rate = 0.003
  batch_size = 50
- learning_rate = 0.005
  batch_size = 30

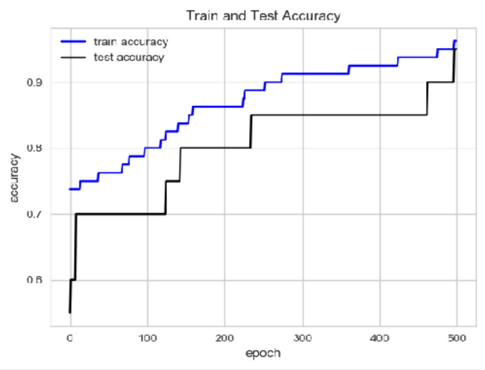
Evaluation & Discussion

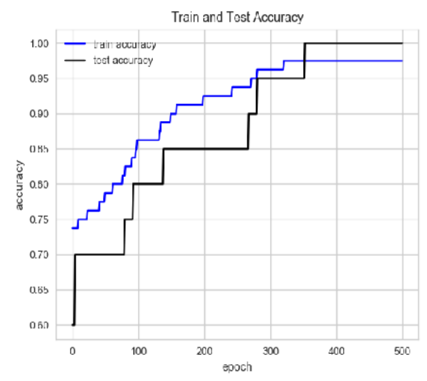












Based on the changing parameters, the above pictures shows the difference in the results.

Conclusions:

Tasks are performed by changing the parameters using the logistic regression.