SINGLE LAYER PERCEPTRON

Date: 19/1/2024

activation = sum(weight_i*x_i)+bias

predication = 1.0 if activation >= 0.0 else 0.0

w= w + learning_rate*(expected-predicated)*x

```
# make predication with weight
def predict(row, weights):
   activation= weights[0]
   for i in range (len(row)-1):
      activation += weights[i+1] * row[i]
   return 1.0 if activation >= 0.0 else 0.0
```

```
# test predictions
dataset = [[2.7810836,2.550537003,0],
[1.465489372,2.362125076,0],
[3.396561688,4.400293529,0],
[1.38807019,1.850220317,0],
[3.06407232,3.005305973,0],
[7.627531214,2.759262235,1],
[5.332441248,2.088626775,1],
[6.922596716,1.77106367,1],
[8.675418651,-0.242068655,1],
[7.673756466,3.508563011,1]]
weights = [-0.1, 0.20653640140000007, -0.23418117710000003]
for row in dataset:
   prediction = predict(row, weights)
   print("Expected=%d, Predicted=%d" % (row[-1], prediction))
```

```
weights = [-0.1, 10, -5.9]
for row in dataset:
   prediction = predict(row, weights)
   print("Expected=%d, Predicted=%d" % (row[-1], prediction))
```

```
#Estimate Perception weights using stochastic gradient desent
def train_weights(train, l_rate, n_epoch):
   weights = [0.0 for i in range(len(train[0]))]
   for epoch in range(n_epoch):
      sum_error = 0.0
      for row in train:
         prediction = predict(row, weights)
         error = row[-1] - predication
         sum_error += error**2
         weights[0] = weights[0] + l_rate*error
         for i in range(len(row)-1):
            weights[i + 1] = weights[i + 1] + l_rate * error * row[i]
```

```
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return weights

# Calculate weights
l_rate = 0.1
n_epoch = 5
weights = train_weights(dataset, l_rate, n_epoch)
print(weights)
```