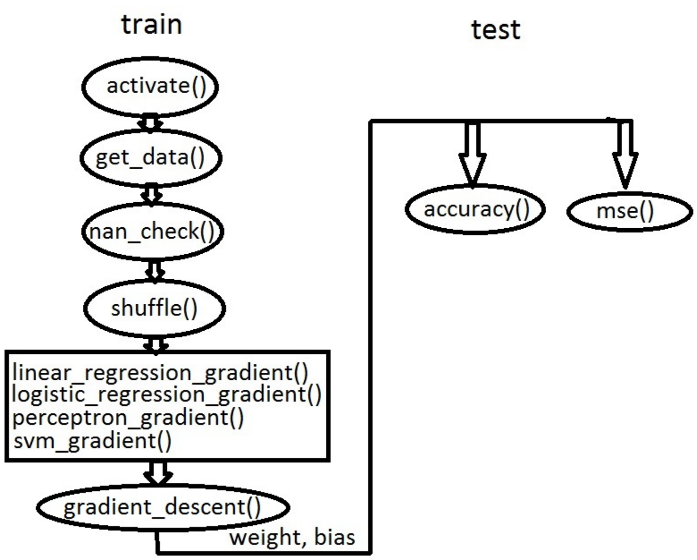
ECE544-Pattern Recognition HW1

Junze Liu

September 17, 2016

1 Pencil-and-paper



1.

$$\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial \sum_i \left((t_i - y_i)^2\right)}{\partial w_j} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - g(w' x_i + b))}{\partial w_j} \\
&= -2\sum_i (t_i - y_i) \cdot g'(w' x_i + b) \cdot x_{i,j}
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial E}{db} &= \frac{\partial \sum_i \left((t_i - y_i)^2\right)}{\partial b} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - g(w' x_i + b))}{\partial b} \\
&= -2\sum_i (t_i - y_i) \cdot g'(w' x_i + b)
\end{aligned}
$$

2.

$$
\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\sum_i \left((t_i - y_i)^2\right)}{\partial w_j} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - y_i)}{\partial w_j} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - g(w' x_i + b))}{\partial w_j} \\
&= -2\sum_i (t_i - y_i) \cdot x_{i,j}
\end{aligned}
$$

3.

$$
\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial \sum_i \left((t_i - y_i)^2\right)}{\partial w_j} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - y_i)}{\partial w_j} \\
&= 2\sum_i (t_i - y_i) \cdot \frac{\partial \sum_i (t_i - g(w' x_i + b))}{\partial w_j} \\
&= -2\sum_i (t_i - y_i) \cdot y_i \cdot (1 - y_i) \cdot x_{i,j}
\end{aligned}
$$

4.

$$
\begin{aligned}
\frac{\partial E}{\partial w_j} &= -\sum_{i:y \neq sign(\vec{w^T}(\vec{x}))} \frac{\partial((w' x_i + b) \cdot t_i)}{\partial w_j} \\
&= -\sum_{i:y \neq sign(\vec{w^T}\vec{x})} x_{i,j} \cdot t_j
\end{aligned}
$$

5.

$$
\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial \|w\|_2^2}{\partial w_j} + C \cdot \sum_i \frac{\partial [1 - t_i(w' x_i + b)]}{\partial w_j} \\
&= 2w_j - C \cdot \sum_{i:y \neq t_i} \frac{d(t_i \cdot (w' x_i + b))}{\partial w_j} \\
&= 2w_j - C \cdot \sum_{i:y \neq t_i} t_j \cdot x_{i,j}
\end{aligned}
$$

2 Code-From-Scratch

2.1 Functions

nan_check(data, label):
Find out the nan-rows in datasets and delete these rows

label_edit(label): Edit label and change the domain of it from 0, 1 to -1, 1

shuffle(data_set, label_set):
Randomly shuffle the data and label

get_data(set_type):
Get data from files and storage them in a array. Return the data_set and label_set.

linear_regression_gradient(data, label, weight, b):
Calculate the gradient of linear node classifier. Return the gradient.

logistic_regression_gradient(data, label, weight, b):
Calculate the gradient of logistic regression . Return the gradient.

perceptron_gradient(data, label, weight, b = 0):
Calculate the gradient of perceptron classifier. Return the gradient.

svm_gradient(C, data, label, w, b = 0):
Calculate the gradient of svm classifier. Return the gradient.

gradient_descent(weight, b, learning_rate, gradient_w = 0, gradient_b = 0):
Update and return weight and b.

compute_mse(data, label, w, b):
Compute the mean square error.

compute_acc(data, label, w, b):
Compute the accuracy

activate(epoch = 2500):
Activate the whole neural network and set the iteration as 2500.

2.2 Lines of codes related to the equations above

1.

$$\frac{\partial E}{\partial w_j} = -2\sum_i (t_i - y_i) \cdot g'(w'x_i + b) \cdot x_{i,j}$$

Codes:
for i in range(len(label)):
gradient_w -= (-2) * (label[i] - (np.dot(weight, data[i]) + b)) * data[i]

$$\frac{\partial E}{db} = -2\sum_i (t_i - y_i) \cdot g'(w'x_i + b)$$

Codes:
for i in range(len(label)):
gradient_b += (-2) * (label[i] - (np.dot(weight, data[i]) + b))

2.

$$\frac{\partial E}{\partial w_j} = -2\sum_i (t_i - y_i) \cdot x_{i,j}$$

Codes:
for i in range(len(label)):
gradient_w -= (-2) * (label[i] - (np.dot(weight, data[i]) + b)) * data[i]

3.

$$\frac{\partial E}{\partial w_j} = -2\sum_i (t_i - y_i) \cdot y_i \cdot (1 - y_i) \cdot x_{i,j}$$

Codes:
for i in range(len(label)):
gradient_w += (-2) * ((np.dot(weight, data[i]) + b) - label[i]) * (np.dot(weight, data[i]) + b) * (1 - (np.dot(weight, data[i]) + b)) * data[i]

4.

$$\frac{\partial E}{\partial w_j} = -\sum_{i:y \neq sign(\vec{w^T}\vec{x})} x_{i,j} \cdot t_j$$

Codes:
for i in range(len(label)):
if np.dot(weight, data[i]) * label[i] ¡ 0 :
gradient_w += (-1) * data[i] * label[i]
else:
gradient_w += 0

5.

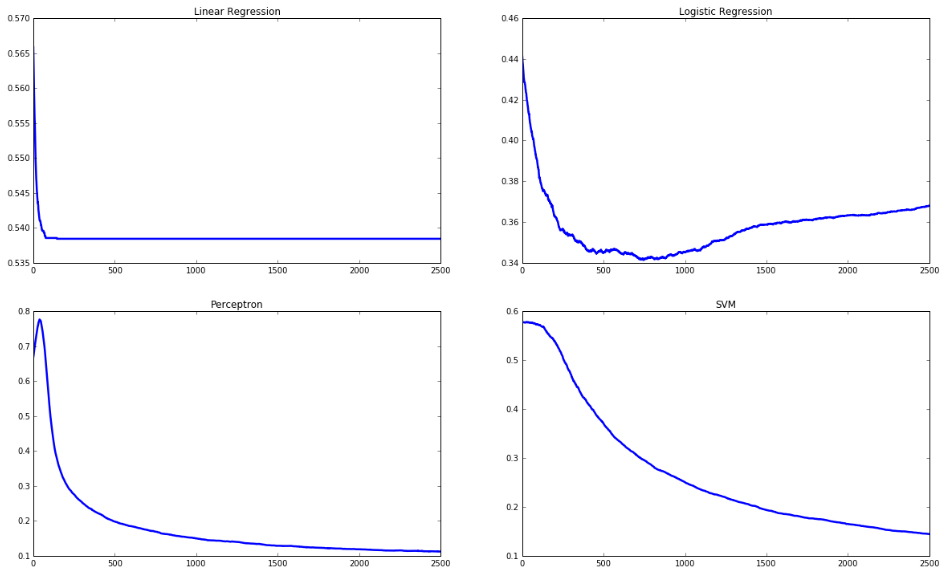
$$\frac{\partial E}{\partial w_j} = 2w_j - C \cdot \sum_{i:y \neq t_i} t_j \cdot x_{i,j}$$

Codes:
for i in range(len(label)):

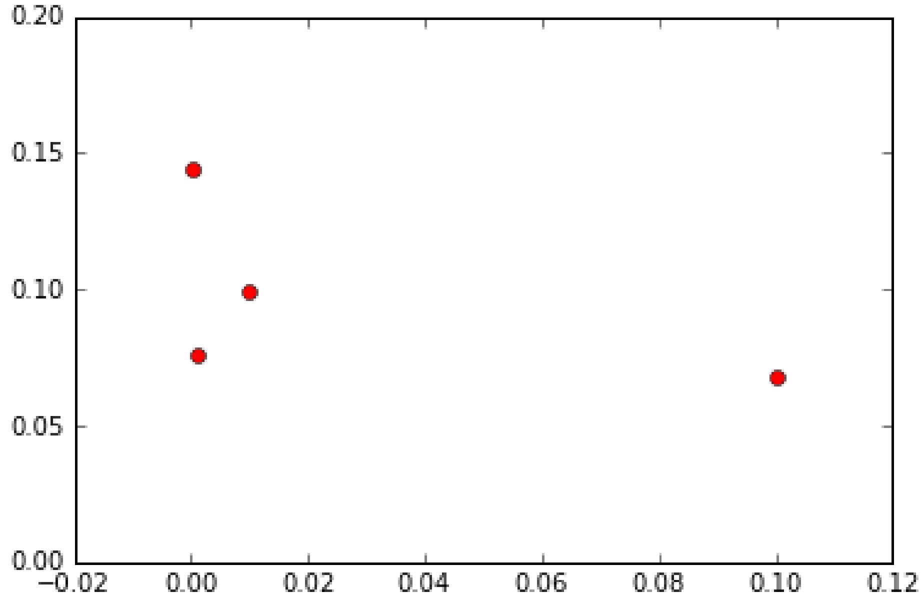
if label[i] * np.dot(w, data[i]) ¡ 1 :
gradient_w += C * (-1) * data[i] * label[i]
gradient_b += C * (-1) * label[i]
else:
gradient_w += 0
gradient_b += 0
gradient_w = (2 * w + gradient_w)

2.3 Results

2.3.1 Error Rates Figure



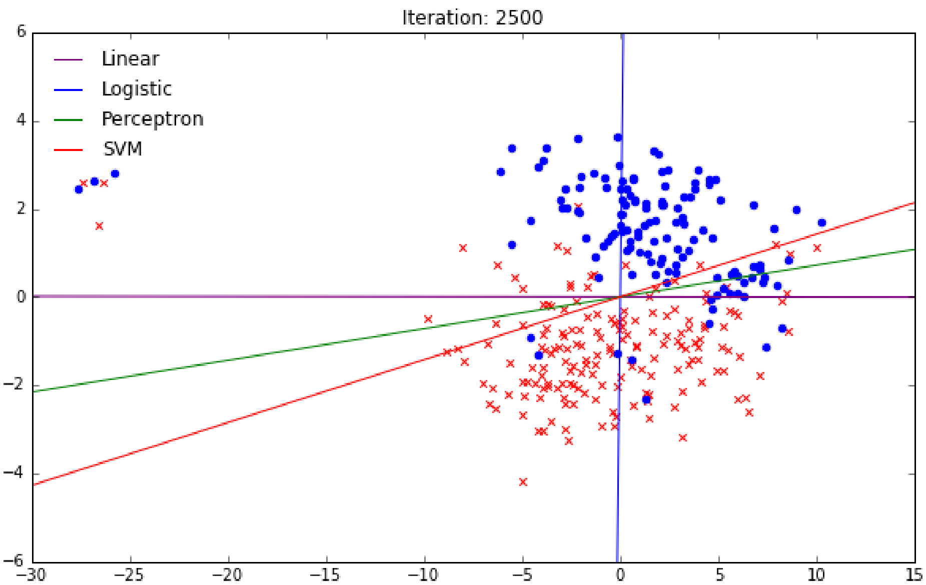
2.3.2



C = 0.1	C = 0.01	C = 0.001	C = 0.0001
error rate = 0.068156	0.099157	0.095678	0.144040

2.3.3 Scatter Plot and Different Classifier

using PCA to reduce the dimensions



3 TensorFlow

3.1 Methods

TensorFlow functions I use and explanations of them are below.

Codes:
x_placeholder = tf.placeholder(tf.float32, [None, 16])
Create a placeholder. For each sample, it has 16 dimensions' feature. When we activate the session, it will input a value.

Codes:
w = tf.Variable(tf.random_normal([16, 2]))
creates a variable. It has the shape of [16, 2] because we have 16 features in a sample and we classify it into to classes.
Codes:
y_hat = tf.nn.softmax(tf.matmul(x_placeholder, w) + b)
means we first compute the output by multiply weights and sample, then we use a softmax node to compute the class possibility.
Codes:
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_placeholder * tf.log(y_hat), reduction _indices=[1]))
calculate the cross entropy and the goal of the algorithm is to minimize it. tf.log computes the logarithm of each element, tf.reduce_mean computes the mean.
Codes:
correct_prediction = tf.equal(tf.argmax(y_hat,1), tf.argmax(y_placeholder,1))
finds the correct predictions. tf.argmax finds the index of the highest entry in y_hat and y_placeholder and compare if they are the same.
Codes:
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.cast turn the booleans in correct_prediction to floating point numbers.
Codes:
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
means we choose gradient descent to minimize croos entropy
Codes:
init = tf.initialize_all_variables()
initializes variables.
Codes:
sess = tf.Session()
launches the model in a session.
Codes:
sess.run(init)
input values into variables.

3.2 Results

Convergence Figure

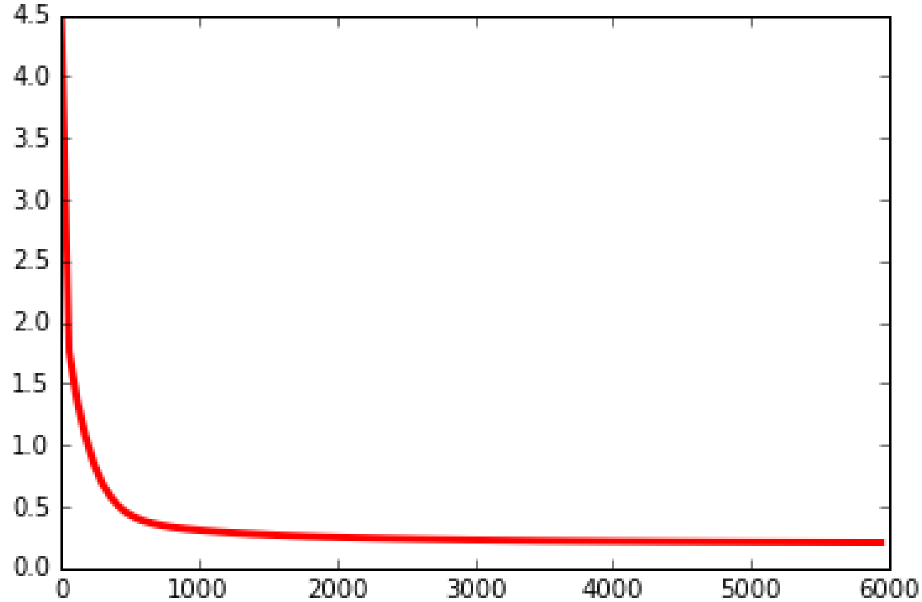


table 3.1

	train	eval
error rate	0.200659	0.131954