In [46]:
```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_svmlight_file
```

In [47]:
```
def shaping_the_data(x_training, x_testing, y_training, y_testing):

    #reshape the data to 1 and 0 for dimension of 124
    n_training = x_training.shape[0]
    n_testing = x_testing.shape[0]

    x_training = np.hstack((np.ones((n_training, 1)), x_training.toarray()))
    x_testing = np.hstack((np.ones((n_testing, 1)), x_testing.toarray()))

    y_training = y_training.reshape((n_training, 1))
    y_testing = y_testing.reshape((n_testing, 1))

    #Change the label to 0, 1
    y_training = np.where(y_training == -1, 0, 1)
    y_testing = np.where(y_testing == -1, 0, 1)

    return x_training, x_testing, y_training, y_testing

def log_likelihood(w, x, y, L2_lamda=0):

    # Cross-Entropy
    res = tf.matmul(tf.matmul(tf.transpose(w), tf.transpose(x)), y)
    - tf.reduce_sum(tf.log(1 + tf.exp(tf.matmul(x, w))))
    res += -0.5 * L2_lamda * tf.norm(w)

    return -res[0][0]

def soft_max_func(x, w):

    # Predict the probability
    y = tf.constant(np.array([0., 1.]), dtype=tf.float32)
    proba = tf.exp(tf.matmul(x, w) * y) / (1 + tf.exp(tf.matmul(x, w)))

    return proba

def score(x, y, w):

    p = soft_max_func(x, w)
    y_pred = tf.cast(tf.argmax(p, axis=1), tf.float32)
    y = tf.squeeze(y)
    acc = tf.reduce_mean(tf.cast(tf.equal(y, y_pred), tf.float32))

    return acc

def optimize(w, w_update):

    return w.assign(w - w_update)

def update(w, x, y, L2_lamda=0):
    mul = tf.sigmoid(tf.matmul(x, w))
    R_flat = mul * (1 - mul)

    dim = x.shape.as_list()[1]
    L2_reg_term = L2_lamda * tf.eye(dim)

    xRx = tf.matmul(tf.transpose(x), R_flat * x) + L2_reg_term
    S, U, V = tf.svd(xRx, full_matrices=True, compute_uv=True)
    S = tf.expand_dims(S, 1)
    S_pinv = tf.where(tf.not_equal(S, 0), 1 / S, tf.zeros_like(S))
    xRx_pinv = tf.matmul(V, S_pinv * tf.transpose(U))

    w_update = tf.matmul(xRx_pinv, tf.matmul(tf.transpose(x), mul - y) + L2_lam
da * w)

    return w_update
```

In [48]:
```
if __name__ == "__main__":
    x_training, y_training = load_svmlight_file("a9a", n_features=123, dtype=n
p.float32)
    X_test, y_test = load_svmlight_file("a9a.t", n_features=123, dtype=np.float
32)
    x_training, X_test, y_training, y_test =shaping_the_data(x_training, X_tes
t, y_training, y_test)

    L2_lamda = 30  # L2 norm parameter lamda

    N, dim = x_training.shape
    X = tf.placeholder(dtype=tf.float32, shape=(None, 124), name="x")
    y = tf.placeholder(dtype=tf.float32, shape=(None, 1), name="y")

    w = tf.Variable(0.01 * tf.ones((dim, 1), dtype=tf.float32), name="w")
    w_update = update(w, X, y, L2_lamda)
    loss = log_likelihood(w, X, y, L2_lamda)
    acc = score(X, y, w)
    optimize_op = optimize(w, w_update)

    config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=Fal
se)
    session = tf.Session(config=config)
    session.run(tf.global_variables_initializer())

    max_iter = 100
    for i in range(1, max_iter):
        print("iteration: {}".format(i))
        print("\n")

        print("log likelihood: {}".format(session.run(loss, feed_dict={X: x_tra
ining, y: y_training})))

        train_acc = session.run(acc, feed_dict={X: x_training, y: y_training})
        test_acc = session.run(acc, feed_dict={X: X_test, y: y_test})

        print("test data accuracy score: {},\ntraining data accuracy score:
{}".format(test_acc, train_acc))
        L2_norm_w = np.linalg.norm(session.run(w))
        print("L2-norm of |w|2: {}".format(L2_norm_w))
        print("\
n==========================================================================")

        deri_w = np.linalg.norm(session.run(w_update, feed_dict={X: x_training,
y: y_training}))

        if deri_w < 0.001:
            break
        w_new = session.run(optimize_op, feed_dict={X: x_training, y: y_trainin
g})

    print("End of Iteration.")
```

```
iteration: 1


log likelihood: -1169.1973876953125
test data accuracy score: 0.23622627556324005,
training data accuracy score: 0.24080955982208252
L2-norm of |w|2: 0.11135528236627579

============================================================
iteration: 2


log likelihood: -906.0789794921875
test data accuracy score: 0.8458325862884521,
training data accuracy score: 0.8442308306694031
L2-norm of |w|2: 2.0898401737213135

============================================================
iteration: 3


log likelihood: -1882.44189453125
test data accuracy score: 0.8495792746543884,
training data accuracy score: 0.8468105792999268
L2-norm of |w|2: 3.069146156311035

============================================================
iteration: 4


log likelihood: -2360.68408203125
test data accuracy score: 0.8506848216056824,
training data accuracy score: 0.847179114818573
L2-norm of |w|2: 3.6801035404205322

============================================================
iteration: 5


log likelihood: -2470.489501953125
test data accuracy score: 0.8509305119514465,
training data accuracy score: 0.8476091027259827
L2-norm of |w|2: 3.954674482345581

============================================================
iteration: 6


log likelihood: -2475.976318359375
test data accuracy score: 0.8511762022972107,
training data accuracy score: 0.8476705551147461
L2-norm of |w|2: 4.0110764503479

============================================================
iteration: 7


log likelihood: -2475.8369140625
test data accuracy score: 0.8511762022972107,
training data accuracy score: 0.8476705551147461
L2-norm of |w|2: 4.013559341430664

============================================================
End of Iteration.
```

In [ ]:

In [ ]:

In [ ]: