hw2

April 11, 2018

1 HW2

1.1 #### Multinomial Classification and Suppport Vector Machine

In [1]:
```
# import packages
import tensorflow as tf
import numpy as np
import os
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn import svm
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(font_scale=1.2)
```

In [2]:
```
# load data
folder_name = 'datasethw2'
file_name = ['data1.txt', 'data2.txt', 'data3.txt']
training_data_rate = 0.8

load_data = []

for i in range(3):
    load_data.append(pd.read_csv(os.path.join(folder_name, file_name[i]), sep='\t', head
```

In [3]:
```
amount_train = int(training_data_rate*load_data[0].shape[0])

def data_suffle_each_class(d):
    for i in range(3):
        d[i] = shuffle(d[i])
    return d

''' Functions for Question 1'''

def train_test_data(data):
    train = []
    test = []
```

```
    for i in range(3):
        train.append(data[i][:amount_train])
        test.append(data[i][amount_train:])

    train_set = np.concatenate((train[0], train[1]), axis=0)
    train_set = np.concatenate((train_set, train[2]), axis=0)
    np.random.shuffle(train_set)

    test_set = np.concatenate((test[0], test[1]), axis=0)
    test_set = np.concatenate((test_set, test[2]), axis=0)

    return train_set, test_set, test_set[:20, :], test_set[20:40, :], test_set[40:,:]

def x_data1(dataset):
    return dataset[:,:5]

''' Functions for Question 2'''

def dataset_excluding(nth_feature, data):
    train_set, test_set, _, _, _ = train_test_data(data)

    train_set = np.delete(train_set, nth_feature, axis=1)
    test_set = np.delete(test_set, nth_feature, axis=1)

    return train_set, test_set, test_set[:20, :], test_set[20:40, :], test_set[40:,:]

def x_data2(dataset):
    return dataset[:,:4]



def x_data(dataset, q_num):
    return x_data2(dataset) if q_num == 2 else x_data1(dataset)

def y_data(dataset):
    return dataset[:,-1].reshape([dataset.shape[0], 1]) - 1 # for one-hot encoding, subt
```

In [4]:
```
nb_classes = 3
learning_rate = 0.01
training_epochs = 20 #
steps = 2000

# define model

class Multinomial_Logistic_Classifier:

    def __init__(self, sess, name, question_num):
        self.sess = sess
```

```
        self.name = name
        self.question_num = question_num
        self._build_net()

    def _build_net(self):
        with tf.variable_scope(self.name):
            # input place holders
            if self.question_num == 1:
                self.X = tf.placeholder(tf.float32, [None, 5])
                self.W = tf.Variable(tf.random_normal([5, nb_classes]), name='weight')
            elif self.question_num == 2:
                self.X = tf.placeholder(tf.float32, [None, 4])
                self.W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')

            self.Y = tf.placeholder(tf.int32, [None, 1])

            self.Y_one_hot = tf.one_hot(self.Y, nb_classes) # one-hot
            self.Y_one_hot = tf.reshape(self.Y_one_hot, [-1, nb_classes])

            self.b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

        logits = tf.matmul(self.X, self.W) + self.b
        self.hypothesis = tf.nn.softmax(logits)

        cost_i = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=self.Y

        self.cost = tf.reduce_mean(cost_i)
        self.optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).

        self.prediction = tf.argmax(self.hypothesis, 1)
        self.correct_prediction = tf.equal(self.prediction, tf.argmax(self.Y_one_hot, 1)
        self.accuracy = tf.reduce_mean(tf.cast(self.correct_prediction, tf.float32))

    def predict(self, x_test):
        return self.sess.run(self.correct_prediction, feed_dict={self.X: x_test})

    def get_accuracy(self, x_test, y_test):
        return self.sess.run(self.accuracy, feed_dict={self.X: x_test, self.Y: y_test})

    def train(self, x_data, y_data):
        return self.sess.run([self.cost, self.optimizer, self.W, self.b], feed_dict={sel

    def values_Wb(self):
        W_val, b_val = self.sess.run([self.W, self.b])
        return W_val, b_val
```

In [5]:
```
def training(q_num, nth_feature=None):
    avg_accuracy = 0
```

```
    avg_acc_class1 = 0
    avg_acc_class2 = 0
    avg_acc_class3 = 0
    max_accuracy = 0

    for i in range(10):
        # dataset
        if q_num == 1:
            train_set, test_set, test_set_1, test_set_2, test_set_3 = train_test_data(da
        elif q_num == 2:
            train_set, test_set, test_set_1, test_set_2, test_set_3 = dataset_excluding(

        # initialize
        sess = tf.Session()
        m1 = Multinomial_Logistic_Classifier(sess, "m1", q_num)

        sess.run(tf.global_variables_initializer())

        #print('Learning Started!')

        # train my model
        for epoch in range(training_epochs):
            avg_cost = 0

            for j in range(steps):
                c, _, W_val, b_val = m1.train(x_data(train_set, q_num), y_data(train_set
                avg_cost += c / steps

            #print('Epoch:', '%02d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

        #print('Learning Finished!')

        acc = m1.get_accuracy(x_data(test_set, q_num), y_data(test_set))
        acc_class1 = m1.get_accuracy(x_data(test_set_1, q_num), y_data(test_set_1))
        acc_class2 = m1.get_accuracy(x_data(test_set_2, q_num), y_data(test_set_2))
        acc_class3 = m1.get_accuracy(x_data(test_set_3, q_num), y_data(test_set_3))

        avg_accuracy += acc / 10
        avg_acc_class1 += acc_class1 / 10
        avg_acc_class2 += acc_class2 / 10
        avg_acc_class3 += acc_class3 / 10

        if max_accuracy < acc :
            max_accuracy = acc
            max_W = W_val
            max_b = b_val

    return avg_accuracy, max_accuracy, avg_acc_class1, avg_acc_class2, avg_acc_class3, m
```

1.2 Q1. Multinomial Classification

In [6]:
```
avg_accuracy, max_accuracy, avg_acc_class1, avg_acc_class2, avg_acc_class3, W, b = train

print('average accuracy:', '{:.9f}'.format(avg_accuracy))
print('average accuracy for class 1:', '{:.9f}'.format(avg_acc_class1))
print('average accuracy for class 2:', '{:.9f}'.format(avg_acc_class2))
print('average accuracy for class 3:', '{:.9f}'.format(avg_acc_class3))
print('\nthe highest accuracy:', '{:.9f}'.format(max_accuracy))
print('weight:\n', W)
print('bias:\n', b)
```

```
average accuracy: 0.800000000
average accuracy for class 1: 0.864999992
average accuracy for class 2: 0.830000001

average accuracy for class 3: 0.705000001
nthe highest accuracy: 0.883333325
weight:
 [[ 1.0360097   1.3999194   0.89743996]
 [-0.13516912  0.17269608  0.10517832]
 [-1.2849635   1.0764158  -0.78427213]
 [-1.1136543   1.6155427   0.7139656 ]
 [ 0.5250165   1.1507765   1.2004191 ]]
bias:
 [ 5.2689934 -6.844008   1.342023 ]
```

1.3 Q2. Multinomial Classification & Feature Reduction

In [7]:
```
for k in range(5):
    print("\n   < A model excluding the feature at column", k, ">")

    avg_accuracy, max_accuracy, avg_acc_class1, avg_acc_class2, avg_acc_class3, W, b = t

    print('average accuracy:', '{:.9f}'.format(avg_accuracy))
    print('average accuracy for class 1:', '{:.9f}'.format(avg_acc_class1))
    print('average accuracy for class 2:', '{:.9f}'.format(avg_acc_class2))
    print('average accuracy for class 3:', '{:.9f}'.format(avg_acc_class3))
    print('the highest accuracy:', '{:.9f}'.format(max_accuracy))
    print('weight:\n', W)
    print('bias:\n', b)
```

```

   < A model excluding the feature at column 0 >
average accuracy: 0.755000013
average accuracy for class 1: 0.850000006
average accuracy for class 2: 0.764999998
average accuracy for class 3: 0.650000000
```

```
the highest accuracy: 0.800000012
weight:
 [[-0.71209604 -0.15263793 -0.45918292]
 [-1.3664936   0.2814427  -0.4648291 ]
 [-0.4416324   1.2639712   1.0551808 ]
 [-1.3364556  -0.30226645 -0.83470106]]
bias:
 [ 4.171041   -4.3506637  -0.25914234]

   < A model excluding the feature at column 1 >
average accuracy: 0.828333330
average accuracy for class 1: 0.869999993
average accuracy for class 2: 0.834999996
average accuracy for class 3: 0.780000007
the highest accuracy: 0.883333325
weight:
 [[-0.10264365  0.42055514 -0.14231323]
 [-0.5148925   2.3349345   0.50254554]
 [-1.1095403   0.5762804   0.14800042]
 [ 0.8398855   1.3390179   1.2579081 ]]
bias:
 [ 5.493771  -5.4288354  2.6585095]

   < A model excluding the feature at column 2 >
average accuracy: 0.734999996
average accuracy for class 1: 0.865000010
average accuracy for class 2: 0.684999996
average accuracy for class 3: 0.655000007
the highest accuracy: 0.783333361
weight:
 [[-0.12909222  0.14141017 -0.27851254]
 [-0.501746   -0.06998561 -0.27936128]
 [-0.9034062   0.47092497  0.34635237]
 [-1.0350657   0.06747928 -0.43283552]]
bias:
 [ 4.330229  -5.913997   1.5248063]

   < A model excluding the feature at column 3 >
average accuracy: 0.741666669
average accuracy for class 1: 0.854999995
average accuracy for class 2: 0.805000007
average accuracy for class 3: 0.565000004
the highest accuracy: 0.816666663
weight:
 [[ 0.40602112  0.79171324  0.30961126]
 [-0.7945556  -0.53408027 -0.5980883 ]
 [-0.82560885  1.1866767  -0.27512613]
 [-1.7005206  -0.00691396 -0.39334705]]
```

```
bias:
 [ 4.8407373 -6.1396646  1.4997874]

   < A model excluding the feature at column 4 >
average accuracy: 0.814999992
average accuracy for class 1: 0.894999999
average accuracy for class 2: 0.825000000
average accuracy for class 3: 0.725000006
the highest accuracy: 0.866666675
weight:
 [[-0.8201985  -0.28106672 -0.8375888 ]
 [ 0.03375107  0.13611214  0.12962776]
 [-1.1861694   1.5676681  -0.09783702]
 [-1.3683645   1.0503572   0.42012948]]
bias:
 [ 4.1869187 -6.751715   1.3469859]
```

According to the result, the accuracy of the models excluding feature 1 and feature 4 is higher then Q1's result.
The accuracy of the model except feature 4 has improved by 1.8405% compared to Q1.
The accuracy of the model except feature 1 has improved by 3.4205% compared to Q1.

More features would improve the performance of Machine Learning (ML). However, noisy data has adverse influences on the ML models. Therefore, we should carefully refine data to make the model work well.

1.4 Q3. Support Vector Machine

In [8]:
```
avg_acc = avg_acc_1 = avg_acc_2 = avg_acc_3 = 0

for i in range(10):
    # get shuffled data
    train_set, test_set, test_set_1, test_set_2, test_set_3 = train_test_data(data_suffl

    # fit the SVM model
    svm_model = svm.SVC(decision_function_shape='ovr')
    svm_model.fit(x_data(train_set, 3), y_data(train_set).ravel())

    # mean accuracy
    accu = svm_model.score(x_data(test_set, 3), y_data(test_set).ravel())
    accu_class1 = svm_model.score(x_data(test_set_1, 3), y_data(test_set_1).ravel())
    accu_class2 = svm_model.score(x_data(test_set_2, 3), y_data(test_set_2).ravel())
    accu_class3 = svm_model.score(x_data(test_set_3, 3), y_data(test_set_3).ravel())

    avg_acc += accu / 10.0
    avg_acc_1 += accu_class1 / 10.0
    avg_acc_2 += accu_class2 / 10.0
    avg_acc_3 += accu_class3 / 10.0
```

```
        print('average test accuracy: ', avg_acc, \
              '\naccuracy for class 1:', avg_acc_1, \
              '\naccuracy for class 2:', avg_acc_2, \
              '\naccuracy for class 3:', avg_acc_3)
```

```
average test accuracy:  0.8016666666666666
accuracy for class 1: 0.9099999999999999
accuracy for class 2: 0.7749999999999999
accuracy for class 3: 0.7200000000000001
```