1.

(a)

This problem is solved by scikit-learn and python

(b)

Mean of the unnormalized training data:

[12.965, 2.27 , 2.376, 19.649, 98.91 , 2.272, 2.029, 0.361, 1.576, 5.091, 0.953, 2.56 , 729.708]

Standard deviation of the unnormalized training data:

[0.82 , 1.103, 0.273, 3.465, 11.559, 0.615, 0.92 , 0.12 , 0.541, 2.404, 0.23 , 0.723, 307.221]

The normalizing factors should be calculated from the training data only, because when we training a model to do some classification or regression, the only thing to know is the training data, we do not have any information of test data, so we could not obtain the mean or standard deviation of the test data.

(c)

(i) From the source code of sklearn.linear_model.Perceptron.fit(), the default initial weight vector is a zero vector.

(ii) There is a parameter in sklearn.linear_model.Perceptron() called tol, if tol=None, the halting condition is when all training data points are correctly classified, if tol is a certain number, the halting condition is when previous loss – current loss is smaller than tol.

(d)

When using the first 2 features:

W1 = [2.398, -2.103]

W2 = [-3.654, -0.825]

W3 = [1.464, -0.399]

Classification accuracy on training set is 79.78%

Classification accuracy on test set is 77.53%

When using all features:

W1 = [4.145, 1.795, 2.46 , -3.219, 1.172, -1.087, 3.737, -0.138, 0.389, -3.034, 2.81 , 2.615, 4.982]

W2 = [-4.953, -3.073, -1.419, 3.116, -2.697, 1.098, 0.227, 1.092, -0.66 , -5.364, 2.838, -0.137, -3.862]

W3 = [0.314, 1.886, 0.408, 1.226, 3.249, -0.892, -4.015, -1.219, -0.46 , 4.781, -3.801, -2.154, -0.08]

Classification accuracy on training set is 100.0%

Classification accuracy on test set is 93.26%

(e)

When using the first 2 features:

W1 = [1.883, -0.191],

W2 = [-3.169, -0.331],

W3 = [0.608, 1.21]

Classification accuracy on training set with two features is 83.14%

Classification accuracy on test set with two features is 79.78%

When using all features:

W1 = [3.506, -1.09 , 1.32 , -1.527, 0.526, -0.525, 2.907, -0.787, -1.414, -0.116, 1.171, 1.94 , 4.562],

W2 = [-5.839, -5.139, -1.669, 1.916, -1.599, 0.711, 0.39 , 2.837, 1.366, -7.251, 2.406, -2.413, -3.524],

W3 = [0.092, 0.36 , 1.894, 0.946, 3.068, -0.123, -2.827, -2.081, -3.644, 5.519, -3.48 , -3.037, -1.132]

Classification accuracy on training set with all features is 100.00%

Classification accuracy on test set with all features is 96.63%

(f)

Both in (d) and (e), using all of the 13 features could obtain a higher classification accuracy, because more feature could provide more information to do classification. As for 2 features in (d) and (e), the randomized initialization is better than the default zero initialization both on training set and test set, and when using 13 features, the performance of randomized initialization is better than zero initialization on test set, and the same on training set.

(g)

When using the first 2 features of unnormalized data, the classification accuracy on the test data is 75.28%.

When using all features of unnormalized data, the classification accuracy on the test data is 97.75%

(h)

When using the first 2 features of normalized data, the classification accuracy on the test data is 75.28%.

When using all features of normalized data, the classification accuracy on the test data is 97.75%.

(i)

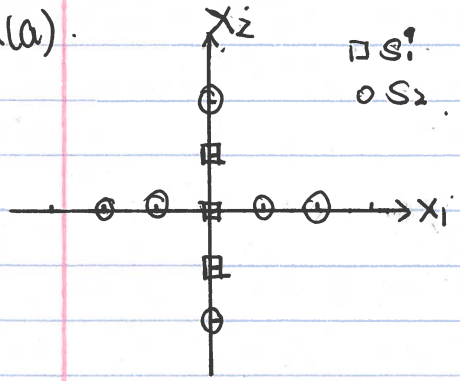
The test accuracy of (g) and (h) are identical.

(j)

The test accuracy of (h) and (e) are similar, the test accuracy when using 2 features is lower when using MSE than when using perceptron, and the test accuracy when using all features is higher when using MSE than when using perceptron, but in both cases the difference between them are relatively small.

Zheng Wen | 7112807212

2.(a).



S_1 and S_2 are not linearly separable.

(b) expanded feature space is consist of

$\phi(x) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)^T$.

S_1: $(1,0,0,0,0,0)^T$

$(1,0,1,0,0,1)^T$

$(1,0,-1,0,0,1)^T$.

S_2: $(1,-2,0,4,0,0)^T$.

$(1,-1,0,1,0,0)^T$

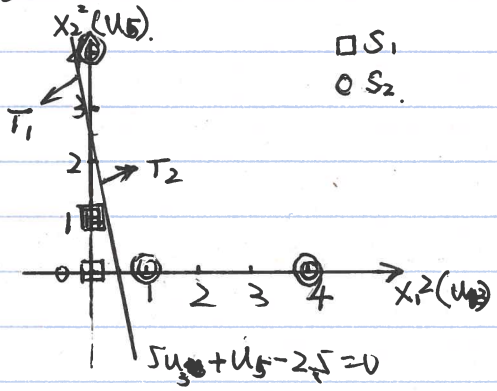
$(1,0,2,0,0,4)^T$

$(1,0,-2,0,0,4)^T$

$(1,1,0,1,0,0)^T$

$(1,2,0,4,0,0)^T$.

(c).

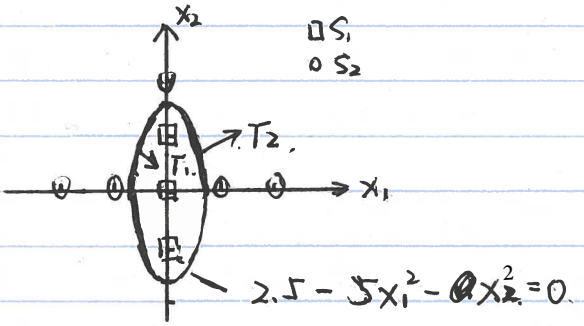


$w' = (2.5, 0, 0, -\sqrt{5}, 0, -1)^T$

$w'\phi(x) \geq 0 \Rightarrow$ class 1. i.e. $2.5 - 5u_3 - u_5 > 0 \Rightarrow$ class 1.

$w'\phi(x) < 0 \Rightarrow$ class 2. $2.5 - 5u_3 - u_5 < 0 \Rightarrow$ class 2

(d).



decision rule:

$2.5 - 5x_1^2 - x_2^2 > 0 \Rightarrow$ class 1.

$2.5 - 5x_1^2 - x_2^2 < 0 \Rightarrow$ class 2.

Problem1.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  1 21:45:40 2020

@author: Lenovo
"""

import numpy as np
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LinearRegression
from sklearn.multiclass import OneVsRestClassifier
from collections import Counter

class MSE_binary(LinearRegression):
    def __init__(self):
        # print('Calling newly created MSE binary function...')
        super(MSE_binary, self).__init__()
    def predict(self, X):
        y = self._decision_function(X)
        y_round = np.round(y)
        y_count = Counter(y_round)
        top_2 = y_count.most_common(2)
        label1 = np.max([top_2[0][0], top_2[1][0]])
        label2 = np.min([top_2[0][0], top_2[1][0]])
        thr = (label1 - label2) / 2 + label2
        y_binary = y
        y_binary[np.where(y_binary >= thr)] = label1
        y_binary[np.where(y_binary < thr)] = label2
```

```
    return y_binary


np.set_printoptions(precision=3, suppress=True)

train_set = np.loadtxt("D:\\EE559\\HW_6\\wine_train.csv",delimiter=",")
train_data = train_set[:, 0:-1]
train_label = train_set[:,-1]

test_set = np.loadtxt("D:\\EE559\\HW_6\\wine_test.csv",delimiter=",")
test_data = test_set[:, 0:-1]
test_label = test_set[:,-1]


## b
train_mean = np.mean(train_data, axis=0)
train_std = np.std(train_data, axis=0)
print('train mean', repr(train_mean))
print('train std', repr(train_std))


nor_train_data = (train_data - train_mean) / train_std
nor_test_data = (test_data - train_mean) / train_std

## d - 1
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(nor_train_data[:, :2], train_label)
train_accuracy = np.sum((train_label == clf.predict(nor_train_data[:, :2])).astype(float)) /
np.shape(train_label)[0]
```

```
test_accuracy = np.sum((test_label == clf.predict(nor_test_data[:, :2])).astype(float)) /
np.shape(test_label)[0]

print('Weight with two Feature', repr(clf.coef_))

print('Classification accuracy on training set with two feature', train_accuracy)

print('Classification accuracy on test set with two feature', test_accuracy)


## d - 2

clf.fit(nor_train_data, train_label)

train_accuracy = np.sum((train_label == clf.predict(nor_train_data)).astype(float)) /
np.shape(train_label)[0]

test_accuracy = np.sum((test_label == clf.predict(nor_test_data)).astype(float)) / np.shape(test_label)[0]

print('Weight with all feature', repr(clf.coef_))

print('Classification accuracy on training set with all feature', train_accuracy)

print('Classification accuracy on test set with all feature', test_accuracy)


clf.fit(nor_train_data[:, :2], train_label)

np.random.seed(200)

## e - 1

max_train = 0

for i in range(100):

    co_in = np.random.rand(3, 2)

    # clf.fit(nor_train_data[:, :2], train_label, coef_init=co_in)

    train_accuracy = np.sum((train_label == clf.predict(nor_train_data[:, :2])).astype(float)) /
np.shape(train_label)[0]

    test_accuracy = np.sum((test_label == clf.predict(nor_test_data[:, :2])).astype(float)) /
np.shape(test_label)[0]

    if train_accuracy >= max_train:

        max_train = train_accuracy

        max_weight = clf.coef_

        max_test = test_accuracy
```

```
print('Weight with two Feature', repr(max_weight))

print('Classification accuracy on training set with two feature', max_train)

print('Classification accuracy on test set with two feature', max_test)


np.random.seed(100)

## e- 2

max_train = 0

clf.fit(nor_train_data, train_label)

for i in range(100):

    co_in = np.random.rand(3, 13)

    clf.fit(nor_train_data, train_label, coef_init=co_in)

    train_accuracy = np.sum((train_label == clf.predict(nor_train_data)).astype(float)) /
np.shape(train_label)[0]

    test_accuracy = np.sum((test_label == clf.predict(nor_test_data)).astype(float)) /
np.shape(test_label)[0]

    if train_accuracy > max_train:

        max_train = train_accuracy

        max_weight = clf.coef_

        max_test = test_accuracy

print('Weight with all Feature', repr(max_weight))

print('Classification accuracy on training set with all feature', max_train)

print('Classification accuracy on test set with all feature', max_test)

print()

## g

train_set_1 = train_set[np.where(train_label == 1)]

train_set_2 = train_set[np.where(train_label == 2)]

train_set_12 = np.concatenate((train_set_1, train_set_2), axis=0)

train_data_12 = train_set_12[:, :-1]

train_label_12 = train_set_12[:, -1]
```

```
test_set_1 = test_set[np.where(test_label == 1)]
test_set_2 = test_set[np.where(test_label == 2)]
test_set_12 = np.concatenate((test_set_1, test_set_2), axis=0)
test_data_12 = test_set_12[:, :-1]
test_label_12 = test_set_12[:, -1]

binary_model = MSE_binary()
mc_model = OneVsRestClassifier(binary_model)

mc_model.fit(train_data, train_label)
pred_MSE_train = mc_model.predict(train_data)
acc_MSE_train = np.sum((pred_MSE_train == train_label).astype(float)) / np.shape(train_label)[0]
print('Classification accuracy on training set with all features with MSE, unnormalized is', acc_MSE_train)
pred_MSE_test = mc_model.predict(test_data)
acc_MSE_test = np.sum((pred_MSE_test == test_label).astype(float)) / np.shape(test_label)[0]
print('Classification accuracy on test set with all features with MSE, unnormalized is', acc_MSE_test)
print()
mc_model.fit(train_data[:, :2], train_label)
pred_MSE_train = mc_model.predict(train_data[:, :2])
acc_MSE_train = np.sum((pred_MSE_train == train_label).astype(float)) / np.shape(train_label)[0]
print('Classification accuracy on training set with two features with MSE, unnormalized is',
acc_MSE_train)
pred_MSE_test = mc_model.predict(test_data[:, :2])
acc_MSE_test = np.sum((pred_MSE_test == test_label).astype(float)) / np.shape(test_label)[0]
print('Classification accuracy on test set with two features with MSE, unnormalized is', acc_MSE_test)
print()
mc_model.fit(nor_train_data, train_label)
pred_MSE_train = mc_model.predict(nor_train_data)
```

```
acc_MSE_train = np.sum((pred_MSE_train == train_label).astype(float)) / np.shape(train_label)[0]
print('Classification accuracy on training set with all features with MSE, normalized is', acc_MSE_train)
pred_MSE_test = mc_model.predict(nor_test_data)
acc_MSE_test = np.sum((pred_MSE_test == test_label).astype(float)) / np.shape(test_label)[0]
print('Classification accuracy on test set with all features with MSE, normalized is', acc_MSE_test)
print()
mc_model.fit(nor_train_data[:, :2], train_label)
pred_MSE_train = mc_model.predict(nor_train_data[:, :2])
acc_MSE_train = np.sum((pred_MSE_train == train_label).astype(float)) / np.shape(train_label)[0]
print('Classification accuracy on training set with two features with MSE, normalized is', acc_MSE_train)
pred_MSE_test = mc_model.predict(nor_test_data[:, :2])
acc_MSE_test = np.sum((pred_MSE_test == test_label).astype(float)) / np.shape(test_label)[0]
print('Classification accuracy on test set with two features with MSE, normalized is', acc_MSE_test)
```