CSC336H, Fall 2019

Assignment 1

Sangwoo Kim - 1004078384

Oct 5th, 2019

Question 1

Nearest Neighbours and the Curse of Dimensionality

(a)

$$\begin{aligned} E[Z] &= E[(X-Y)^2] \\ &= E[X^2 - 2XY + Y^2] \\ &= E[X^2] - E[2XY] + E[Y^2] \\ &= E[x^2] - 2E[X]E[Y] + E[Y^2] \end{aligned}$$

We have to calculate $E[X], E[Y], E[X^2]$ and $E[Y^2]$.

Since X and Y sampled from the same probability distribution and are mutually independent, they are independent and identically distribute. So, we know that $E[X] = E[Y]$ and $E[X^2] = E[Y^2]$.

Thus we have to get the value of $E[X]$ and $E[X^2]$. Since these are continuous random variable and uniformly distributed. So $p(x) = \frac{1}{1-0} = 1$. Then,

$$\begin{aligned} E[X] &= \int_0^1 Xp(X)dx = \int_0^1 X dx \\ &= \left[\frac{1}{2}X^2\right]_0^1 = \frac{1}{2} \\ E[X^2] &= \int_0^1 X^2 p(X)dx = \int_0^1 X^2 dx \\ &= \left[\frac{1}{3}X^3\right]_0^1 = \frac{1}{3} \end{aligned}$$

So, the result will be

$$E[Z] = \frac{1}{3} - 2 \cdot (\frac{1}{2})^2 + \frac{1}{3}$$
$$= \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$$

Now $Var[x]$.

$$Var[X] = E[Z^2] - E[Z]^2$$
$$= E[(X^2 - 2XY + Y^2)^2] - \left(\frac{1}{6}\right)^2$$
$$= E[(X^4 - 4X^3Y + 6X^2Y^2 - 4XY^3 + Y^4] - \frac{1}{36}$$
$$= E[X^2] - 4E[X^3]E[Y] + 6E[X^2][Y^2] - 4E[X]E[Y^3] + E[Y^4] - \frac{1}{36}$$
$$= 2E[X^4] - 8E[X^3]E[X] + 6E[X^2]^2 - \frac{1}{36}$$

By simliar calculation with $E[X]$ and $E[X^2]$, we know that $E[X^3] = \frac{1}{4}$ and $E[x^4] = \frac{1}{5}$

$$= 2 \times \frac{1}{5} - 8 \times \frac{1}{4} \times \frac{1}{2} + 6 \times \frac{1}{9} - \frac{1}{36}$$
$$= \frac{1}{15} - \frac{1}{36}$$
$$= \frac{7}{180}$$

(b)

$$R = Z_1 + \cdots + Z_d$$
$$E[R] = E[Z_1 + \cdots + Z_d]$$
$$= E[Z_1] + \cdots + E[Z_d]$$
$$= \frac{1}{6} \times d = \frac{d}{6}$$
$$Var[R] = Var[Z_1 + \cdots + Z_d]$$
$$= Var[Z_1] + \ldots Var[Z_d]$$

since all random variables are independent, Var[X+Y] = Var[X] + Var[Y]

$$= \frac{7}{180} \times d = \frac{7d}{180}$$

(c) The maximum Euclidean distance will be

$$\sqrt{(1_1)^2 + (1_2)^2 + \cdots + (1_d)^2} = \sqrt{d}$$

Thus, squared Euclidean distance is d.

We know that $Var[R] = \frac{7d}{180}$. So, standard deviation is $\sqrt{\frac{7d}{180}}$.

And $E[R] = \frac{d}{6}$. Standard deviation has square root growth rate while expected value has linear growth rate. In other words, expected value grows faster than standard deviation as d becomes larger. Therefore, most points are far away. And approximately the same distance, because standard deviation is relatively smaller, so it does not spread much.

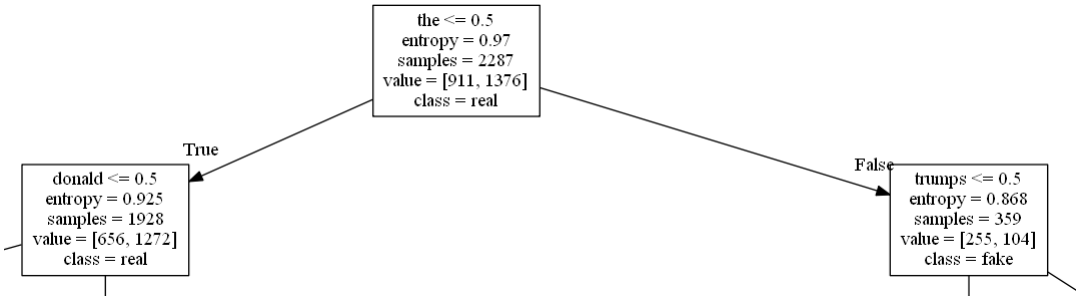
Question 2

(a) Code is included below.

(b) These results are results from the function selectmodel().

```
In[3]: dataset: object = load_data("C:/Users/exle0/Documents/2019 Fall/CSC311/Project")
In[4]: select_model(dataset)
The accuracy we get by using info gain with max depth of 5 = 0.735593220338983
The accuracy we get by using info gain with max depth of 10 = 0.7491525423728813
The accuracy we get by using info gain with max depth of 25 = 0.7932203389830509
The accuracy we get by using info gain with max depth of 50 = 0.8101694915254237
The accuracy we get by using info gain with max depth of 75 = 0.7898305084745763
The accuracy we get by using gini with max depth of 5 = 0.7389830508474576
The accuracy we get by using gini with max depth of 10 = 0.7491525423728813
The accuracy we get by using gini with max depth of 25 = 0.7966101694915254
The accuracy we get by using gini with max depth of 50 = 0.7898305084745763
The accuracy we get by using gini with max depth of 75 = 0.7830508474576271
```

(c) This is a visualization of the first two layer of the decision tree. Decision tree classifier that has entropy criteria and max depth 50 is used.



(d) This is the output of the function compute information gain().

```
In[7]: compute_information_gain(dataset, "the")
Out[7]: 0.053818996397352104
In[8]: compute_information_gain(dataset, "donald")
Out[8]: 0.051106172368596825
In[9]: compute_information_gain(dataset, "trump")
Out[9]: 0.03799508982830635
In[10]: compute_information_gain(dataset, "hillary")
Out[10]: 0.03135743831707605
In[11]: compute_information_gain(dataset, "america")
Out[11]: 0.009396006545869917
```

code is on next page.

```python
"""
CSC311 Assignment 1 Question 1 Sangwoo Kim
"""
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import pydot
import math


def load_data(file_path):
    """

    :param file_path: path that where files locate
    :type file_path: string
    :return: data set that includesx_train, x_valid, x_test, y_train, y_valid, y_test sets and
        features
    :rtype: dictionary[string:matrix]
    """
    data = []
    size = 0

    files = [file_path + "/clean_fake.txt", file_path + "/clean_real.txt"]

    with open(files[0]) as datafile:
        fake_data = datafile.read().split("\n")
        data += fake_data
        num_fake = len(fake_data)
        size += len(fake_data)

    with open(files[1]) as datafile:
        real_data = datafile.read().split("\n")
        data += real_data
        num_real = len(real_data)
        size += len(real_data)  # print(txts)

    label = np.zeros((num_fake, 1))
    label = np.vstack((label, np.ones((num_real, 1))))

    vector = CountVectorizer()
    datan = vector.fit_transform(data)
    datan = datan.toarray()
    features = vector.get_feature_names()

    x_train, x_test, y_train, y_test = train_test_split(datan, label,
                                                        test_size=0.3)
    x_test, x_valid, y_test, y_valid = train_test_split(x_test, y_test,
                                                        test_size=0.3)

    data_set = {"x_train": x_train,
                "x_valid": x_valid,
```

```
                "x_test": x_test,
                "y_train": y_train,
                "y_valid": y_valid,
                "y_test": y_test,
                "features": features}

    return data_set


def select_model(data_set):
    max_depth = [5, 10, 25, 50, 75]
    gini = []
    ig = []
    ig_count = 0
    gini_count = 0

    x_train = data_set["x_train"]
    x_valid = data_set["x_valid"]
    y_train = data_set["y_train"]
    y_valid = data_set["y_valid"]

    for depth in max_depth:
        dt_ig = DecisionTreeClassifier(criterion="entropy",
                                       max_depth=depth)
        dt_ig.fit(x_train, y_train)

        ig_prediction = dt_ig.predict(x_valid)
        for i in range(len(ig_prediction)):
            if ig_prediction[i] == y_valid[i]:
                ig_count += 1
        ig_accuracy = ig_count / len(ig_prediction)
        ig.append(ig_accuracy)
        ig_count = 0

    for depth in max_depth:
        dt_gini = DecisionTreeClassifier(criterion="gini",
                                         max_depth=depth)
        dt_gini.fit(x_train, y_train)

        gini_prediction = dt_gini.predict(x_valid)
        for i in range(len(gini_prediction)):
            if gini_prediction[i] == y_valid[i]:
                gini_count += 1
        gini_accuracy = gini_count / len(gini_prediction)
        gini.append(gini_accuracy)
        gini_count = 0

    for i in range(len(max_depth)):
        print("The accuracy we get by using info gain with max depth of "
              + str(max_depth[i]) + " = " + str(ig[i]))
    for i in range(len(max_depth)):
        print("The accuracy we get by using gini with max depth of "
              + str(max_depth[i]) + " = " + str(gini[i]))

```

```
108
109 def draw_tree(classifier, data_set):
110
111     x_train = data_set["x_train"]
112     y_train = data_set["y_train"]
113
114     trained = classifier.fit(x_train, y_train)
115     tree.export_graphviz(trained, feature_names=data_set["features"],
116                          class_names=["fake", "real"],
117                          out_file='textVisual.dot')
118     (graph,) = pydot.graph_from_dot_file('textVisual.dot')
119     im_name = '{}.png'.format("tree1")
120     graph.write_png(im_name)
121
122
123 def compute_information_gain(data_set, word):
124
125     x_train = data_set["x_train"]
126     y_train = data_set["y_train"]
127     vocab = data_set["features"]
128
129     real_pos = 0
130     real_neg = 0
131     fake_pos = 0
132     fake_neg = 0
133
134     index = vocab.index(word)
135
136     for i in range(len(y_train)):
137         if y_train[i] == 1:
138             if x_train[i, index] == 0:
139                 real_neg += 1
140             else:
141                 real_pos += 1
142         else:
143             if x_train[i][index] == 0:
144                 fake_neg += 1
145             else:
146                 fake_pos += 1
147
148     # Calculate root_entropy (H(Y)).
149     root_entropy = entropy(real_pos + real_neg, fake_pos + fake_neg)
150
151     # Calculate p(X=+) and p(X=-) with real_pos, real_neg,
152     # fake_pose and fake_neg.
153     pos_prob = (real_pos + fake_pos) / len(y_train)
154     neg_prob = (real_neg + fake_neg) / len(y_train)
155
156     # Calculate H(Y|X=+) and H(Y|X=-).
157     pos_entropy = entropy(real_pos, fake_pos)
158     neg_entropy = entropy(real_neg, fake_neg)
159
160     # Calculate H(Y|X)
161     cond_entropy = pos_prob * pos_entropy + neg_prob * neg_entropy
```

```
162
163     # Calculate IG(Y|X) = H(Y) - H(Y|X)
164     ig = root_entropy - cond_entropy
165     return ig
166
167
168 def entropy(num1, num2):
169     """
170     Helper function for computing entropy
171     """
172
173     total = num1 + num2
174     prob1 = num1 / total
175     prob2 = num2 / total
176
177     result = - prob1 * math.log(prob1, 2.0) - \
178              prob2 * math.log(prob2, 2.0)
179     return result
180
181
182 if __name__ == '__main__':
183     dataset = load_data("C:/Users/exle0/Documents/2019 Fall/CSC311/
    Project")
184     select_model(dataset)
185     classif = DecisionTreeClassifier(criterion='entropy', max_depth=50)
186     draw_tree(classif, dataset)
187     compute_information_gain(dataset, "the")
188     compute_information_gain(dataset, "donald")
189     compute_information_gain(dataset, "trump")
190     compute_information_gain(dataset, "hillary")
191     compute_information_gain(dataset, "america")
192
```

Question 3

1. (a)

$$P(\mathbf{t}|\mathbf{X},\mathbf{w}) = \prod_i^n [\frac{1}{(2\pi\sigma^2)^{1/2}} exp(-\frac{1}{2}(\mathbf{t}_i - \mathbf{X}_i^T\mathbf{w})^T\sigma^{-2}\mathbf{I}(\mathbf{t} - \mathbf{X}_i^T\mathbf{w}))]$$

$$logP(\mathbf{t}|\mathbf{X},\mathbf{w}) = -nlog(\sqrt{2\pi}\sigma) - \sum_{i=1}^n \frac{1}{2}(\mathbf{t}_i - \mathbf{X}_i^T\mathbf{w})^T\sigma^{-2}\mathbf{I}(\mathbf{t} - \mathbf{X}_i^T\mathbf{w}))] \qquad \text{take a log}$$

$$= \mathbf{C} - \frac{1}{2\sigma^2}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w}) \qquad \mathbf{C} \text{ is constant}$$

$$= \mathbf{C} - \frac{1}{2\sigma^2}(\mathbf{t}^T\mathbf{t} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{t} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

Computing the gradient w.r.t $\mathbf{w}$

$$\begin{aligned}\nabla\ell(\mathbf{w}) = \frac{\partial}{\partial\mathbf{w}} logP(\mathbf{t}|\mathbf{X},\mathbf{w}) &= \frac{\partial}{\partial\mathbf{w}}(-\frac{1}{2\sigma^2})(\mathbf{t}^T\mathbf{t} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{t} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}) \\ &= -\frac{1}{2\sigma^2}(-2\mathbf{X}^T\mathbf{t} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}) \\ &= -\frac{1}{\sigma^2}(-\mathbf{X}^T\mathbf{t} + \mathbf{X}^T\mathbf{X}\mathbf{w}) \\ &= 0\end{aligned}$$

Therefore,

$$-\frac{1}{\sigma^2}(-\mathbf{X}^T\mathbf{t} + \mathbf{X}^T\mathbf{X}\mathbf{w}) = 0$$

$$\nabla\ell(\mathbf{w}) = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t}) = 0$$

According to the week3 lecture,

$$\nabla\ell(\mathbf{w}) = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t}) = 0 \Rightarrow \hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

(b)

$$\begin{aligned}E[\hat{\mathbf{w}}] &= E[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}] \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T E[\mathbf{t}] \\ &= (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{w} \\ &= \mathbf{w}\end{aligned}$$

$$
\begin{aligned}
Var[\hat{\mathbf{w}}] &= ((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)Var[\mathbf{t}]((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T \\
&= ((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)\sigma^2 I((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}
$$

From the preliminaries.pdf, we know that the linear transformation of a Gaussian random vector is again Gaussian. Therefore,

$$\hat{\mathbf{w}} \sim \mathcal{N}(\mathbf{w}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$$

2. First, of all the definition of $\hat{\mathbf{w}}_{MAP}$ is

$$\hat{\mathbf{w}}_{MAP} = argmax_{\mathbf{w}} P(\mathbf{t}|\mathbf{X}, \mathbf{w})P(\mathbf{w})$$

Note that

$$
\begin{aligned}
logP(\mathbf{t}|\mathbf{X}, \mathbf{w}) &= -nlog(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(\mathbf{t}_i - \mathbf{X}_i^T\mathbf{w})^2 \\
logP(\mathbf{w}) &= -nlog(\sqrt{2\pi}\tau) - \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

Then,

$$
\begin{aligned}
logP((\mathbf{t}|\mathbf{X}, \mathbf{w})P(\mathbf{w})) &= logP(\mathbf{t}|\mathbf{X}, \mathbf{w}) + logP(\mathbf{w}) \\
&= -nlog(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{i}^{n}(\mathbf{t}_i - \mathbf{X}_i^T\mathbf{w})^2 - nlog(\sqrt{2\pi}\tau) - \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

We have to take a gradient w.r.t. $\mathbf{w}$ to obtain the maximum.

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} logP((\mathbf{t}|\mathbf{X}, \mathbf{w})P(\mathbf{w})) &= \frac{\partial}{\partial \mathbf{w}}[-nlog(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{i}^{n}(\mathbf{t}_i - \mathbf{X}_i^T\mathbf{w})^2 - nlog(\sqrt{2\pi}\tau) - \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}] \\
&= \frac{1}{2\sigma^2}2\mathbf{X}^T(\mathbf{t} - \mathbf{X}\mathbf{w}) - \frac{1}{2\tau^2}2\mathbf{w}\mathbf{I} \\
&= \frac{1}{\sigma^2}(\mathbf{X}^T\mathbf{t} - \mathbf{X}^T\mathbf{X}\mathbf{w}) - \frac{1}{\tau^2}\mathbf{w}\mathbf{I} \\
&= \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{t} - \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{1}{\tau^2}\mathbf{w}\mathbf{I}
\end{aligned}
$$

Setting it to equal to 0, then we get

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t} \qquad \text{where } \lambda = \frac{\sigma^2}{\tau^2}$$

```python
"""
CSC311 Assignment 1 Question 3 Sangwoo Kim
"""
import numpy as np
import matplotlib.pyplot as plot


def shuffle_data(data):
    t, x = data
    data = np.concatenate((x, t.reshape(len(t), 1)), axis=1)
    np.random.shuffle(data)
    t = data[:, -1]
    x = data[:, :-1]
    return (t, x)


def split_data(data, num_folds, fold):
    t, x = data
    len_block = len(t) // num_folds
    start = len_block * (fold - 1)
    end = start + len_block
    x_data_fold = x[start: end, :]
    t_data_fold = t[start:end]
    x_data_rest = np.concatenate((x[: start, :], x[end:, :]))
    t_data_rest = np.array(list(t[:start]) + list(t[end:]))
    return (t_data_fold, x_data_fold), (t_data_rest, x_data_rest)


def train_model(data, lambd):
    t, x = data

    xtx = np.dot(x.T, x)
    inv = np.linalg.inv(xtx + lambd * np.identity(len(x[0])))
    beta =  np.dot(inv, np.dot(x.T, t.reshape(len(t), 1)))
    return beta


def predict(data, model):
    t, x = data
    predictions = np.dot(x, model)
    return predictions


def loss(data, model):
    t, x = data
    t = t.reshape(len(t), 1)
    prediction = predict(data, model)
    loss_value = np.dot((t - prediction).T, (t - prediction))/float(len(t
))
    return float(loss_value)


def cross_validation(data, num_folds, sequence):
    error = np.zeros((len(sequence)))
```

```
    data = shuffle_data(data)
    for i in range(len(sequence)):
        lambd = sequence[i]
        cv_loss_lmd = 0
        for fold in range(1, num_folds + 1):
            val_cv, train_cv = split_data(data, num_folds, fold)
            model = train_model(train_cv, lambd)
            cv_loss_lmd += loss(val_cv, model)
        error[i] = cv_loss_lmd / float(num_folds)
    return error


def get_train_test_error(train_data, test_data, sequence):
    l_trains = []
    l_tests = []
    for i in sequence:
        model = train_model(train_data, i)
        l_train = loss(train_data, model)
        l_test = loss(test_data, model)
        l_trains.append(l_train)
        l_tests.append(l_tests)
        print("Lambda = {}, Training error = {}, Test error = {}".
              format(i, l_train, l_test))
    return l_trains, l_tests


if __name__ == '__main__':
    # LOAD DATA SET
    data_train = {'X': np.genfromtxt('C:/Users/exle0/Documents/2019 Fall/CSC311/Project/data_train_X.csv', delimiter=','),
    't': np.genfromtxt('C:/Users/exle0/Documents/2019 Fall/CSC311/Project/data_train_y.csv', delimiter=',')}
    data_test = {'X': np.genfromtxt('C:/Users/exle0/Documents/2019 Fall/CSC311/Project/data_test_X.csv', delimiter=','),
    't': np.genfromtxt('C:/Users/exle0/Documents/2019 Fall/CSC311/Project/data_test_y.csv', delimiter=',')}

    # Create a sequence of lambda
    lambd_seq = np.linspace(0.02, 1.5, num=50)

    # Get train error and test error for each lambda
    train_loss, test_loss = get_train_test_error((data_train['t'], data_train['X']), (data_test['t'], data_test['X']), lambd_seq)

    # Get errors for five fold and ten fold.
    five_fold_cv_error = cross_validation((data_train['t'], data_train['X']), 5, lambd_seq)
    ten_fold_cv_error = cross_validation((data_train['t'], data_train['X']), 10, lambd_seq)

    # 3-d

    plot.plot(lambd_seq, train_loss, label="train loss")
    plot.plot(lambd_seq, test_loss, label='test loss')
```

```
101     plot.plot(lambd_seq, five_fold_cv_error, label="5 folds")
102     plot.plot(lambd_seq, ten_fold_cv_error, label="10 folds")
103
104     plot.xlabel('lambd')
105     plot.ylabel('error')
106
107     plot.legend()
108     plot.savefig("Q3d.png")
109     plot.show()
110
111     print("Optimal lambda of five-fold cross validation = {}".
112           format(lambd_seq[np.argmin(five_fold_cv_error)]))
113     print("Optimal lambda of ten-fold cross validation = {}".
114           format(lambd_seq[np.argmin(ten_fold_cv_error)]))
115
```

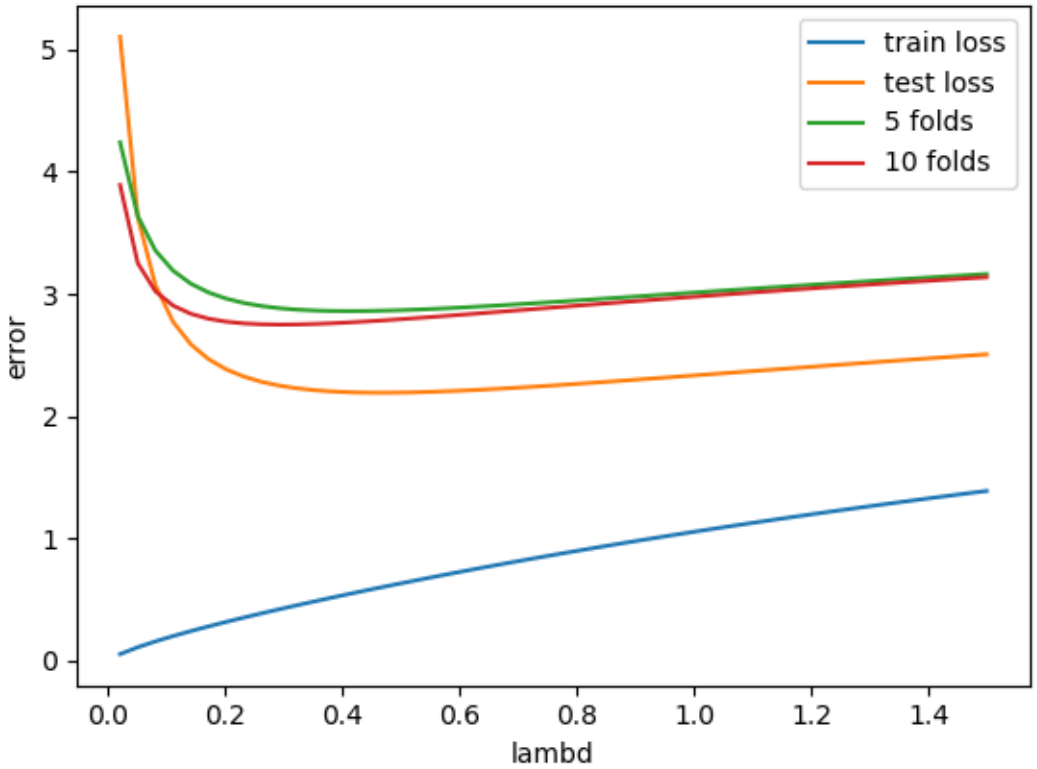
3. (a) It's in the above code

   (b) It's also in code. The order should be shuffle data → split data → train model → predict → loss.

   (c) These are training and test errors

```
C:\Users\exle0\Anaconda3\envs\csc311\python.exe "C:/Users/exle0/Documents/2019 Fall/CSC311/Project/asn1_q3.py"
Lambda = 0.02, Training error = 0.04973649154744332, Test error = 5.106959783590836
Lambda = 0.050204081632653066, Training error = 0.10583474384162328, Test error = 3.6308336211216234
Lambda = 0.08040816326530613, Training error = 0.15397029597916248, Test error = 3.070316730406787
Lambda = 0.11061224489795919, Training error = 0.19754784437778414, Test error = 2.770942353158943
Lambda = 0.14081632653061224, Training error = 0.2381299849860619, Test error = 2.587353108983472
Lambda = 0.1710204081632653, Training error = 0.2765778425897503, Test error = 2.4660055412368447
Lambda = 0.20122448979591837, Training error = 0.3134077787636036, Test error = 2.3821345226047566
Lambda = 0.23142857142857143, Training error = 0.3489485445553723, Test error = 2.3226049520446184
Lambda = 0.2616326530612245, Training error = 0.3834197262933723, Test error = 2.279768261137276
Lambda = 0.2918367346938776, Training error = 0.41697403355811985, Test error = 2.2488554532446545
Lambda = 0.32204081632653064, Training error = 0.4497214500203871, Test error = 2.226732904004412
Lambda = 0.3522448979591837, Training error = 0.48174368509289694, Test error = 2.211254262729599
Lambda = 0.38244897959183677, Training error = 0.5131031658916717, Test error = 2.2008990424758865
Lambda = 0.41265306122448986, Training error = 0.5438488213771094, Test error = 2.1945597160259624
Lambda = 0.4428571428571429, Training error = 0.5740199114355471, Test error = 2.1914104953100058
Lambda = 0.47306122448979593, Training error = 0.6036486259742307, Test error = 2.190823326742993
Lambda = 0.503265306122449, Training error = 0.6327618883816957, Test error = 2.1923123455903792
Lambda = 0.5334693877551021, Training error = 0.6613826315442853, Test error = 2.195496109072519
Lambda = 0.5636734693877552, Training error = 0.6895307165151995, Test error = 2.2000712857772204
Lambda = 0.5938775510204082, Training error = 0.7172236043300798, Test error = 2.2057939302127716
Lambda = 0.6240816326530613, Training error = 0.744476854302519, Test error = 2.212465901009172
Lambda = 0.6542857142857144, Training error = 0.7713044984197196, Test error = 2.2199248420992013
Lambda = 0.6844897959183673, Training error = 0.7977193260052295, Test error = 2.228036679363815
Lambda = 0.7146938775510204, Training error = 0.8237331025510496, Test error = 2.2366899238664684
Lambda = 0.7448979591836735, Training error = 0.8493567396836511, Test error = 2.2457912928224273
Lambda = 0.7751020408163266, Training error = 0.874600428464105, Test error = 2.2552623053593623
Lambda = 0.8053061224489797, Training error = 0.8994737449027032, Test error = 2.2650366087044236
Lambda = 0.8355102040816327, Training error = 0.9239857342239822, Test error = 2.2750578581743093
Lambda = 0.8657142857142858, Training error = 0.9481449787418323, Test error = 2.2852780216246797
Lambda = 0.8959183673469389, Training error = 0.9719596529922098, Test error = 2.295656012489221
Lambda = 0.9261224489795918, Training error = 0.9954375688851004, Test error = 2.3061565795529297
Lambda = 0.9563265306122449, Training error = 1.0185862129836332, Test error = 2.316749399043649
Lambda = 0.986530612244898, Training error = 1.0414127775313982, Test error = 2.3274083274349833
Lambda = 1.016734693877551, Training error = 1.0639241864834081, Test error = 2.3381107828617442
Lambda = 1.0469387755102042, Training error = 1.0861271175193965, Test error = 2.348837230176407
Lambda = 1.0771428571428572, Training error = 1.10802802080713, Test error = 2.359570750067329
Lambda = 1.1073469387755104, Training error = 1.1296331351214226, Test error = 2.3702966767743345
Lambda = 1.137551020408163, Training error = 1.1509485017993362, Test error = 2.381002292102869
Lambda = 1.1677551020408163, Training error = 1.1719799769147732, Test error = 2.3916765658918986
Lambda = 1.1979591836734695, Training error = 1.1927332419795647, Test error = 2.4023099350067407
Lambda = 1.2281632653061225, Training error = 1.213213813418383, Test error = 2.412894114434182
Lambda = 1.2583673469387755, Training error = 1.2334270510175607, Test error = 2.4234219352488062
Lambda = 1.2885714285714287, Training error = 1.2533781655104117, Test error = 2.43388720516795
Lambda = 1.3187755102040817, Training error = 1.2730722254317446, Test error = 2.444284588171957
Lambda = 1.3489795918367347, Training error = 1.2925141633503363, Test error = 2.4546095002775874
Lambda = 1.3791836734693879, Training error = 1.311708781568855, Test error = 2.4648580190469573
Lambda = 1.4093877551020408, Training error = 1.3306607573652038, Test error = 2.475026804816475
Lambda = 1.439591836734694, Training error = 1.3493746478366548, Test error = 2.4851130319587353
Lambda = 1.469795918367347, Training error = 1.3678548943978908, Test error = 2.4951143287599042
Lambda = 1.5, Training error = 1.3861058269757283, Test error = 2.5050287247173055
```

(d) This is a plot graph




```
52 Optimal lambda of five-fold cross validation = 0.3522448979591837
53 Optimal lambda of ten-fold cross validation = 0.38244897959183677
54
55 Process finished with exit code 0
56
```

As we can see from above result, five-fold cross validation's optimal lambda is 0.3522448979591837 and ten-fold cross validation's optimal lambda is 0.38244897959183677. We can check that test loss, 5 folds and 10 folds decrease at first and then start increasing while train loss keep increasing as lambda gets bigger.