CSC311 Assignment 1

Yujie Wu, 1003968904

October 5, 2019

1. a)Expected Value: since X, Y are independent uniformly distributed random variables,
$E[Z] = E[(X-Y)^2] = E[X^2 - 2XY + Y^2]$
$= E[X^2] - 2E[XY] + E[Y^2]$
$= E[X^2] - 2E[X] * E[Y] + E[Y^2]$
$= \frac{1^2+1*0+0^2}{3} - 2 * \frac{1+0}{2} * \frac{1+0}{2} + \frac{1^2+1*0+0^2}{3}$
$= \frac{1}{3} - \frac{1}{2} + \frac{1}{3} = \frac{1}{6}$

$Var[Z] = Var[(X-Y)^2] = E[((X-Y)^2)^2] - E[(X-Y)^2]^2$
$= E[(X-Y)^4] - E[(X-Y)^2]^2$
$= E[X^4 - 4X^3Y + 6X^2Y^2 - 4XY^3 + Y^4] - E[(X-Y)^2]^2$
$= E[X^4] - 4E[X^3]E[Y] + 6E[X^2]E[Y^2] - 4E[X]E[Y^3] + E[Y^4] - (\frac{1}{6})^2$
$= \frac{1^4+1^30+1^20^2+10^3+0^4}{5} - 4\frac{1}{4}\frac{1}{2} + 6\frac{1}{3}\frac{1}{3} - 4\frac{1}{2}\frac{1}{4} + \frac{1^4+1^30+1^20^2+10^3+0^4}{5} - \frac{1}{36}$
$= \frac{1}{5} - \frac{1}{2} + \frac{2}{3} - \frac{1}{2} + \frac{1}{5} - \frac{1}{36}$
$= \frac{7}{180}$

b. $E[R] = E[X_1] + ... + E[X_d] = \sum_{i=1}^{d} E[Z_i] = d * \frac{1}{6} = \frac{d}{6}$
$Var[R] = Var[Z_1] + Var[Z_2] + ... + Var[Z_d] + 2Cov[Z_1Z_2] + 2Cov[Z_1Z_3] + ... + 2Cov[Z_(n-1)Z_n]$
$= Var[Z_1] + Var[Z_2] + ... + Var[Z_d] + 0 + ... + 0$
$= d\frac{7}{180}$
$= \frac{7d}{180}$

c. The maximum squared euclidean distance for dimension d unit cube can be written as:
$$(\sqrt{(1-0)^2 + ... + (1-0)^2})^2 = d$$
alone with mean for euclidean distance, as dimension d increases,
$$\lim_{d\to\infty} \frac{d}{6} = \frac{\infty}{6} = \infty = \lim_{d\to\infty} d,$$
thus it confirms that "most points are far away in high dimensions".

Then, we can calculate CV of R by $\frac{SD}{mean}$, where SD is $\sqrt{Var} = \sqrt{\frac{7d}{180}}$,
thus CV[R] = $\frac{\sqrt{\frac{7d}{180}}}{\frac{d}{6}}$, as the dimension increases, $\lim_{d\to\infty} \frac{\sqrt{\frac{7d}{180}}}{\frac{d}{6}} = 0$. This means although points in high dimensional space have higher standard deviation, the variability relative to the mean is approaching to o, thus they have sam distance.

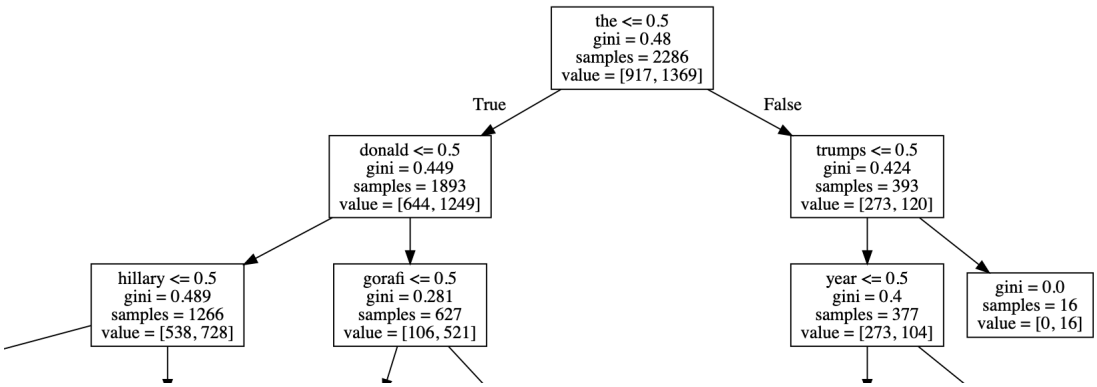
2.
a) Code will be provided at the end of question 2.

b) Code will be provided at the end of question 2.
max_depths selected to train the decision tree: [2, 3, 4, 5, 6]

result accuracies of split criteria = "information gain": [0.6877551020408164, 0.7244897959183674, 0.7224489795918367, 0.7244897959183674, 0.7244897959183674]

result accuracies of split criteria = "gini coefficient": [0.6877551020408164, 0.7224489795918367, 0.726530612244898, 0.7306122448979592, 0.736734693877551]

c) Code will be provided at the end of question 2.
Hyperparameters that result highest validation score: *split_criterion* = 'gini coefficient', *max_depth* = 6 with visualization:



d) Code will be provided at the end of question 2.
topmost split for c): the, information gain = 0.05301135064440743
some other splits:
"csc311" = 0.0, since it is completely not relevent
"watch" = 0.010140038653621897
"trump" = 0.03362895380345787
"saying" = 1.854969234238446e-05
"america" = 0.007378140084137841

code:

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import graphviz
import math

def load_data(fake_file, real_file):
    fake_set = open(fake_file, "r").read().split("\n")[:-1]
    real_set = open(real_file, "r").read().split("\n")[:-1]
    whole_set = fake_set + real_set
    lable = [0] * len(fake_set) + [1] * len(real_set)

    vectorizer = CountVectorizer()
    data = vectorizer.fit_transform(whole_set)
    train_data, test_data, train_lable, test_lable = train_test_split(data, lable, test_size=0.3)

    test_data, validation_data, test_lable, validation_lable = train_test_split(
        test_data, test_lable, test_size=0.5)

    comb = [train_data, train_lable, validation_data, validation_lable, test_data, test_data,
            vectorizer.get_feature_names(), vectorizer]
    return comb
```

```python
def select_model(train_data, train_label, validation_data, validation_label, feature_name):
    trees = []
    scores = []
    for i in range(2):
        for j in range(5):
            criter = "entropy" if i else "gini"
            dt = DecisionTreeClassifier(criterion=criter, max_depth=(j+2))
            trees.append(dt.fit(train_data, train_label))
    for item in trees:
        scores.append(item.score(validation_data, validation_label))

    print(trees)
    print(scores)
    selected = trees[scores.index(max(scores))]
    print(selected)
    dot_data = tree.export_graphviz(selected, out_file=None, feature_names=feature_name)
    graph = graphviz.Source(dot_data)
    graph.render("iris")
    return selected
```

```python
def compute_information_gain(data, lable, keyword, vectorizer):
    present = 0
    absent = 0
    present_real = 0
    absent_real = 0
    for x in range(data.shape[0]):
        document = vectorizer.inverse_transform(data[x])[0]
        if keyword in document:
            present += 1
            present_real += lable[x]
        else:
            absent += 1
            absent_real += lable[x]

    total_real = present_real + absent_real
    total_fake = len(lable) - total_real
    present_fake = present - present_real
    absent_fake = absent - absent_real

    root_entropy = -(total_real/len(lable)) * math.log2(total_real/len(lable)) - (
            total_fake/len(lable)) * math.log2(total_fake/len(lable))
    present_entropy = (0 if present == 0 else -(present_real/present) * math.log2(
        0 if present == 0 else present_real/present) + (-(
        0 if present == 0 else present_fake/present) * math.log2(0 if present == 0 else present_fake/present)))
    absent_entropy = (-(0 if absent == 0 else absent_real/absent) * math.log2(
        0 if absent == 0 else absent_real/absent) + (-(
        0 if absent == 0 else absent_fake/absent) * math.log2(0 if absent == 0 else absent_fake/absent)))
    IG = root_entropy - (present_entropy * (present/len(lable)) + absent_entropy * (absent/len(lable)))
    print(IG)
```

```python
if __name__ == "__main__":
    lst = load_data(
        "/Users/wuqiang/Desktop/311/a1/data/clean_fake.txt", "/Users/wuqiang/Desktop/311/a1/data/clean_real.txt")
    tree = select_model(lst[0], lst[1], lst[2], lst[3], lst[6])
    root = lst[6][list(tree.feature_importances_).index(max(tree.feature_importances_))]
    print(root)
    compute_information_gain(lst[0], lst[1], root, lst[7]) #root

    compute_information_gain(lst[0], lst[1], "csc311", lst[7])
    compute_information_gain(lst[0], lst[1], "watch", lst[7])
    compute_information_gain(lst[0], lst[1], "trump", lst[7])
    compute_information_gain(lst[0], lst[1], "saying", lst[7])
    compute_information_gain(lst[0], lst[1], "america", lst[7])
```

3.

3.1 a) $\log \ell(w) = \underset{w}{\text{argmax}} \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma^2 I}} exp\{-\frac{1}{2\sigma^2}(t - Xw)^T(t - Xw)\}$

$$\log \ell(w) = \underset{w}{\text{argmax}}(\log(2\pi\sigma^2 I)^{-\frac{1}{2}} + \log(exp\{-\frac{1}{2\sigma^2}(t - Xw)^T(t - Xw)\})$$
$$= \underset{w}{\text{argmin}}(t - Xw)^T(t - Xw)$$

$$s(w) = t^T t - t^T Xw - w^T X^T t + w^T X^T Xw$$
$$= t^T t - 2t^T Xw + w^T X^T Xw$$

$$\frac{d-2tX^T w}{dw} = -2tX^T, \frac{dw^T X^T Xw}{dw} = 2X^T Xw$$
$$\frac{ds(w)}{dw} = -2tX^T + X^T Xw = 0$$
$$2X^T X\hat{w} = 2X^T t$$
$$\hat{w} = (X^T X)^{-1} X^T t$$

b) $E[\hat{w}] = E[X^T X)^{-1} X^T (wX + e)$

$$= E[(X^T X) - 1X^T wX + (X^T X)^{-} 1X^T e]$$

$$
\begin{aligned}
&= E[w + (X^T X^{-}1)X^T e] \\
&= E[w] + 0 \\
&= w \\
Cov(\hat{w}) &= E[(\hat{w} - w)(\hat{w} - w)^T] \\
&= E[(X^T X)^{-1} X^T \epsilon\epsilon^T X (X^T X)^{-1}] \\
&= (X^T X)^{-1} X^T E(\epsilon\epsilon^T) X (X^T X)^{-1} \\
&= \sigma^2 (X^T X)^{-1} X^T I X (X^T X)^{-1} \\
&= \sigma^2 (X^T X)^{-1}
\end{aligned}
$$

Thus, the distribution of $\hat{w}$ should be $\mathcal{N}(w, \sigma^2(X^T X)^{-1})$.

3.2 $\hat{w}_{MAP} \propto \underset{w}{\text{argmax}}\ p(t|X,w)p(w|x) =$

$\text{argmax} \sum_{i=1}^{n} (\log \frac{1}{\sqrt{2\pi\sigma^2 I}} exp\{-\frac{1}{2\sigma^2}(t - Xw)^T(t - Xw)\} * \log \frac{1}{\sqrt{2\pi\tau^2 I}} exp\{-\frac{1}{2\tau^2}(w^T w\})$

$= \underset{w}{\text{argmax}} (\log(2\pi\sigma^2 I)^{-\frac{1}{2}} + \log(exp\{-\frac{1}{2\sigma^2}(t - Xw)^T(t - Xw)\}) + ((2\pi\tau^2 I)^{-\frac{1}{2}} + \log(exp\{-\frac{1}{2\tau^2} w^T w\}))$

$= \underset{w}{\text{argmin}} \frac{1}{2\sigma^2}(t - Xw)^T(t - Xw) + \frac{1}{2\tau^2}\log(w^T w)$

$s(w) = \frac{1}{2\sigma^2}(t^T t - t^T Xw - w^T X^T t + w^T X^T Xw) + \frac{1}{2\tau^2} w^T w$

$= \frac{1}{2\sigma^2} t^T t - \frac{1}{2\sigma^2} 2t^T Xw + \frac{1}{2\sigma^2} w^T X^T Xw + \frac{1}{2\tau^2} w^T w$

$\frac{d - \frac{1}{2\sigma^2} 2tX^T w}{dw} = -2\frac{1}{2\sigma^2} tX^T$, $\frac{d \frac{1}{2\sigma^2} w^T X^T Xw}{dw} = 2\frac{1}{2\sigma^2} X^T Xw$

$\frac{d \frac{1}{2\tau^2} w^T w}{dw} = 2\frac{1}{2\tau^2} w$

$$
\begin{aligned}
\frac{ds(w)}{dw} = -2\frac{1}{2\sigma^2} tX^T + \frac{1}{2\sigma^2} X^T Xw + \frac{1}{2\tau^2} 2w &= 0 \\
\tau^2(-2tX^T) + \tau^2(2X^T Xw) + \sigma^2 2w &= 0 \\
\tau^2(X^T Xw) + \sigma^2 w &= \tau^2 tX^T \\
X^T Xw + \frac{\sigma^2}{\tau^2} w &= tX^T \\
(X^T X + \frac{\sigma^2}{\tau^2} I)w &= tX^T \\
w &= (X^T X + \lambda I)^{-1} tX^T, \text{ where } \lambda = \frac{\sigma^2}{\tau^2}
\end{aligned}
$$

3.3

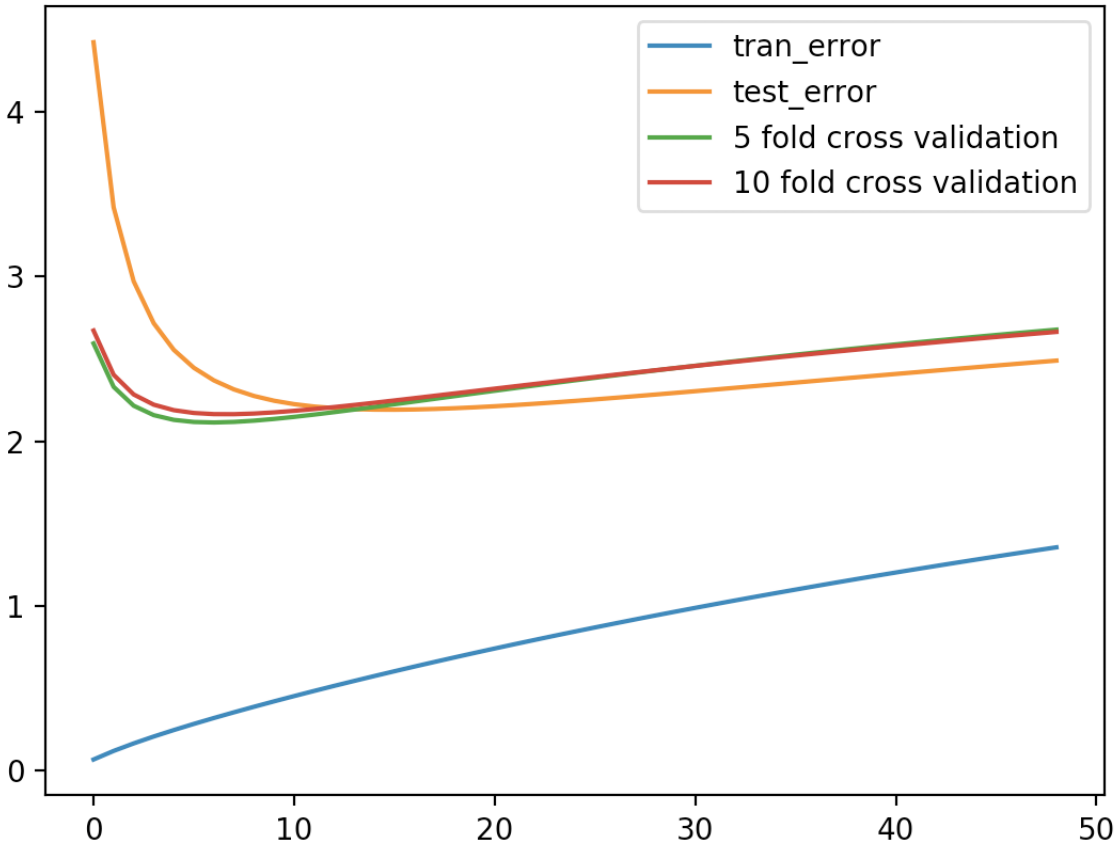
a) Code will be provided below question 3.

b) Code will be provided below question 3.

c) Code will be provided below question 3.

d) Code will be provided below question 3.

plot:



value of λ I proposed for 5 folds: `5 fold validation lambda: 0.2072`

value of λ I proposed for 10 folds: `10 fold validation lambda: 0.2368`

The error value for the cross validation and *test_error* curve start with high level and decreased rapidly at very low lambda, but not the initial one, and then increased slowly as the lambda increases. This is because at the initial lambda, we have only little error on train data since it over fitted, but we get high test error. As the lambda increases, we found the best lambda for this set of data. After this lambda value, our model get more and more complex and the result bias and variance increases, so the results spread out wider and further away from expected result. This explains why error value increases after we found best lambda. code:

```python
import numpy as np
import matplotlib.pyplot as plt

data_train = {'X': np.genfromtxt('/Users/wuqiang/Desktop/311/a1/data/data_train_X.csv', delimiter=','),
        't': np.genfromtxt('/Users/wuqiang/Desktop/311/a1/data/data_train_Y.csv', delimiter=',')}

data_test = {'X': np.genfromtxt('/Users/wuqiang/Desktop/311/a1/data/data_test_X.csv', delimiter=','),
        't': np.genfromtxt('/Users/wuqiang/Desktop/311/a1/data/data_test_Y.csv', delimiter=',')}


def shuffle_data(data):
    data_shf = {}
    i = np.random.permutation(data['t'].size)
    np.random.shuffle(i)
    data_shf['X'] = data['X'][i]
    data_shf['t'] = data['t'][i]
    return data_shf


def split_data(data, num_folds, fold):
    x = np.vsplit(data['X'], num_folds)
    t = np.array_split(data['t'], num_folds)
    data_fold = {'X': x.pop(fold - 1), 't': t.pop(fold - 1)}
    data_rest = {'X': np.asarray([itemm for sublist in x for itemm in sublist]),
                 't': np.asarray([itm for sublistt in t for itm in sublistt])}
    return data_fold, data_rest


def train_model(data, lambd):
    return np.dot(np.dot(np.linalg.inv(np.dot(np.matrix.transpose(data['X']), data['X']) + lambd * np.identity(
        data['X'][0].size)), np.matrix.transpose(data['X'])), data['t'])


def predict(data, model):
    return np.dot(data['X'], model)
```

```python
def loss(data, model):
    return sum(np.square(data['t'] - predict(data, model))) / data['t'].size


def cross_validation(data_in, num_folds, lambd_seq):
    cv_error = []
    data = shuffle_data(data_in)
    for i in range(len(lambd_seq)):
        lambd = lambd_seq[i]
        cv_loss_lmd = 0
        for fold in range(num_folds):
            val_cv, train_cv = split_data(data, num_folds, fold)
            model = train_model(train_cv, lambd)
            cv_loss_lmd += loss(val_cv, model)
        cv_error.append(cv_loss_lmd / num_folds)
    return cv_error


def getlambd(amount, lower, upper):
    unit = (upper - lower) / amount
    return [x * unit for x in range(1, amount)]
```

```python
if __name__ == "__main__":
    lambd_seq = getlambd(50, 0.02, 1.5)

    tran_error = []
    test_error = []
    train = shuffle_data(data_train)
    test = shuffle_data(data_test)
    cross_5_error = (cross_validation(data_test, 5, lambd_seq))
    cross_10_error = (cross_validation(data_test, 10, lambd_seq))
    for item in lambd_seq:
        m_tra = train_model(train, item)
        m_test = train_model(train, item)
        tran_error.append(loss(train, m_tra))
        test_error.append(loss(test, m_test))
    lamb_5 = lambd_seq[cross_5_error.index(min(cross_5_error))]
    lamb_10 = lambd_seq[cross_10_error.index(min(cross_10_error))]
    print("5 fold validation lambda: " + str(lamb_5))
    print("10 fold validation lambda: " + str(lamb_10))
    plt.plot(tran_error, label="tran_error")
    plt.plot(test_error, label="test_error")
    plt.plot(cross_5_error, label="5 fold cross validation")
    plt.plot(cross_10_error, label="10 fold cross validation")
    plt.legend(("tran_error", "test_error", "5 fold cross validation", "10 fold cross validation"), loc='upper right')
    plt.show()
```