Q1.

a) $E((X-Y)^2) = E(X^2 - 2XY + Y^2)$

$= E(X^2) - 2E(X)E(Y) + E(Y^2)$

$= \int_0^1 \frac{1}{b-a} x^2 dx - 2\int_0^1 \frac{1}{b-a} x dx \int_0^1 \frac{1}{b-a} y dy + \int_0^1 \frac{1}{b-a} y^2 dy$

$= \frac{1}{b-a} \cdot \frac{1}{3} x^3 \Big|_0^1 - 2\left(\frac{1}{b-a} \cdot \frac{1}{2} x^2 \Big|_0^1\right)\left(\frac{1}{b-a} \cdot \frac{1}{2} \cdot y^2 \Big|_0^1\right) + \frac{1}{b-a} \cdot \frac{1}{3} y^3 \Big|_0^1$

$= \frac{1}{3(b-a)} - \frac{2}{2^2(b-a)^2} + \frac{1}{3(b-a)}$

$= \frac{2}{3(b-a)} - \frac{1}{2(b-a)^2} = \frac{4(b-a)-3}{6(b-a)^2}$ where $b=1, a=0$

$= \frac{1}{6}$

$Var((X-Y)^2) = E((X-Y)^4) - E((X-Y)^2)^2$

$= E(x^4 - 4x^3y + 6x^2y^2 - 4xy^3 + y^4) - E(x^2 - 2xy + y^2)^2$

$= E(x^4) - 4E(x^3)E(y) + 6E(x^2)E(y^2) - 4E(x)E(y^3) + E(y^4)$
$- \left(E(x^2) - 2E(x)(y) + E(y^2)\right)^2$

Integrated similarly as expectations

$= \left(\frac{1}{5} - 4 \cdot \frac{1}{4} \cdot \frac{1}{2} + 6 \cdot \frac{1}{3} \cdot \frac{1}{3} - 4 \cdot \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{5}\right) - \left(\frac{1}{3} - 2 \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{3}\right)^2$

$= \left(\frac{6+10-15}{15}\right) - \left(\frac{1}{6}\right)^2 = \frac{1}{15} - \frac{1}{36} = \frac{7}{180}$

b) $E(R) = E(Z_1 + Z_2 + \cdots + Z_d)$

$= E(Z_1) + E(Z_2) + \cdots + E(Z_d)$

$= E((X_1 - Y_1)^2) + E((X_2 - Y_2)^2) + \cdots + E((X_d - Y_d)^2)$

$= \sum_{i=1}^{d} E((X_i - Y_i)^2) = \sum_{i=1}^{d} \frac{1}{6} = \frac{d}{6}$

$Var(R) = Var(Z_1 + Z_2 + \cdots + Z_d)$

$= Var(Z_1) + Var(Z_2) + \cdots + Var(Z_d)$

$= \sum_{i=1}^{d} Var(Z_i) = \sum_{i=1}^{d} Var((X_i - Y_i)^2)$

$= \sum_{i=1}^{d} \frac{7}{180} = \frac{7d}{180}$

c) From part b), it is obvious that as the dimension goes higher, the mean and variance of Euclidean distance grow a lot along with it.

Q2. (Since it is painful to read python syntax in plain text document, I put it into screenshots)

a)

```python
def load_data(fake_file, real_file):
    dataset = []
    f = open(fake_file)
    line = f.readline()
    while (line):
        dataset.append([line,0])
        line = f.readline()
    f.close()

    f = open(real_file)
    line = f.readline()
    while (line):
        dataset.append([line,1])
        line = f.readline()
    f.close()

    np.random.seed(0)
    np.random.shuffle(dataset)

    str_array = []
    label_array = []
    for i in range(0, len(dataset)-1):
        str_array.append(dataset[i][0])
        label_array.append(dataset[i][1])

    count_vectorizer = CountVectorizer(analyzer='word')
    dataset_vector = count_vectorizer.fit_transform(str_array).toarray()
    dataset_vector_feature_names = count_vectorizer.get_feature_names()

    headline_train, headline_temp, label_train, label_temp = train_test_split(dataset_vector, label_array, test_size = 0.3,shuffle=False)
    headline_vad, headline_test, label_vad, label_test = train_test_split(headline_temp, label_temp, test_size = 0.5,shuffle=False)

    return headline_train, headline_vad, headline_test, label_train, label_vad, label_test, dataset_vector_feature_names,dataset
```

b)

```python
def select_model(headline_train, label_train, headline_vad, label_vad):
    criteria = ['entropy', 'gini']
    maxdepth = [ 2, 4, 6, 8, 10]
    max_score = 0
    best_model = None
    for c in criteria:
        for d in maxdepth:
            model = DecisionTreeClassifier(criterion=c, max_depth=d)
            ##print(model)
            model.fit(headline_train, label_train)
            result = model.predict(headline_vad)
            score = model.score(headline_vad, label_vad)
            print(score)
            if score > max_score:
                max_score = score
                best_model = model
    return best_model
```

The output for select_model is as following:
0.6877551020408164
0.7224489795918367
0.7183673469387755
0.7326530612244898
0.7428571428571429
0.6877551020408164
0.7224489795918367
0.726530612244898
0.7285714285714285
0.7489795918367347
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
random_state=None, splitter='best')

c)
The visualization of the tree with a max depth of 3 is:
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
edge [fontname=helvetica] ;
0 [label="the <= 0.5\ngini = 0.483\nsamples = 2285\nvalue = [930, 1355]\nclass = real", fillcolor="#c1e0f7"] ;
1 [label="donald <= 0.5\ngini = 0.455\nsamples = 1926\nvalue = [675, 1251]\nclass = real", fillcolor="#a4d2f3"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="hillary <= 0.5\ngini = 0.493\nsamples = 1273\nvalue = [560, 713]\nclass = real", fillcolor="#d5eaf9"] ;
1 -> 2 ;
3 [label="trumps <= 0.5\ngini = 0.483\nsamples = 1201\nvalue = [490, 711]\nclass = real", fillcolor="#c1e1f7"] ;
2 -> 3 ;
4 [label="(...)", fillcolor="#C0C0C0"] ;
3 -> 4 ;
31 [label="(...)", fillcolor="#C0C0C0"] ;
3 -> 31 ;
34 [label="administration <= 0.5\ngini = 0.054\nsamples = 72\nvalue = [70, 2]\nclass = fake", fillcolor="#e6853f"] ;
2 -> 34 ;
35 [label="(...)", fillcolor="#C0C0C0"] ;
34 -> 35 ;
40 [label="(...)", fillcolor="#C0C0C0"] ;
34 -> 40 ;
41 [label="de <= 0.5\ngini = 0.29\nsamples = 653\nvalue = [115, 538]\nclass = real", fillcolor="#63b2eb"] ;
1 -> 41 ;
42 [label="hillary <= 0.5\ngini = 0.278\nsamples = 646\nvalue = [108, 538]\nclass = real", fillcolor="#61b1ea"] ;
41 -> 42 ;
43 [label="(...)", fillcolor="#C0C0C0"] ;
42 -> 43 ;

78 [label="(...)", fillcolor="#C0C0C0"] ;
42 -> 78 ;
91 [label="gini = 0.0\nsamples = 7\nvalue = [7, 0]\nclass = fake", fillcolor="#e58139"] ;
41 -> 91 ;
92 [label="trumps <= 0.5\ngini = 0.412\nsamples = 359\nvalue = [255, 104]\nclass = fake", fillcolor="#f0b48a"] ;
0 -> 92 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
93 [label="era <= 0.5\ngini = 0.388\nsamples = 345\nvalue = [254, 91]\nclass = fake", fillcolor="#eeae80"] ;
92 -> 93 ;
94 [label="person <= 0.5\ngini = 0.379\nsamples = 339\nvalue = [253, 86]\nclass = fake", fillcolor="#eeac7c"] ;
93 -> 94 ;
95 [label="(...)", fillcolor="#C0C0C0"] ;
94 -> 95 ;
120 [label="(...)", fillcolor="#C0C0C0"] ;
94 -> 120 ;
121 [label="policy <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]\nclass = real", fillcolor="#61b1ea"] ;
93 -> 121 ;
122 [label="(...)", fillcolor="#C0C0C0"] ;
121 -> 122 ;
123 [label="(...)", fillcolor="#C0C0C0"] ;
121 -> 123 ;
124 [label="clean <= 0.5\ngini = 0.133\nsamples = 14\nvalue = [1, 13]\nclass = real", fillcolor="#48a5e7"] ;
92 -> 124 ;
125 [label="gini = 0.0\nsamples = 13\nvalue = [0, 13]\nclass = real", fillcolor="#399de5"] ;
124 -> 125 ;
126 [label="gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = fake", fillcolor="#e58139"] ;
124 -> 126 ;
}

```python
def extract_and_visualize(best_model,features):
    classnames = ['fake', 'real']
    return export_graphviz(best_model,feature_names = features, class_names = classnames,
                           max_depth=3,filled=True,rounded=True)
```

d)

```
def compute_information_gain(keyword, dataset, headline_train):
    keyword_in_real = 0
    keyword_notin_real = 0
    keyword_in_fake = 0
    keyword_notin_fake = 0
    training_data = dataset[:len(headline_train)]

    for i in range(0, len(headline_train)):
        if keyword in training_data[i][0] and training_data[i][1] == 1:
            keyword_in_real += 1
        elif keyword not in training_data[i][0] and training_data[i][1]  == 1:
            keyword_notin_real += 1
        elif keyword in training_data[i][0] and training_data[i][1]  == 0:
            keyword_in_fake += 1
        elif keyword not in training_data[i][0] and training_data[i][1]  == 0:
            keyword_notin_fake += 1

    real = float(keyword_in_real + keyword_notin_real)
    fake = float(keyword_in_fake + keyword_notin_fake)
    bothin = float(keyword_in_real + keyword_in_fake)
    bothout = float(keyword_notin_real + keyword_notin_fake)

    H_keyword = -(real/len(headline_train)) * np.log2(real/len(headline_train)) - (fake/len(headline_train)) * np.
    log2(fake/len(headline_train))

    H_Cond_keyword = (- (keyword_in_real/bothin) * np.log2(keyword_in_real/bothin) - (keyword_in_fake/bothin) * np
    .log2(keyword_in_fake/bothin))*(bothin/len(headline_train)) +(- (keyword_notin_real/bothout) * np.log2(
    keyword_notin_real/bothout) - (keyword_notin_fake/bothout) * (np.log2(keyword_notin_fake/bothout)))*(bothout/
    len(headline_train))

    result = round(H_keyword - H_Cond_keyword, 4)
    print("The information gain for '", keyword,"' is: ", result);

    return result
```

The output for this function is:
The information gain for ' the ' is: 0.0568
The information gain for ' clinton ' is: 0.0105
The information gain for ' hillary ' is: 0.0364

Appendix for Q2:

```
if __name__ == "__main__":
    headline_train, headline_vad, headline_test, label_train, label_vad, label_test, features, dataset = load_data
    ("./clean_fake.txt", "./clean_real.txt")

    best_model = select_model(headline_train,label_train,headline_vad,label_vad)

    print(best_model)
    tree = extract_and_visualize(best_model, features)
    print(tree)


    compute_information_gain("the", dataset, headline_train)
    compute_information_gain("clinton", dataset, headline_train)
    compute_information_gain("hillary", dataset, headline_train)
```

Q3.

3.1 a)

Maximum likelihood:

$$L(\hat{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y^{(i)} - w^T x^{(i)})^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \cdot e^{\frac{-\sum_{i=1}^{n} (y^{(i)} - w^T x^{(i)})^2}{2\sigma^2}}$$

Log-likelihood:

$$l(\hat{w}) = \log L(\hat{w})$$

$$= -\frac{1}{2\sigma^2}(y^{(i)} - w^T x^{(i)})^2 - \log(\sqrt{2\pi\sigma^2})$$

$$= -\frac{1}{2\sigma^2}(Y - XW)^T(Y - XW) - \sum_{i=1}^{n} \log\sqrt{2\pi\sigma^2}$$

gradient: $$\frac{\partial l(\hat{w})}{\partial W} = \frac{1}{2\sigma^2} \frac{\partial (Y - XW)^T (Y - XW)}{\partial W}$$

$$= \frac{1}{2\sigma^2} \frac{\partial (Y^2 - 2X^T W Y + X^T X W^2)}{\partial W}$$

$$= \frac{-\cancel{2}X^T Y + \cancel{2}X^T X W}{\cancel{2}\sigma^2} = 0$$

$$-X^T Y + X^T X W = 0$$

$$X^T X W = X^T Y$$

$$W = (X^T X)^{-1} X^T Y$$

3.1 b) $E(\hat{w}) = E((X^TX)^{-1}X^Tt)$

$$= (X^TX)^{-1}X^TE(t)$$

$$= (X^TX)^{-1}X^TXw = w$$

$$Var(\hat{w}) = Var((X^TX)^{-1}X^Tt)$$

$$= Var(At) = AVar(t)A^T$$

$$= ((X^TX)^{-1}X^T)\cdot(\sigma^2 I)\cdot((X^TX)^{-1}X^T)^T$$

$$= \sigma^2((X^TX)^{-1}X^T)\cdot X((X^TX)^{-1})^T$$

$$= \sigma^2((X^TX)^{-1}X^T)X((X^TX)^T)^{-1}$$

$$= \sigma^2((X^TX)^{-1}X^T)X(X^TX)^{-1}$$

$$= \sigma^2(X^TX)^{-1}$$

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

3.2 $t \mid X, w \sim N(Xw, \sigma^2 I)$

normal prior on $w \mid X$: $w \sim N(0, r^2 I)$

$$\hat{w}_{MAP} = \underset{w}{\operatorname{argmax}} \{ p(w \mid X, t) \propto P(t \mid X, w) \, p(w \mid X) \}$$

$$p(t \mid X, w) \propto \exp\left\{ -\frac{\sigma^2}{2} (Y - Xw)^T (Y - Xw) \right\}$$

$$p(w \mid X) \propto \exp\left\{ \frac{\gamma^2}{2} w^T w \right\}$$

$$\log(p(t \mid X, w)) + \log(p(w \mid X)) \propto$$

$$-\frac{\sigma^2}{2} (Y^T Y - 2w^T X^T Y + w^T X^T X w) - \frac{\gamma^2}{2} w^T w$$

$$\frac{\partial \log(p(t \mid X, w)) \log(p(w \mid X))}{\partial w}$$

$$= -\frac{\sigma^2}{2} (-2X^T Y + 2X^T X w) + \left(\frac{r^2}{2} w + \frac{\gamma^2}{2} w \right)$$

$$= \sigma^2 X^T Y - \sigma^2 X^T X w + \gamma^2 w = 0.$$

$$-\sigma^2 X^T X w + \gamma^2 w = -\sigma^2 X^T Y$$

$$\sigma^2 X^T X w - \gamma^2 w = \sigma^2 X^T Y$$

$$X^T X w - \frac{r^2}{\sigma^2} w = X^T Y$$

$$w = (X^T X + \lambda I)^{-1} X^T Y$$

Q3.3

a)

```
data_train = {"X": np.genfromtxt("data_train_X.csv", delimiter=","),
              "t": np.genfromtxt("data_train_y.csv", delimiter=",")}
data_test = {"X": np.genfromtxt("data_test_X.csv", delimiter=","),
             "t": np.genfromtxt("data_test_y.csv", delimiter=",")}
```

b)

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

def shuffle_data(data):
    length = np.arange(len(data["X"]))
    np.random.seed(0)
    index = np.random.permutation(length)

    result = {"X":[], "t":[]}
    for i in index:
        result["X"].extend(data["X"][index])
        result["t"].extend(data["t"][index])

    return result


def split_data(data, num_folds, fold):
    fold_size = int(len(data["X"])//num_folds)
    data_fold = {"X":[],"t":[]}
    data_rest = {"X":[], "t":[]}

    data_fold["X"] = data["X"][(fold-1)*fold_size: fold*fold_size]
    data_fold["t"] = data["t"][(fold-1)*fold_size: fold*fold_size]
    data_rest["X"] = data["X"][0: (fold-1)*fold_size] + data["X"][fold*fold_size: len(data["X"])]
    data_rest["t"] = data["t"][0: (fold-1)*fold_size] + data["t"][fold*fold_size: len(data["t"])]
    return data_fold, data_rest
```

```python
def train_model(data, lambd):
    train_data = np.array(data["X"])
    label = np.array(data["t"])
    label.reshape(len(data["t"]), 1)
    I_shape = train_data.shape[1]
    data_transpose = train_data.transpose()
    dataT_data = np.dot(data_transpose, train_data)
    inv = np.linalg.inv(dataT_data+lambd*np.identity(I_shape))
    inv_data_transpose = np.dot(inv, data_transpose)

    result = np.dot(inv_data_transpose, label)
    return result

def predict(data, model):
    train_data = np.array(data["X"])
    w = np.array(model).reshape(train_data.shape[1],1)
    prediction = np.dot(train_data, w)

    return prediction

def loss(data, model):
    prediction = np.array(predict(data, model)).reshape(len(data["t"]) ,1)
    target = np.array(data["t"]).reshape(len(data["t"]) ,1)
    sum_sqr = 0
    for i in range(0, len(prediction)):
        sum_sqr += (target[i] - prediction[i]) ** 2

    return sum_sqr/len(data["t"])


def cross_validation(data, num_folds, lambd_seq):
    data = shuffle_data(data)
    cv_error = len(lambd_seq) * [0]
    for i in range(0, len(lambd_seq)):
        print("cross", i)
        lambd = lambd_seq[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)
        cv_error[i] = cv_loss_lmd/num_folds
    return cv_error
```

c)

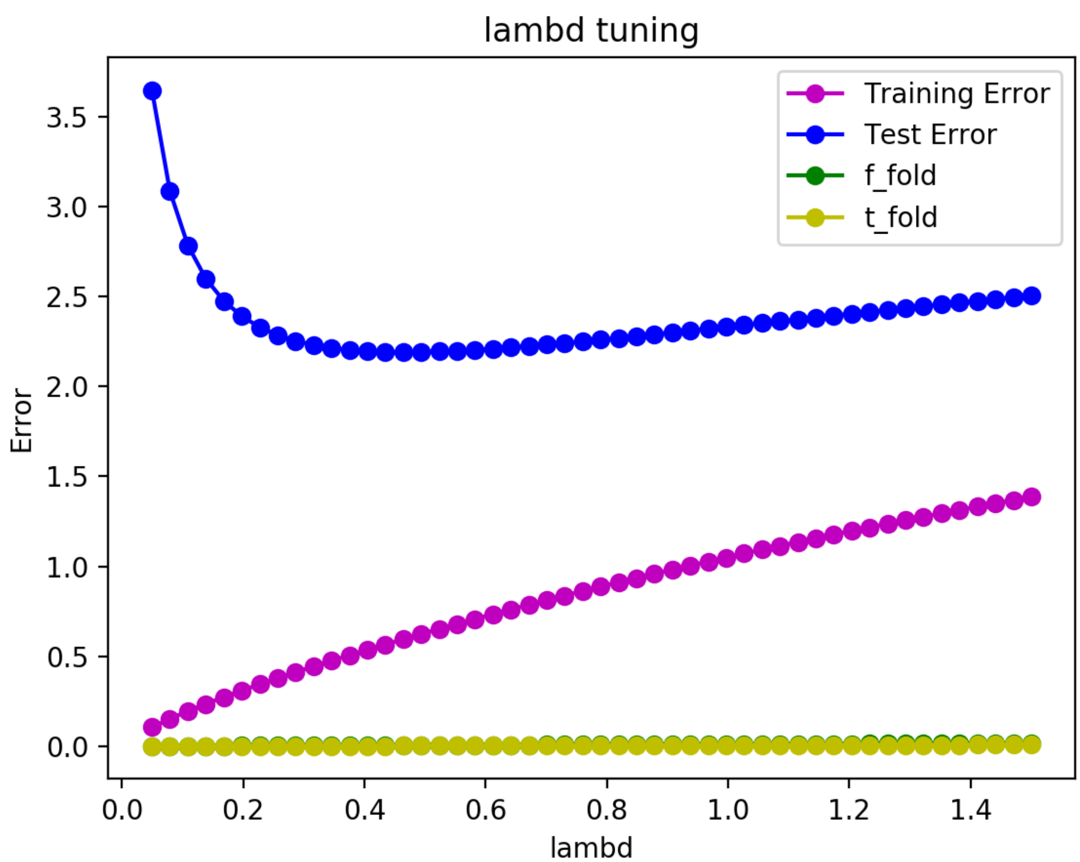
```
def lambd_seq_error(train_data, test_data, lambd_seq):

    train_error = []
    test_error = []
    for i in range(0, len(lambd_seq)):

        model = train_model(train_data, lambd_seq[i])
        train_error.append(loss(train_data, model))
        test_error.append(loss(test_data, model))
    return train_error, test_error



if __name__ == "__main__":
    data_train = {"X": np.genfromtxt("data_train_X.csv", delimiter=","),
                  "t": np.genfromtxt("data_train_y.csv", delimiter=",")}
    data_test = {"X": np.genfromtxt("data_test_X.csv", delimiter=","),
                 "t": np.genfromtxt("data_test_y.csv", delimiter=",")}
    lambd_seq = []
    acc = 0.02
    for i in range(0, 50):
        acc += (1.5-0.02)/50
        lambd_seq.append(acc)

    training_err, test_err = lambd_seq_error(data_train, data_test, lambd_seq)
    for i in range(0, len(lambd_seq)):
        print("training error of lambd " ,lambd_seq[i]," is : ",training_err[i][0])
    for i in range(0, len(lambd_seq)):
        print("test error of lambd " ,lambd_seq[i]," is : ",test_err[i][0])
```

The following output will show the training error and testing error for each lambd:
training error of lambd 0.049600000000000005 is : 0.10480791529912924
training error of lambd 0.0792 is : 0.1521464552011346
training error of lambd 0.10880000000000001 is : 0.19502879738018036
training error of lambd 0.13840000000000002 is : 0.23497059256003724
training error of lambd 0.16800000000000004 is : 0.2728125649094834
training error of lambd 0.19760000000000005 is : 0.30906150506234664
training error of lambd 0.22720000000000007 is : 0.3440413963261663
training error of lambd 0.2568000000000001 is : 0.37796931381054727
training error of lambd 0.2864000000000001 is : 0.4109965611827344
training error of lambd 0.3160000000000001 is : 0.4432322729953669
training error of lambd 0.34560000000000013 is : 0.474757594054879
training error of lambd 0.37520000000000014 is : 0.5056345320456189
training error of lambd 0.40480000000000016 is : 0.5359116689491276
training error of lambd 0.4344000000000002 is : 0.5656279518857784
training error of lambd 0.4640000000000002 is : 0.5948152722711206
training error of lambd 0.4936000000000002 is : 0.6235002592205299
training error of lambd 0.5232000000000002 is : 0.6517055508923163
training error of lambd 0.5528000000000002 is : 0.679450711406649

training error of lambd 0.5824000000000001 is : 0.7067529024687077
training error of lambd 0.6120000000000001 is : 0.733627382263509
training error of lambd 0.6416000000000001 is : 0.7600878808119871
training error of lambd 0.6712 is : 0.786146885714929
training error of lambd 0.7008 is : 0.8118158620560552
training error of lambd 0.7303999999999999 is : 0.837105423360862
training error of lambd 0.7599999999999999 is : 0.8620254657798713
training error of lambd 0.7895999999999999 is : 0.8865852743659296
training error of lambd 0.8191999999999998 is : 0.9107936079822923
training error of lambd 0.8487999999999998 is : 0.9346587677082028
training error of lambd 0.8783999999999997 is : 0.9581886523994319
training error of lambd 0.9079999999999997 is : 0.9813908041765305
training error of lambd 0.9375999999999997 is : 1.0042724459597692
training error of lambd 0.9671999999999996 is : 1.0268405126824212
training error of lambd 0.9967999999999996 is : 1.0491016774475364
training error of lambd 1.0263999999999995 is : 1.0710623736156943
training error of lambd 1.0559999999999996 is : 1.0927288135992166
training error of lambd 1.0855999999999997 is : 1.1141070049753965
training error of lambd 1.1151999999999997 is : 1.1352027644052498
training error of lambd 1.1447999999999998 is : 1.156021729746189
training error of lambd 1.1743999999999999 is : 1.176569370670245
training error of lambd 1.204 is : 1.196850998039069
training error of lambd 1.2336 is : 1.2168717722390834
training error of lambd 1.2632 is : 1.2366367106423
training error of lambd 1.2928000000000002 is : 1.2561506943278709
training error of lambd 1.3224000000000002 is : 1.2754184741752588
training error of lambd 1.3520000000000003 is : 1.2944446764202628
training error of lambd 1.3816000000000004 is : 1.3132338077493582
training error of lambd 1.4112000000000005 is : 1.3317902599949871
training error of lambd 1.4408000000000005 is : 1.3501183144839688
training error of lambd 1.4704000000000006 is : 1.3682221460826895
training error of lambd 1.5000000000000007 is : 1.3861058269757283
test error of lambd 0.049600000000000005 is : 3.6470640267598005
test error of lambd 0.0792 is : 3.086147163181072
test error of lambd 0.10880000000000001 is : 2.784853140492007
test error of lambd 0.13840000000000002 is : 2.5992954389128937
test error of lambd 0.16800000000000004 is : 2.476166554235303
test error of lambd 0.19760000000000005 is : 2.3907203231549756
test error of lambd 0.22720000000000007 is : 2.329797633877647
test error of lambd 0.2568000000000001 is : 2.285720949317952
test error of lambd 0.2864000000000001 is : 2.2536958317114326
test error of lambd 0.3160000000000001 is : 2.230567957357408
test error of lambd 0.34560000000000013 is : 2.2141744723023415
test error of lambd 0.37520000000000014 is : 2.2029817440164137
test error of lambd 0.40480000000000016 is : 2.1958717036194
test error of lambd 0.4344000000000002 is : 2.1920100264502773
test error of lambd 0.4640000000000002 is : 2.190761670910101
test error of lambd 0.4936000000000002 is : 2.1916349892650326
test error of lambd 0.5232000000000002 is : 2.1942437023297865
test error of lambd 0.5528000000000002 is : 2.198280393502023
test error of lambd 0.5824000000000001 is : 2.203497634435469
test error of lambd 0.6120000000000001 is : 2.209694288756179
test error of lambd 0.6416000000000001 is : 2.216705404361297
test error of lambd 0.6712 is : 2.2243946403930237

test error of lambd 0.7008 is : 2.2326485153433513
test error of lambd 0.7303999999999999 is : 2.241371984001442
test error of lambd 0.7599999999999999 is : 2.25048499775357
test error of lambd 0.7895999999999999 is : 2.259919801966904
test error of lambd 0.8191999999999998 is : 2.26961879239761
test error of lambd 0.8487999999999998 is : 2.2795328001871757
test error of lambd 0.8783999999999997 is : 2.289619708739487
test error of lambd 0.9079999999999997 is : 2.2998433299750802
test error of lambd 0.9375999999999997 is : 2.310172485041678
test error of lambd 0.9671999999999996 is : 2.320580247477478
test error of lambd 0.9967999999999996 is : 2.331043316414703
test error of lambd 1.0263999999999995 is : 2.341541494602066
test error of lambd 1.0559999999999996 is : 2.3520572514666065
test error of lambd 1.0855999999999997 is : 2.3625753555889184
test error of lambd 1.1151999999999997 is : 2.373082564161839
test error of lambd 1.1447999999999998 is : 2.3835673594806703
test error of lambd 1.1743999999999999 is : 2.394019724448201
test error of lambd 1.204 is : 2.404430950599387
test error of lambd 1.2336 is : 2.414793473354352
test error of lambd 1.2632 is : 2.42510073016703
test error of lambd 1.2928000000000002 is : 2.435347038004011
test error of lambd 1.3224000000000002 is : 2.445527487206154
test error of lambd 1.3520000000000003 is : 2.455637849285451
test error of lambd 1.3816000000000004 is : 2.4656744966161988
test error of lambd 1.4112000000000005 is : 2.475634332311908
test error of lambd 1.4408000000000005 is : 2.485514728851977
test error of lambd 1.4704000000000006 is : 2.495313474246957
test error of lambd 1.5000000000000007 is : 2.505028724717304

d)



On the above error plot, my f-fold and t_fold are overlapping each other with errors mostly slightly over zero, i.e., very small. The training error would gradually go higher as lambd increases. However, the test error dropped a lot when lambd increased from 0.02 to 0.2 approximately and after that the test error would slightly increase. The best fit lambd should be around 0.2 since it is the lowest point for test error and training error would not be too high with this lambd.

```python
if __name__ == "__main__":
    data_train = {"X": np.genfromtxt("data_train_X.csv", delimiter=","),
                  "t": np.genfromtxt("data_train_y.csv", delimiter=",")}
    data_test = {"X": np.genfromtxt("data_test_X.csv", delimiter=","),
                 "t": np.genfromtxt("data_test_y.csv", delimiter=",")}
    lambd_seq = []
    acc = 0.02
    for i in range(0, 50):
        acc += (1.5-0.02)/50
        lambd_seq.append(acc)

    training_err, test_err = lambd_seq_error(data_train, data_test, lambd_seq)
    for i in range(0, len(lambd_seq)):
        print("training error of lambd " ,lambd_seq[i]," is : ",training_err[i][0])
    for i in range(0, len(lambd_seq)):
        print("test error of lambd " ,lambd_seq[i]," is : ",test_err[i][0])

    f_fold = cross_validation(data_train, 5, lambd_seq)
    t_fold = cross_validation(data_test, 10, lambd_seq)

    plt.plot(lambd_seq, training_err, 'mo-')
    plt.plot(lambd_seq, test_err, 'bo-')
    plt.plot(lambd_seq, f_fold, 'go-')
    plt.plot(lambd_seq, t_fold, 'yo-')

    plt.title('lambd tuning')
    plt.ylabel('Error')
    plt.xlabel('lambd')
    plt.legend(['Training Error', 'Test Error','f_fold','t_fold'], loc='upper right')
    plt.show()
```