hw 1 writing part:

Question 1:

(a) $Z = (X-Y)^2$ $E[Z]$? $Var(Z)$?

$$E[(X-Y)^2] = E[(X\cdot X + Y\cdot Y - 2X\cdot Y)]$$
$$= E[XX] + E[YY] - 2E[XY]$$
$$= E[XX] + E[YY] - 2E[X]E[Y]$$
$$= \int_0^1 P(x)\cdot x^2 dx + \int_0^1 P(y)\cdot y^2 dy - 0.5$$
$$= \int_0^1 x^2 dx + \int_0^1 y^2 dy - \frac{1}{2}$$
$$= \frac{x^3}{3}\Big|_0^1 \times 2 - \frac{1}{2}$$
$$= \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$$

$$Var[(X-Y)^2] = E[(X-Y)^4] - E[(X-Y)^2]^2$$
$$= E[X^4 - 4X^3Y + 6X^2Y^2 - 4XY^3 + Y^4] - \left(\frac{1}{6}\right)^2$$
$$= E[X^4] - 4E[X^3]E[Y] + 6E[X^2]E[Y^2] - 4E[X]E[Y^3] + E[Y^4] - \frac{1}{36}$$
$$= \frac{1}{5} - 4\cdot\frac{1}{4}\cdot\frac{1}{2} + 6\cdot\frac{1}{3}\cdot\frac{1}{3} - 4\cdot\frac{1}{2}\cdot\frac{1}{4} + \frac{1}{5} - \frac{1}{36}$$
$$= \frac{1}{5} - \frac{1}{2} + \frac{2}{3} - \frac{1}{2} + \frac{1}{5} - \frac{1}{36}$$
$$= \frac{2}{5} - 1 + \frac{2}{3} - \frac{1}{36}$$
$$= \frac{1}{15} - \frac{1}{36}$$
$$= \frac{12}{180} - \frac{5}{180} = \frac{7}{180}$$

(b) $E[R] = E[Z_1 + Z_2 + \cdots + Z_d] = \sum_{i=1}^{d} E[Z_i] = \sum_{i=1}^{d} E[Z]$

$Var[R] = Var[Z_1 + Z_2 + \cdots + Z_d]$

$= Var\left[\sum_{i=1}^{d} Z_i\right]$

$= \sum_{i=1}^{d} Var[Z_i] + \sum_{i\neq j}^{d} Cov(Z_i, Z_j)$

$= d\,Var[Z]$

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

$= dE[Z]$

$= \frac{1}{6} d$

(c) The maximum squared Euclidean distance of cube in dimension d

$= \sum_{i=1}^{d} \max(x_i - y_i)^2 = \sum_{i=1}^{d} 1 = d$

According to central limit theorem, when d is large, R is approximately normally distributed, for normal distribution, we know that approximately 99.7% of data is within 3 standard deviations of the mean. $3\sigma_R = 3\sqrt{\frac{7}{180}d} = \sqrt{\frac{7}{20}d}$. Comparing to the maximum squared Euclidean distance of the hypercube which is d, most data deviate from mean in a really small region, which means the squared Euclidean distances of most of the points in hypercube of large dimension are almost the same.

Question 3.1:

known: $X \in \mathbb{R}^{n \times m}$ $(n \geq m)$ $\vec{t} \in \mathbb{R}^n$ $\boxed{\vec{t} \mid (X, \vec{w}) \sim N(X\vec{w}, \sigma^2 I)}$ this means, given the input and the weight of each feature, the distribution of the label

Proof: $MLE(\vec{w}) = \hat{w} = (X^T X)^{-1} X^T \vec{t}$ $I \in \mathbb{R}^{n \times n}$

Note X means there are n inputs each has m features

$\vec{t}$ is the label of each input

(a) $P(\vec{t} \mid (X, \vec{w})) = (2\pi)^{-\frac{n}{2}} \det(\sigma^2 I)^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{t} - X\vec{w})^T (\sigma^2 I)^{-1} (\vec{t} - X\vec{w})}$

$\log P(\vec{t} \mid (X, \vec{w})) = \underbrace{\log (2\pi)^{-\frac{n}{2}}}_{A} + \underbrace{\log \det(\sigma^2 I)^{-\frac{1}{2}}}_{B} + \left(-\frac{1}{2}(\vec{t} - X\vec{w})^T (\sigma^2 I)^{-1} (\vec{t} - X\vec{w})\right)$

$\log P(\vec{t} \mid (X, \vec{w})) = A + B - \frac{1}{2}(\vec{t} - X\vec{w})^T (\sigma^2 I)^{-1} (\vec{t} - X\vec{w})$

$$\frac{\partial \log P(\vec{t} \mid (X, \vec{w}))}{\partial \vec{w}} = -\frac{1}{2} \frac{\partial (\vec{t} - X\vec{w})^T (\sigma^2 I)^{-1} (\vec{t} - X\vec{w})}{\partial \vec{w}}$$

$$= -\frac{1}{2\sigma^2} \frac{\partial (\vec{t} - X\vec{w})^T (\vec{t} - X\vec{w})}{\partial \vec{w}}$$

$$= -\frac{1}{2\sigma^2} 2 \frac{\partial (\vec{t} - X\vec{w})}{\partial \vec{w}} (\vec{t} - X\vec{w})$$

$$= -\frac{1}{\sigma^2} \frac{\partial (\vec{t} - X\vec{w})}{\partial \vec{w}} (\vec{t} - X\vec{w})$$

$$= -\frac{1}{\sigma^2} \left(\frac{\partial \vec{t}}{\partial \vec{w}} - \frac{\partial X\vec{w}}{\partial \vec{w}}\right)(\vec{t} - X\vec{w})$$

$$\nabla \log P(\vec{t}|(X,\vec{w})) = \frac{1}{\sigma^2} X^T(\vec{t} - X\vec{w})$$

$$\nabla \log P(\vec{t}|(X,\vec{w})) = 0$$

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

$$X^T\vec{t} - X^TX\vec{w} = 0$$

$$X^T\vec{t} = X^TX\vec{w}$$

$$(X^TX)^{-1}X^T\vec{t} = \vec{w}$$

given X, what parameter $\vec{w}$ most possibly generate $\vec{t}$

$$\underline{\vec{w} = (X^TX)^{-1}X^T\vec{t}}$$

(b) Since, the linear transformation of Gaussian random vector is again Gaussian, and $\vec{t}$ follows Gaussian distribution, since $\vec{w}$ is a linear transformation of $\vec{t}$, therefore, $\vec{w}$ also follows Gaussian distribution.

Distribution of $\vec{w}$:

$$(X^TX)^{-1}X^T\vec{t}\,|(X,\vec{w}) \sim \mathcal{N}((X^TX)^{-1}X^TX\vec{w}, (X^TX)^{-1}X^T\sigma^2I((X^TX)^{-1}X^T)^T)$$

$$\underline{\sim \mathcal{N}(\vec{w}, \sigma^2(X^TX)^{-1}X^T((X^TX)^{-1}X^T)^T)}$$

Question 3.2

$A|(B|C) = A|B\cap C \quad \vec{w}|X \sim \mathcal{N}(0, \tau^2 I)$

Proof $\hat{w}_{MAP} = (X^TX + \lambda I)^{-1}X^T\vec{t}$

$$\hat{w}_{MAP} = \underset{\vec{w}}{\operatorname{argmax}} \{P(\vec{w}|X,\vec{t})\}$$

$$= \underset{\vec{w}}{\operatorname{argmax}} \{P(\vec{t}|X,\vec{w})\,p(\vec{w}|X)\}$$

$$= \underset{\vec{w}}{\operatorname{argmax}} (2\pi)^{-\frac{n}{2}} \det(\sigma^2 I^{n\times n})^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{t}-X\vec{w})^T(\sigma^2 I)^{-1}(\vec{t}-X\vec{w})} \times (2\pi)^{-\frac{m}{2}} \det(\tau^2 I^{m\times m})^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{w}^T(\tau^2 I^{m\times m})^{-1}\vec{w})}$$

$$= \underset{\vec{w}}{\operatorname{argmax}} \underbrace{\log(2\pi)^{-\frac{n}{2}-\frac{m}{2}} \det(\sigma^2 I^{n\times n})^{-\frac{1}{2}} \det(\tau^2 I^{m\times m})^{-\frac{1}{2}}}_{A} - \frac{1}{2}(\vec{t}-X\vec{w})^T(\sigma^2 I)^{-1}(\vec{t}-X\vec{w}) - \frac{1}{2}\vec{w}^T(\tau^2 I^{m\times m})^{-1}\vec{w}$$

$$\frac{2\left(A - \frac{1}{2}(\vec{t}-X\vec{w})^T(\sigma^2 I)^{-1}(\vec{t}-X\vec{w}) - \frac{1}{2}\vec{w}^T(\tau^2 I^{m\times m})^{-1}\vec{w}\right)}{\partial \vec{w}}$$

$$= -\frac{1}{2}\cdot\frac{1}{\sigma^2}\frac{\partial(\vec{t}-X\vec{w})^T(\vec{t}-X\vec{w})}{\partial\vec{w}} - \frac{1}{2}\frac{\partial\vec{w}^T(\gamma^2 I^{m\times m})^{-1}\vec{w}}{\partial\vec{w}}$$

$$= -\frac{1}{\sigma^2}\frac{\partial(\vec{t}-X\vec{w})}{\partial\vec{w}}(\vec{t}-X\vec{w}) - \frac{1}{2\gamma^2}\frac{\partial\vec{w}^T\vec{w}}{\partial\vec{w}}$$

$$= \frac{1}{\sigma^2}(X^T)(\vec{t}-X\vec{w}) - \frac{1}{\gamma^2}\vec{w}$$

$$= \frac{1}{\sigma^2}(X^T\vec{t} - X^TX\vec{w}) - \frac{1}{\gamma^2}\vec{w}$$

$$= \frac{1}{\sigma^2}X^T\vec{t} - \frac{1}{\sigma^2}X^TX\vec{w} - \frac{1}{\gamma^2}\vec{w}$$

$$\frac{\partial P(\vec{t}|X,\vec{w})P(\vec{w}|X)}{\partial\vec{w}} = 0$$

$$\Leftrightarrow \frac{1}{\sigma^2}X^T\vec{t} - \frac{1}{\sigma^2}X^TX\hat{w} - \frac{1}{\gamma^2}\hat{w} = 0$$

$$\frac{1}{\sigma^2}X^T\vec{t} = \left(\frac{1}{\sigma^2}X^TX + \frac{1}{\gamma^2}I\right)\hat{w}$$

$$\cancel{\sigma^2}\,X^T\vec{t} = \cancel{\sigma^2}\left(X^TX + \frac{\sigma^2}{\gamma^2}I\right)\hat{w}$$

$$\hat{w} = \left(X^TX + \frac{\sigma^2}{\gamma^2}I\right)^{-1}X^T\vec{t}$$

$$\underline{\hat{w} = (X^TX + \lambda I)^{-1}X^T\vec{t}} \quad \text{where } \lambda = \frac{\sigma^2}{\gamma^2}$$

Question 2

October 4, 2019

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
```

0.0.1 (a)

[3]:
```
def load_data():
    # read from txt file, label the data
    real_title = pd.read_csv("data/clean_real.txt", header=None)
    real_title.columns = ["title"]
    real_title["is real"] = 1
    fake_title = pd.read_csv("data/clean_fake.txt", header=None)
    fake_title.columns = ["title"]
    fake_title["is real"] = 0
    # combine real and fake titles
    title = pd.concat([real_title, fake_title]).reset_index(drop=True)
    vectorizer = CountVectorizer()

    # seperate the document and the label, vectorize the word
    X = vectorizer.fit_transform(title['title'])
    y = title['is real']

    # split the training set and test set

    # first split to 70% training set, 30% test set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

    # then split to 15% test set, 15% validation set
    X_test, X_validate, y_test, y_validate = train_test_split(X_test, y_test,␣
↪test_size=0.5, random_state=42)
    return vectorizer, X_train, X_test, X_validate, y_train, y_test, y_validate
```

[4]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import ParameterGrid
```

[5]:
```
def validation_score(test, predict):
    values = test.values
    correct = 0
    for i in range(len(values)):
        if(values[i] == predict[i]):
            correct += 1
    return correct/len(values)
```

0.0.2 (b)

[6]:
```
def select_model(X_train, X_test, X_validate, y_train, y_test, y_validate):

    # init the classifier
    classifier = DecisionTreeClassifier(random_state=0)

    # tune hyperparameters by finding the best accuracy among the model trained␣
 ↪by the combination below
    param_grid = {
        'criterion': ['gini', 'entropy'],
        'max_depth': range(1, 2000)
    }

    accuracy = 0
    for param in ParameterGrid(param_grid):
        classifier.set_params(**param)
        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_validate)
        # save if best
        current_score = validation_score(y_validate, y_pred)
        if current_score > accuracy:
            accuracy = current_score
            best_grid = param

    # train the model with the parameters which yeild the best performance
    classifier.set_params(**best_grid)
    classifier.fit(X_train, y_train)

    # output the accuracy metric
    print(validation_score(y_test, classifier.predict(X_test)))
    return classifier
```

[7]:
```
vectorizer, X_train, X_test, X_validate, y_train, y_test, y_validate =␣
 ↪load_data()
```

[8]:
```
model = select_model(X_train, X_test, X_validate, y_train, y_test, y_validate)
```

```
0.753061224489796
```

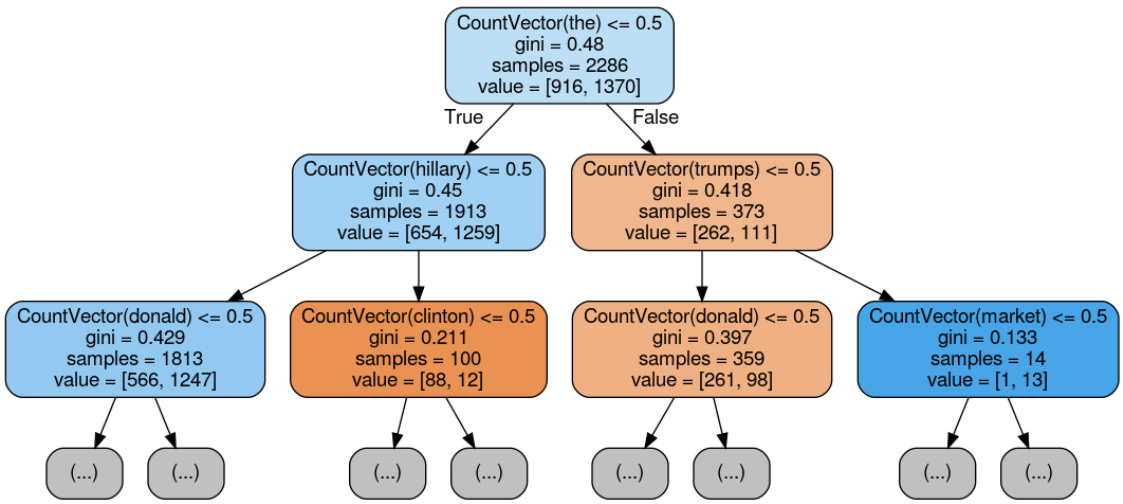
0.0.3 (c)

[10]:
```
# construct feature names appear on the graph, export the graph to the file␣
 ↪tree.dot
feature_names = list(map(lambda name: "CountVector(" + name + ")", vectorizer.
 ↪get_feature_names()))
dot = export_graphviz(model, max_depth = 2, feature_names = feature_names,␣
 ↪rounded = True, filled = True)
```

[11]:
```
import pydotplus
from IPython.display import Image
```

[12]:
```
graph = pydotplus.graph_from_dot_data(dot)
Image(graph.create_png())# load the graph
```

[12]:



0.0.4 (d)

[13]:
```
import numpy as np
from collections import Counter
```

[14]:
```
def entropy(num_real, num_fake):
    num_total = num_real + num_fake
    prob_fake, prob_real = num_fake / num_total, num_real / num_total
    return -(prob_fake * np.log2(prob_fake) + prob_real * np.log2(prob_real))
```

[15]:
```
def compute_information_gain(data, labels, word, word_to_index):
    # calculate the entropy before the split: H(Y)
    label_count = len(labels)
    freq = Counter(labels)
    old_entropy = entropy(freq[1], freq[0])

    # preparing the real and fake count for each split
```

```
    split_one, split_two = {"real": 0, "fake": 0}, {"real": 0, "fake": 0}

    # find the index of the word
    index = word_to_index[word]

    # iterate through each vector of title
    for i in range(len(data)):
        vec = data[i]
        # the vector belongs to split_one if the word doesn't exist, otherwise,␣
↪split_two
        if vec[index] <= 0.5:
            split = split_one
        else:
            split = split_two

        # increment the count according to wheather the title is fake or real
        if labels[i] == 1:
            split["real"] += 1
        elif labels[i] == 0:
            split["fake"] += 1

    prob_one, prob_two = sum(split_one.values()) / label_count, sum(split_two.
↪values()) / label_count
    # calculate the sum of entropy after spliting all titles: H(X|Y)
    new_entropy = (prob_one * entropy(split_one["real"], split_one["fake"])
                   + prob_two * entropy(split_two["real"], split_two["fake"]))
    # find information gain: IG
    return old_entropy - new_entropy
```

[16]:
```
compute_information_gain(X_train.toarray(), y_train.values, "the", vectorizer.
↪vocabulary_)
```

[16]:
```
0.052637477270443433
```

[17]:
```
compute_information_gain(X_train.toarray(), y_train.values, "hillary",␣
↪vectorizer.vocabulary_)
```

[17]:
```
0.04434458731584301
```

[18]:
```
compute_information_gain(X_train.toarray(), y_train.values, "trumps",␣
↪vectorizer.vocabulary_)
```

[18]:
```
0.04500636360104693
```

[19]:
```
compute_information_gain(X_train.toarray(), y_train.values, "donald",␣
↪vectorizer.vocabulary_)
```

[19]:
```
0.04939884792647942
```

[20]:
```
compute_information_gain(X_train.toarray(), y_train.values, "clinton",␣
↪vectorizer.vocabulary_)
```

[20]:
```
0.011983306127556492
```

[21]:
```
compute_information_gain(X_train.toarray(), y_train.values, "market",␣
↪vectorizer.vocabulary_)
```

[21]:
```
0.00027756163437764325
```

[ ]:
```

```

Question 3.3

October 4, 2019

0.0.1 (a) ~ (b)

[1]:
```
import numpy as np
```

Function which shuffles the list in uniformly random permutation

[2]:
```
def shuffle_data(data):
    order = np.random.permutation(len(data['X']))
    # shuffle X and t in the same order
    for i in range(0, len(order)):
        X = data['X'][order]
        t = data['t'][order]
    # return the shuffled array
    return {'X': X, 't': t}
```

Given the data, fold number, and the fold to be selected as the validation set, split the data with the given requirements. The function will distribute the fold as evenly as possible, the algorithm will is: 1. `len(data) % num_folds` folds have `size len(data) // num_folds + 1` 2. Other folds have size `len(data) // num_folds`

[3]:
```
def split_data(data, num_folds, fold):
    X, t = data['X'], data['t']
    length = len(X)
    first_batch = length % num_folds;

    # the longer fold is 1 unit longer than the short fold
    short_fold_len = length // num_folds
    long_fold_len = short_fold_len + 1

    # the fold can belong the short folds or the long folds
    if fold <= first_batch:
        start = (fold - 1) * long_fold_len
        end = start + long_fold_len
    else:
        start = (first_batch * long_fold_len) + (fold - first_batch - 1) *␣
↪short_fold_len
        end = start + short_fold_len

    # construct the dataset
    test_data = {'X': X[start: end], 't': t[start: end]}
```

```
    rest_data = {'X': np.concatenate((X[:start], X[end:])), 't': np.
↪concatenate((t[:start], t[end:]))}
    return test_data, rest_data
```

Train the model with **ridge regression** with weight $\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{t}$

[4]:
```
def train_model(data, lambd):
    X, t = data['X'], data['t']
    X_T = X.transpose()
    # using the equation
    return np.linalg.inv((X_T @ X) + (lambd * np.identity(len(X[0])))) @ X_T @ t
```

Given the data and the estimator $\hat{\mathbf{w}}_{MAP}$, predict the label vector of the data

[5]:
```
def predict(data, model):
    return data @ model
```

Calculate the average squared *error* loss based on model using the equation: $\|\mathbf{t} - \mathbf{X}\hat{\mathbf{w}}\|^2/n$

[6]:
```
def loss(data, model):
    X, t = data['X'], data['t']
    return (np.linalg.norm(t - (predict(X, model))) ** 2) / len(X)
```

[7]:
```
def cross_validation(data, num_folds, lambd_seq):
    data = shuffle_data(data)
    cv_error = np.empty(shape=(len(lambd_seq),))

    # for each lambda in the lambda sequence, calculate the average squared␣
↪error with cross validation
    for i in range(len(lambd_seq)):
        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
```

Gathering the train and test data

[8]:
```
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 = ',')}
```

0.0.2 (c)

Generate lambda sequence using the number evenly spaced in the interval (0.02, 1.5)

[9]:
```
lambd_seq = np.linspace(0.02, 1.5, num=50)
```

Calculate the train and test errors for each lambda in the lambda sequence

[10]:
```
data_train = shuffle_data(data_train)
train_error, test_error = np.empty(shape=(len(lambd_seq),)), np.
 ↪empty(shape=(len(lambd_seq),))
for i in range(len(lambd_seq)):
    lambd = lambd_seq[i]
    model = train_model(data_train, lambd)
    train_error[i], test_error[i] = loss(data_train, model), loss(data_test,␣
 ↪model)
```

0.0.3 (d)

Calculate five and ten fold cross validation errors

[11]:
```
five_fold_cv_error = cross_validation(data_train, 5, lambd_seq)
```

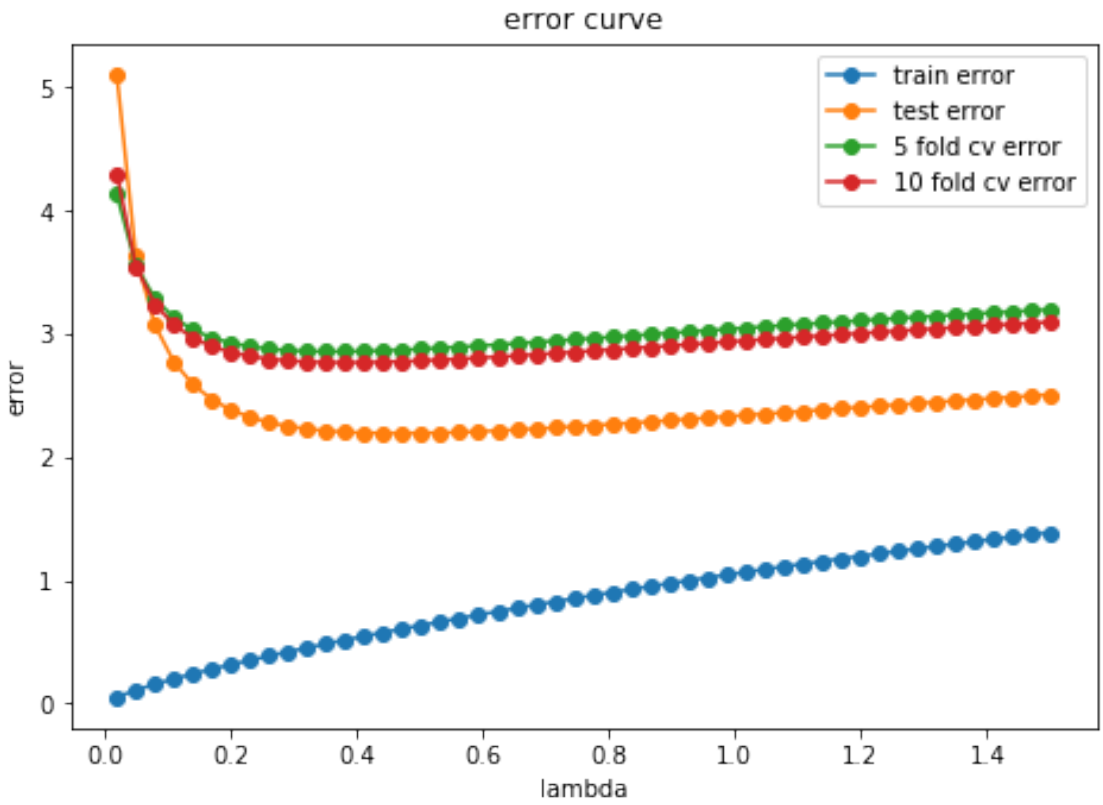
[12]:
```
ten_fold_cv_error = cross_validation(data_train, 10, lambd_seq)
```

Plot test, train, five, ten fold cross validation errors for each lambda

[13]:
```
from matplotlib import pyplot as plt
```

[28]:
```
%matplotlib inline
x = np.array(lambd_seq)
y_train, y_test, y_five_cv, y_ten_cv = train_error, test_error,␣
 ↪five_fold_cv_error, ten_fold_cv_error
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.plot(x, y_train, '-o', label="train error")
ax.plot(x, y_test, '-o', label="test error")
ax.plot(x, y_five_cv, '-o', label="5 fold cv error")
ax.plot(x, y_ten_cv, '-o', label="10 fold cv error")
ax.set_title("error curve")
ax.set_xlabel("lambda")
ax.set_ylabel("error")
plt.legend()
```

[28]:
```
<matplotlib.legend.Legend at 0x7f36a72c8a10>
```



[20]:
```
ten_fold_min_lambd = lambd_seq[np.argmin(y_ten_cv)];
```

[21]:
```
five_fold_min_lambd = lambd_seq[np.argmin(y_five_cv)];
```

[22]:
```
min_lambd = min(ten_fold_min_lambd, five_fold_min_lambd)
```

[23]:
```
min_lambd
```

[23]:
```
0.3522448979591837
```

0.0.4 Comment

The lambda proposed by the plot is the mininum between the argmin of 5 and 10 fold cross validation error which is about 0.352. The argmin of train and test error are not proposed since they don't have a valid validation set.

According to the plot, the model trained with training data directly is overfitting. Since the loss of prediciting test data with this model is much larger than the training error.

The trainning error is increasing as lambda increases, since as lambda increases, the model is affected more by the regularizer. Initially, the error should be the minimum average squared error, but the regularizer penalize the the loss more and more as the lambda increases, thats why the training error increases.

The cross validation method reduces the extend of over fitting of the model. It can be shown by the fact that the shape of the cross validation error curve is very similar to the actual test error curve.