pdfpages

Homework 1 CSC 311

Maria Quintero

October 4, 2019

Problem 1 (a)

since X is uniformly distributed and so is y, and x and y are independent, E(X) = E(Y) and E(X^2) = $E(Y^2)$.
Firstly,

E(x) = $\int_a^b \frac{x}{b-a}dx$, since x is uniformly distributed. a = 0 and b =1, since x is a random variable between [0,1]

$= \int_0^1 \frac{x}{1}dx$

$= [\frac{x}{2}]_0^1$

$= \frac{1}{2}$

E(X^2) =$\int_a^b x^2 dx$, since x is uniformly distributed. a = 0 and b =1, since x is a random variable between [0,1]

$= \int_0^1 \frac{x}{1}^2 dx$

$= [\frac{x}{3}]_0^1$

$= \frac{1}{3}$

Hence, E(Z) can be solved as ,

E(Z) = E((X - Y)^2)

= E(x$^2 - 2XY - Y^2$)

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

$= \frac{1}{3} - 2(\frac{1}{2})(\frac{1}{2}) + \frac{1}{3}$, since x and y are independet E(XY) = E(x)E(Y)

$= \frac{1}{6}$

first, let's solve E(Z^2)

We know that the moment generating function for the uniform distribution is

U$_n$ =b$^{n+1} - a^{n+1}$ $_{\overline{(n+1)(b-a)}}$, where a = 0 and b = 1 because X is random variable in the interval [0,1] and n is the moment we want to solve ie E(X^n) = U_n

hence,

E(X^4) =1$^{4+1} - 0^{4+1}$ $_{\overline{(4+1)(1-0)}}$ $= \frac{1}{5}$ = E(Y^4)

E(X^3) =1$^{3+1} - 0^{3+1}$ $_{\overline{(3+1)(1-0)}}$ $= \frac{1}{4}$ = E(Y^3)

E(X^2) =1$_{\overline{3}}$ = E(Y^2)$asshownpreviously.$

Thus,

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

= E(X$^4 - 4X^3Y + 6X^2Y^2 - 4XY^3 + Y^4$)

= E(X^4) $- 4E(X^3)E(Y) + 6E(X^2)E(Y^2) - 4E(Y^3)E(X) + E(Y^4)$

$= \frac{1}{5} - 4(\frac{1}{4})(\frac{1}{2}) + 6(\frac{1}{3})(\frac{1}{3}) - 4(\frac{1}{4})(\frac{1}{2}) + \frac{1}{5} = \frac{1}{15}$

thus,

Var(Z) = E(Z^2) $- E(Z)^2$

$= \frac{1}{15} - \frac{1}{6}^2$

$= \frac{1}{15}\frac{1}{36} = \frac{7}{180}$

b)

E(R) = $E(Z_1 + Z_2 + ... + Z_d)$

= d(E(Z)), since each Z$_i$, *where* i $\in \mathbb{N}0 < i \leq d$ are independent from one another hence we add E(Z) d times.

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

Var(R) = Var(Z$_1$ + Z_2 + .. + Z_d)

= d(Var(Z)) for the same reasoning above and if all Z$_i$, *where* i $\in \mathbb{N}0 < i \leq d$ are independent from one another then we are able to add Var(Z) d times

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

c)

First lets compute the maximum squared euclidean distance. Since R = Z$_1 + Z_2 + ... + Z_d, andZ_i = (X_i - Y_i)^2 wherei$$\in \mathbb{N}0 <$

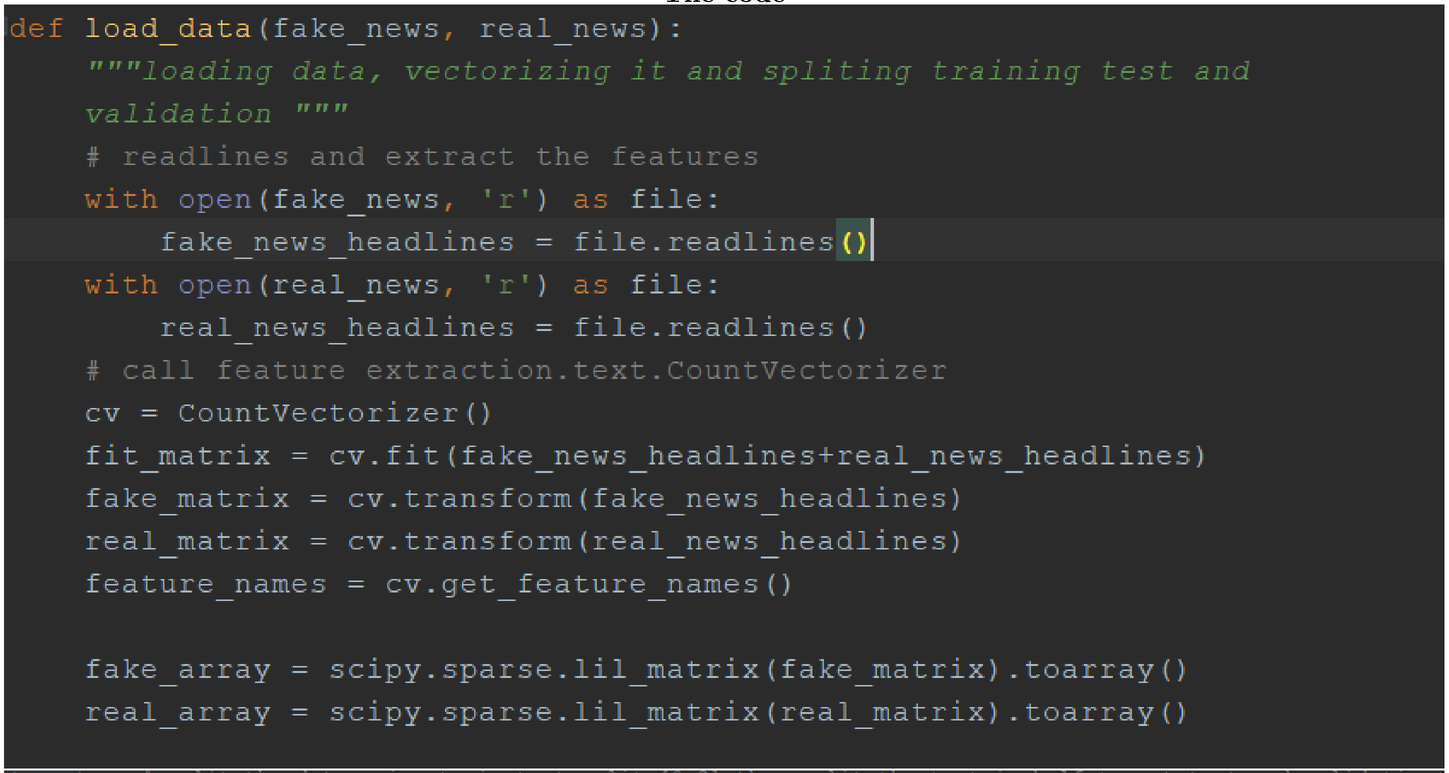
$i \leq d$, then we know that the max distance between $\mathrm{X}_i and Y_i is X_i = 1 and Y_i = 0 (or vice versa). Hence,$

$$\text{max squared distance} = \sqrt{(1-0)^2 + (1_0)^2 + ... + (1-0)^2}^2 where we add (1-0)^2 d times, hence$$
$$= \sqrt{(}d)^2$$
$$= \mathrm{d}$$

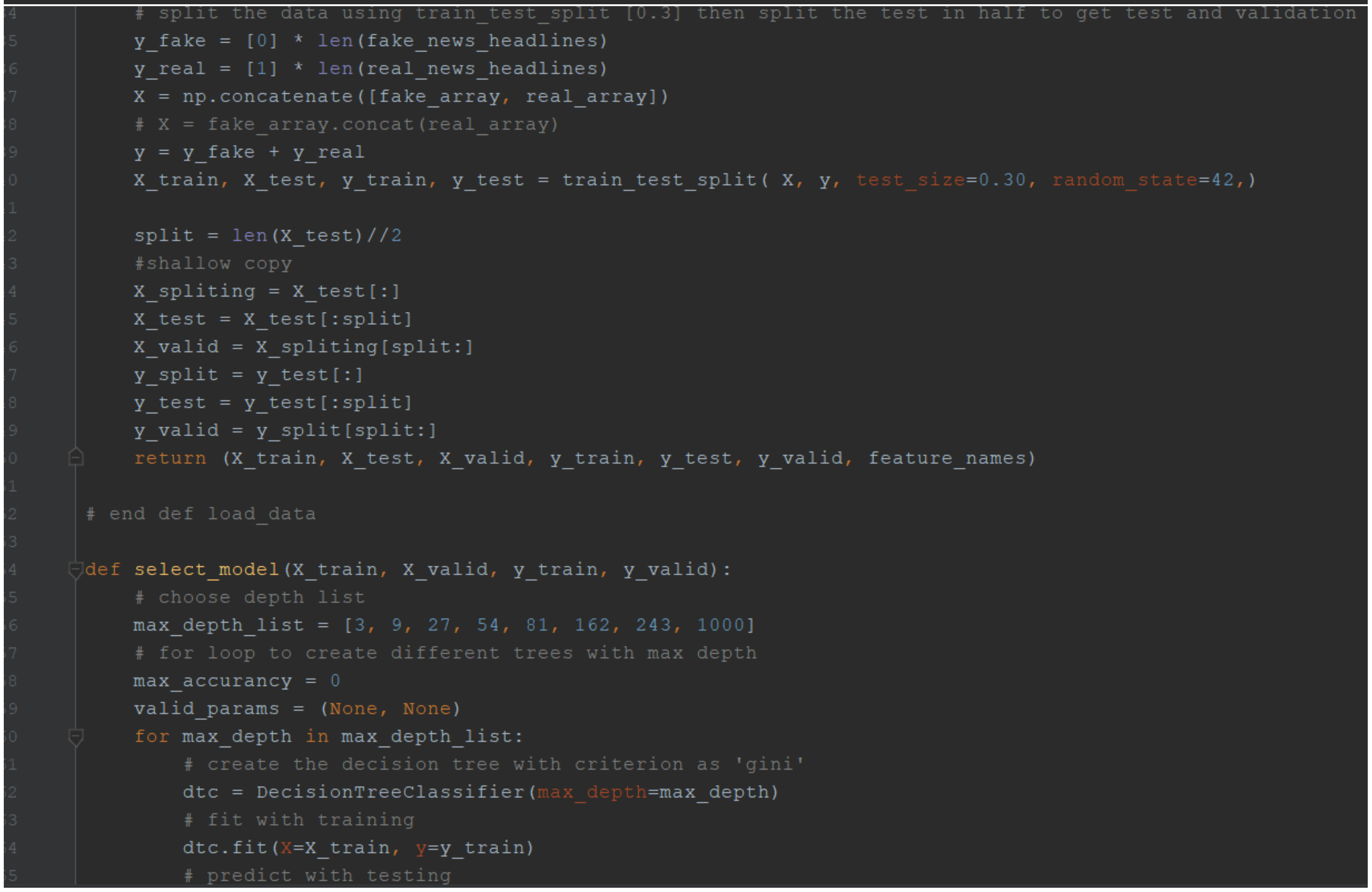
We solve the variance to be $\frac{7d}{180}$ hence the standard deviation is $\sqrt{\frac{7d}{180}}$. Since when d is large ie. in high dimension, since it belongs $\mathbb{O}$(d) it grows at a faster rate than $\sqrt{\frac{7d}{180}} \in \mathbb{O}(\sqrt{d})$. Thus the claim that most points are far away and approximately the same distance is true Problem 3 (a) P(t—X,w) = ?

Problem 2 2.a

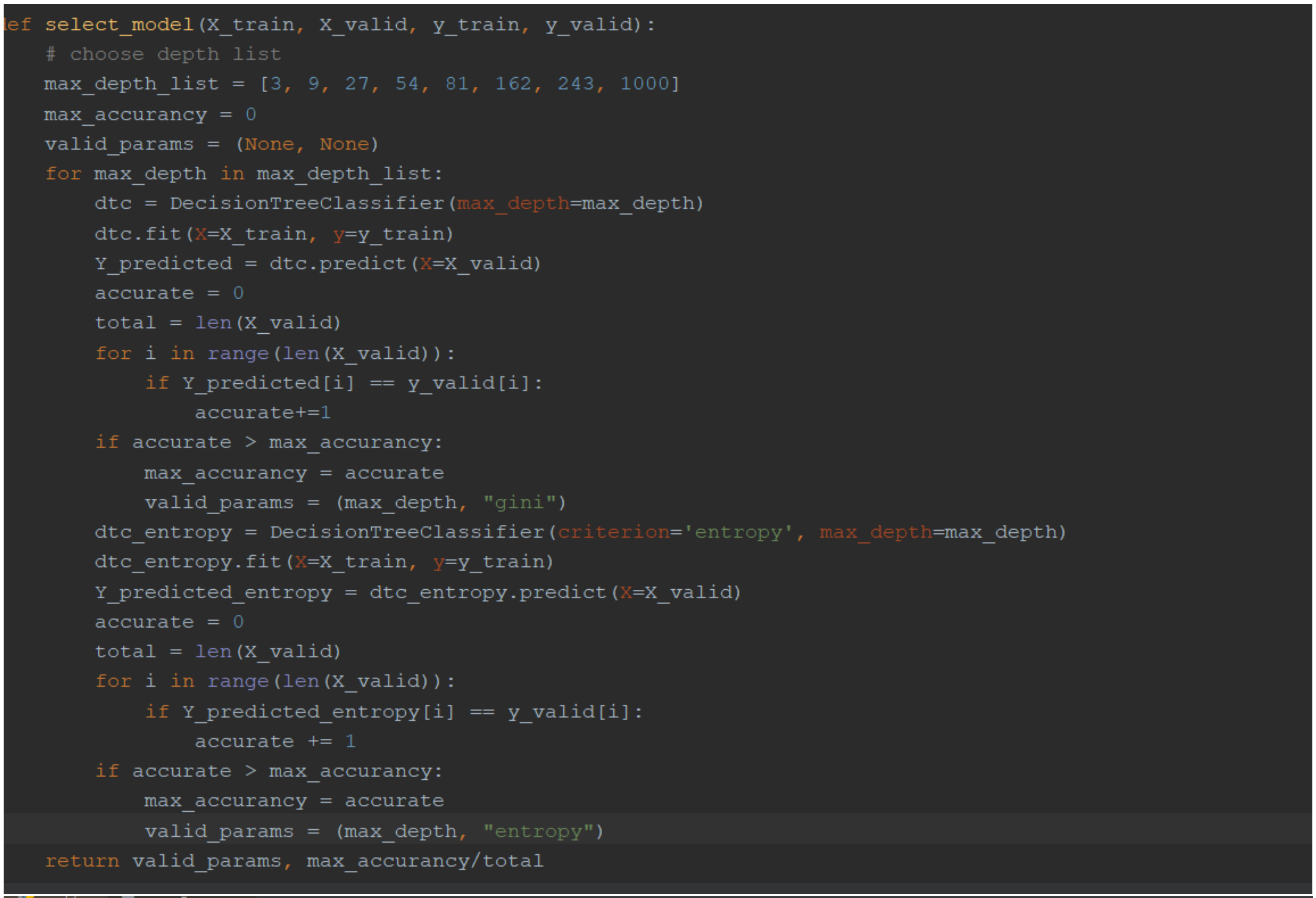
The code



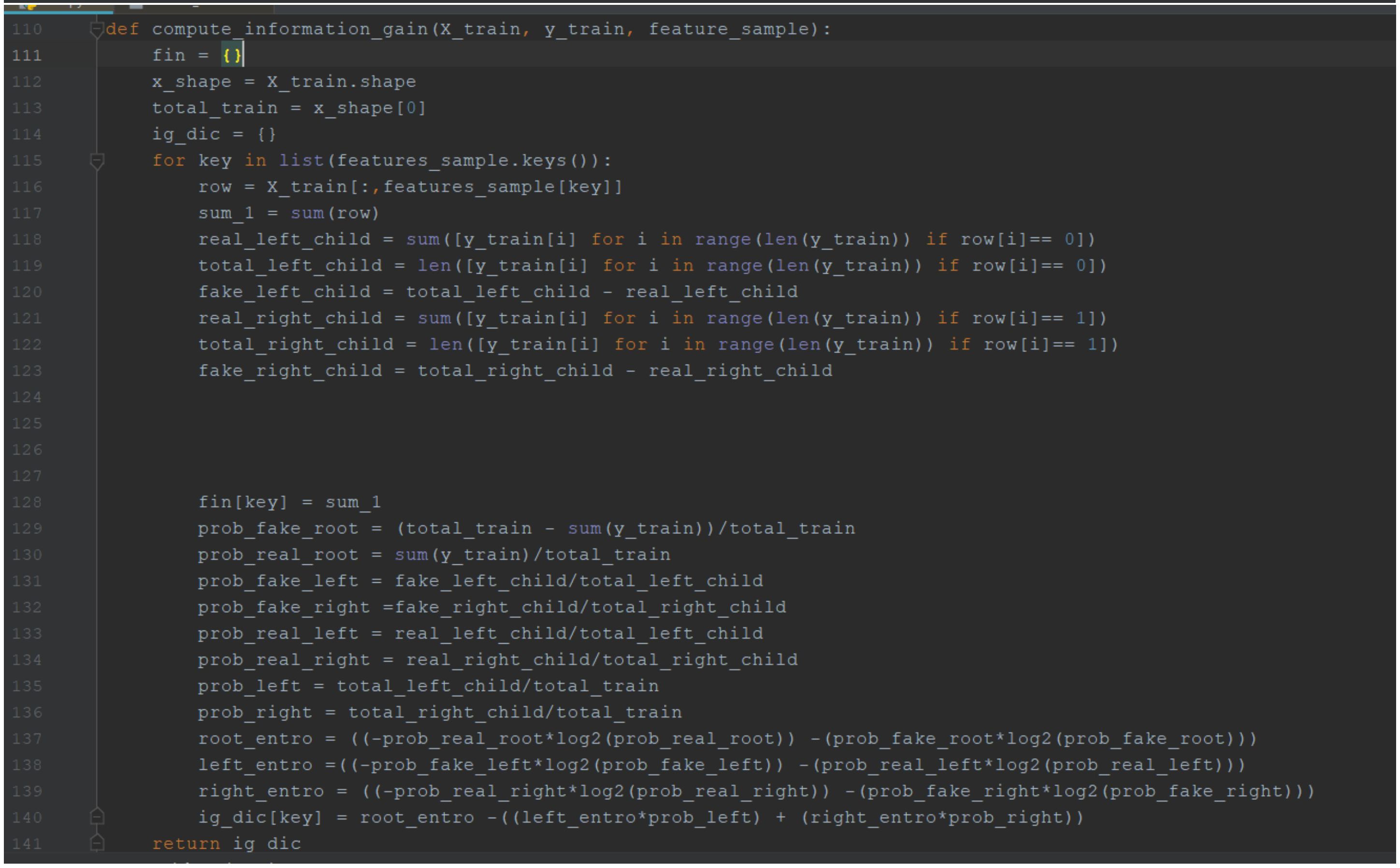
```python
def load_data(fake_news, real_news):
    """loading data, vectorizing it and spliting training test and
    validation """
    # readlines and extract the features
    with open(fake_news, 'r') as file:
        fake_news_headlines = file.readlines()
    with open(real_news, 'r') as file:
        real_news_headlines = file.readlines()
    # call feature extraction.text.CountVectorizer
    cv = CountVectorizer()
    fit_matrix = cv.fit(fake_news_headlines+real_news_headlines)
    fake_matrix = cv.transform(fake_news_headlines)
    real_matrix = cv.transform(real_news_headlines)
    feature_names = cv.get_feature_names()

    fake_array = scipy.sparse.lil_matrix(fake_matrix).toarray()
    real_array = scipy.sparse.lil_matrix(real_matrix).toarray()
```



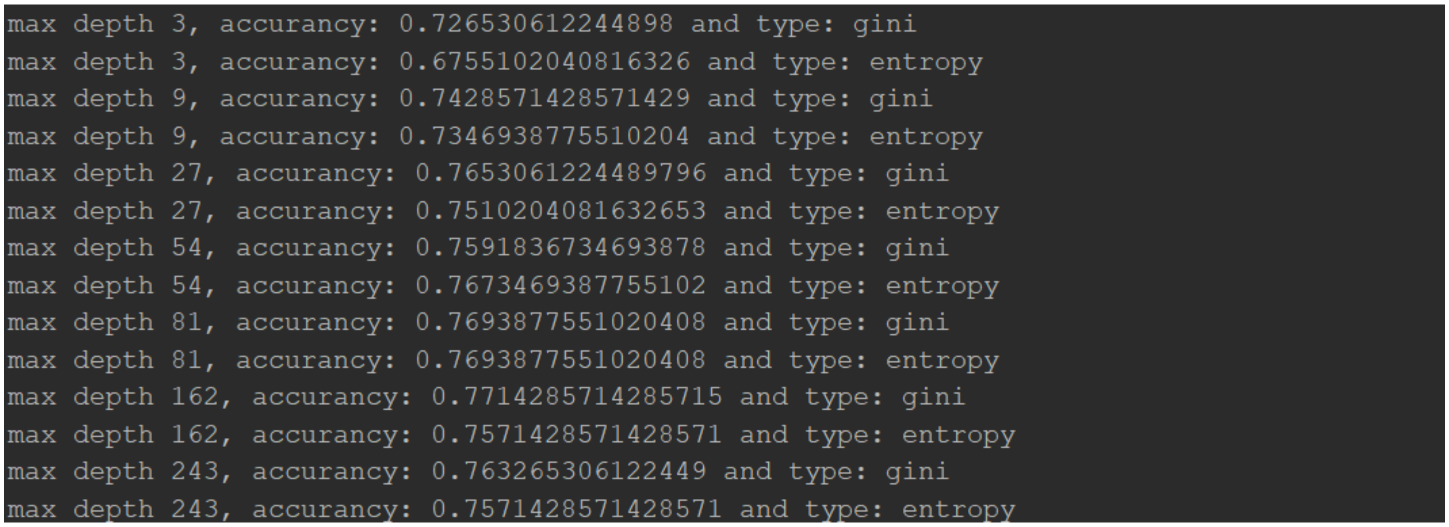
```python
    # split the data using train_test_split [0.3] then split the test in half to get test and validation
    y_fake = [0] * len(fake_news_headlines)
    y_real = [1] * len(real_news_headlines)
    X = np.concatenate([fake_array, real_array])
    # X = fake_array.concat(real_array)
    y = y_fake + y_real
    X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.30, random_state=42,)

    split = len(X_test)//2
    #shallow copy
    X_spliting = X_test[:]
    X_test = X_test[:split]
    X_valid = X_spliting[split:]
    y_split = y_test[:]
    y_test = y_test[:split]
    y_valid = y_split[split:]
    return (X_train, X_test, X_valid, y_train, y_test, y_valid, feature_names)

# end def load_data

def select_model(X_train, X_valid, y_train, y_valid):
    # choose depth list
    max_depth_list = [3, 9, 27, 54, 81, 162, 243, 1000]
    # for loop to create different trees with max depth
    max_accurancy = 0
    valid_params = (None, None)
    for max_depth in max_depth_list:
        # create the decision tree with criterion as 'gini'
        dtc = DecisionTreeClassifier(max_depth=max_depth)
        # fit with training
        dtc.fit(X=X_train, y=y_train)
        # predict with testing
```



```
lef select_model(X_train, X_valid, y_train, y_valid):
    # choose depth list
    max_depth_list = [3, 9, 27, 54, 81, 162, 243, 1000]
    max_accurancy = 0
    valid_params = (None, None)
    for max_depth in max_depth_list:
        dtc = DecisionTreeClassifier(max_depth=max_depth)
        dtc.fit(X=X_train, y=y_train)
        Y_predicted = dtc.predict(X=X_valid)
        accurate = 0
        total = len(X_valid)
        for i in range(len(X_valid)):
            if Y_predicted[i] == y_valid[i]:
                accurate+=1
        if accurate > max_accurancy:
            max_accurancy = accurate
            valid_params = (max_depth, "gini")
        dtc_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth)
        dtc_entropy.fit(X=X_train, y=y_train)
        Y_predicted_entropy = dtc_entropy.predict(X=X_valid)
        accurate = 0
        total = len(X_valid)
        for i in range(len(X_valid)):
            if Y_predicted_entropy[i] == y_valid[i]:
                accurate += 1
        if accurate > max_accurancy:
            max_accurancy = accurate
            valid_params = (max_depth, "entropy")
    return valid_params, max_accurancy/total
```

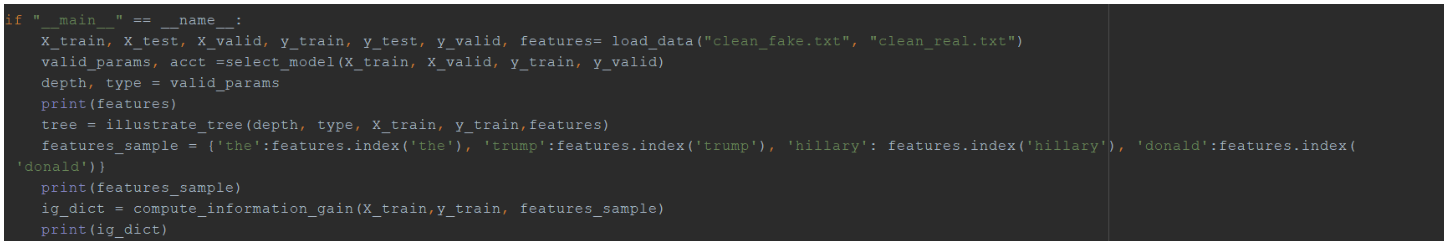


```
def compute_information_gain(X_train, y_train, feature_sample):
    fin = {}
    x_shape = X_train.shape
    total_train = x_shape[0]
    ig_dic = {}
    for key in list(features_sample.keys()):
        row = X_train[:,features_sample[key]]
        sum_1 = sum(row)
        real_left_child = sum([y_train[i] for i in range(len(y_train)) if row[i]== 0])
        total_left_child = len([y_train[i] for i in range(len(y_train)) if row[i]== 0])
        fake_left_child = total_left_child - real_left_child
        real_right_child = sum([y_train[i] for i in range(len(y_train)) if row[i]== 1])
        total_right_child = len([y_train[i] for i in range(len(y_train)) if row[i]== 1])
        fake_right_child = total_right_child - real_right_child




        fin[key] = sum_1
        prob_fake_root = (total_train - sum(y_train))/total_train
        prob_real_root = sum(y_train)/total_train
        prob_fake_left = fake_left_child/total_left_child
        prob_fake_right =fake_right_child/total_right_child
        prob_real_left = real_left_child/total_left_child
        prob_real_right = real_right_child/total_right_child
        prob_left = total_left_child/total_train
        prob_right = total_right_child/total_train
        root_entro = ((-prob_real_root*log2(prob_real_root)) -(prob_fake_root*log2(prob_fake_root)))
        left_entro =((-prob_fake_left*log2(prob_fake_left)) -(prob_real_left*log2(prob_real_left)))
        right_entro = ((-prob_real_right*log2(prob_real_right)) -(prob_fake_right*log2(prob_fake_right)))
        ig_dic[key] = root_entro -((left_entro*prob_left) + (right_entro*prob_right))
    return ig_dic
```



```
max depth 3, accurancy: 0.726530612244898 and type: gini
max depth 3, accurancy: 0.6755102040816326 and type: entropy
max depth 9, accurancy: 0.7428571428571429 and type: gini
max depth 9, accurancy: 0.7346938775510204 and type: entropy
max depth 27, accurancy: 0.7653061224489796 and type: gini
max depth 27, accurancy: 0.7510204081632653 and type: entropy
max depth 54, accurancy: 0.7591836734693878 and type: gini
max depth 54, accurancy: 0.7673469387755102 and type: entropy
max depth 81, accurancy: 0.7693877551020408 and type: gini
max depth 81, accurancy: 0.7693877551020408 and type: entropy
max depth 162, accurancy: 0.7714285714285715 and type: gini
max depth 162, accurancy: 0.7571428571428571 and type: entropy
max depth 243, accurancy: 0.763265306122449 and type: gini
max depth 243, accurancy: 0.7571428571428571 and type: entropy
```

Figure 1: 2b) **Output**



```
if "__main__" == __name__:
    X_train, X_test, X_valid, y_train, y_test, y_valid, features= load_data("clean_fake.txt", "clean_real.txt")
    valid_params, acct =select_model(X_train, X_valid, y_train, y_valid)
    depth, type = valid_params
    print(features)
    tree = illustrate_tree(depth, type, X_train, y_train,features)
    features_sample = {'the':features.index('the'), 'trump':features.index('trump'), 'hillary': features.index('hillary'), 'donald':features.index(
 'donald')}
    print(features_sample)
    ig_dict = compute_information_gain(X_train,y_train, features_sample)
    print(ig_dict)
```

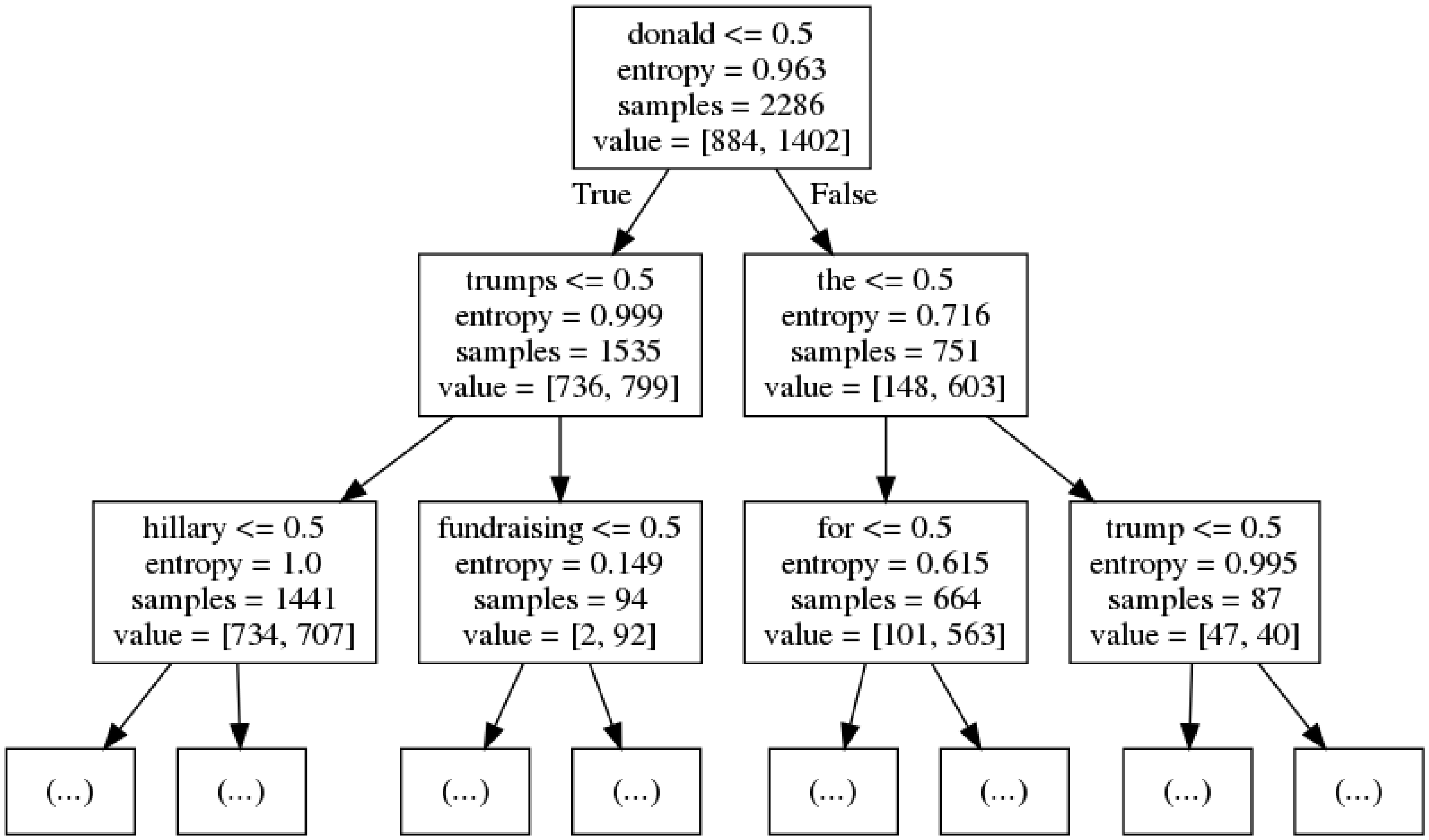
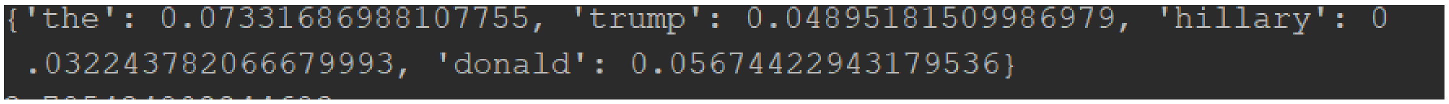


Figure 2: 2c)



```
{'the': 0.07331686988107755, 'trump': 0.04895181509986979, 'hillary': 0
 .03224378206667999З, 'donald': 0.05674422943179536}
```

Figure 3: Computation gain 2d

Let $X \in \mathbb{R}^{n \times m}$ where $n \geq m$ and $t \in \mathbb{R}^n$

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

Hence we know the joint density to be

$$= \prod_{i=1}^{n} \left(\frac{1}{2\pi^{n/2} \det(\sigma^2 I)^{1/2}} \exp\left\{ -\frac{1}{2} (t - Xw)^T (\sigma^2 I)^{-1} (t - Xw) \right\} \right)^{*}$$

*from the prelims pdf.

Let $L(w)$ denote what is above.

Hence

$$\log(L(w)) = \log\left(\prod_{i=1}^{n} \left(\frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left\{ \frac{-1}{2} (t - Xw)^T (\sigma^2 I)^{-1} (t - Xw) \right\} \right) \right)$$

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

$$-\frac{1}{2}(t - Xw)^T(\sigma^2 I)^{-1}(t - Xw)$$

$$= \sum_{i=1}^{n} -\frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi - \frac{1}{2\sigma^2}(t - Xw)^T (t - Xw)$$

Next, we find the derivative in respect to W and set to 0

$$1 = \frac{\partial L}{\partial W}\left(\sum_{i=1}^{n} -\frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi - \frac{1}{2\sigma^2}(t_i - X_i w)^T (t_i - X_i w_i)\right)$$

$$= \frac{\partial L}{\partial W}\left(-\frac{1}{2\sigma^2}(t_i - Xw)^T(t - Xw)\right)$$

$$= \frac{\partial L}{\partial W}\left(\frac{-1}{2\sigma^2}\left(t^T - w^T X^T\right)(t - Xw)\right)$$

[illegible]

$$= \frac{\partial L}{\partial W}\left(\frac{-1}{2\sigma^2}\left[t^T t - t^T X W - W^T X^T t + W^T X^T X W\right]\right.$$

$$= \frac{\partial L}{\partial W}\left(\frac{-1}{2\sigma^2}\left[t^T t - 2t^T X W \overset{*}{+} W^T X^T X W\right]\right.$$

* we know that $t^T X W = W^T X^T t$ because they are symmtric $(t^T X W)^T = W^T X^T t$.

$$= \frac{\partial L}{\partial W}\left(\frac{-1}{2\sigma^2}\left[t^T t - 2t^T X W + W^T X^T X W\right)\right.$$

$$0 = \frac{t^T X}{\sigma^2} - 2\left(\frac{1}{2\sigma^2}\right) X^T X W)$$

$$0 = \frac{1}{\sigma^2}\left(X^T t - 2\left(\frac{1}{2}\right) X^T X W\right)$$

$$0 = \frac{1}{\sigma^2}\left(X^T t - X^T X W\right)$$

$$X^T X W = X^T t$$

$$\vec{W} = (X^T X)^{-1} X^T t \quad \Rightarrow \text{from prelims}$$

$$E(\tilde{W}) = E\left((X^T X)^{-1} X^T t \right)$$

$$= (X^T X)^{-1} X^T E(t) \rightarrow \text{since } t \mid X, w \sim N(Xw, \sigma^2 I)$$

$$= (X^T X)^{-1} X^T (Xw)$$

$$= w$$

Hence we have from the prelims

if $X \sim N(\mu, \Sigma)$ then $AX \sim (A\mu, A\Sigma A^T)$

Then

$$t \sim N(Xw, \sigma^2 I)$$

$$At \sim N(AXw, A\sigma^2 I A^T)$$

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

[illegible]

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

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

since $\tilde{W} = (X^T X)^{-1} X^T t$

Thus covariance $(\tilde{w})$ by the formula is

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

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

and $w \perp X$, $w \sim N(0, \tau^2 I)$

Since $w \perp X$, we will do a similar solution to 3.1 a) and multiply the new term

Let $L(w)$ represent the likehood function

$$L(w) = \underset{w}{\text{argmax}} \prod_{i=1}^{n} \left(\frac{1}{2\pi^{n/2} \det(\sigma^2 I)^{1/2}} \exp\left\{ -\frac{1}{2} (t - Xw)^T (\sigma^2 I)^{-1} (t - Xw) \right\} \right)$$

$$\left(\frac{1}{2\pi^{m/2} \det(\tau^2 I)} \exp\left\{ -\frac{1}{2} (w - 0)^T (\tau^2 I)^{-1} (w - 0) \right\} \right)$$

taking log we get

$$\log L(w) = \sum_{i=1}^{n} \Big(-\frac{n}{2} \log(2\pi) - \frac{n}{2} \log[\sigma^2]$$

* constant! in respect to W

$$-\frac{1}{2}(t - Xw)^T (\theta^2 I)^{-1} (t - Xw)$$

$$-\frac{m}{2} \log(2\pi) - \frac{m}{2} \log(\tau^2 I) -$$

$$\frac{1}{2}(w)^T (\tau^2 I)^{-1} (w)$$

$$\frac{\partial L}{\partial w} = -\frac{1}{2}(-X)^T (\theta^2 I)(t - Xw)$$

$$+ \Big(-\frac{1}{2}(t - Xw)^T (\theta^2 I)(-X)$$

$$-\frac{1}{2}(\tau^2 I)^{-1}(w) - \frac{1}{2} w^T (\tau^2 I)$$

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

$$- \frac{1}{2\tau^2}(w)w^T \; *$$

* since we derived this part on part 3.1 I showed the conclusion of both.

$$0 = \frac{1}{2}\left[\frac{1}{\sigma^2}\left[(X)^T(t - Xw) + (t - Xw)^T(X)\right]\right.$$

$$\left. - \frac{2}{\tau^2}(w)\right]$$

$$0 = \left[\frac{1}{\sigma^2}\left[(X)^T(t - Xw) + (X)^T(t - Xw)\right]\right.$$

$$\left. - \frac{2}{\tau^2}(w)\right]$$

$$0 = \frac{2}{\sigma^2}[X^T t - X^T X w] - \frac{2}{\tau^2}(w)$$

$$0 = \frac{I}{\sigma^2}[X^T t - X^T X w] - \frac{1}{\tau^2}(w)$$

$$\frac{1}{\tau^2}(w) + X^T X w\left(\frac{1}{\sigma^2}\right) = \frac{I}{\sigma^2}[X^T t]$$

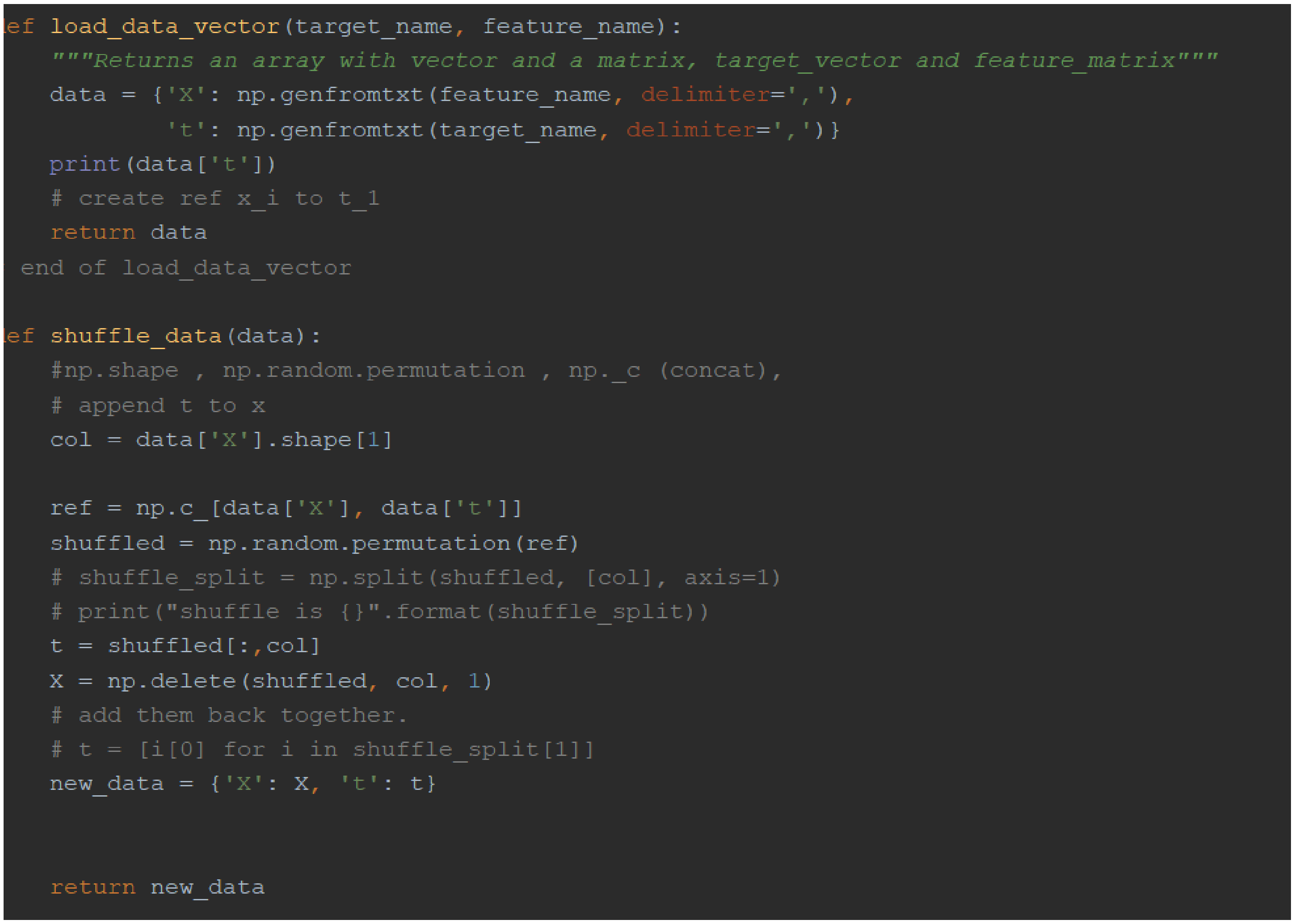
$$w\left[\frac{I}{\tau^2} + X^T X\left(\frac{I}{\sigma^2}\right)\right] = \frac{I}{\sigma^2}[X^T t]$$

$$w\left[\frac{\sigma^2 I}{\tau^2} + X^T X\right] = [X^T t]$$

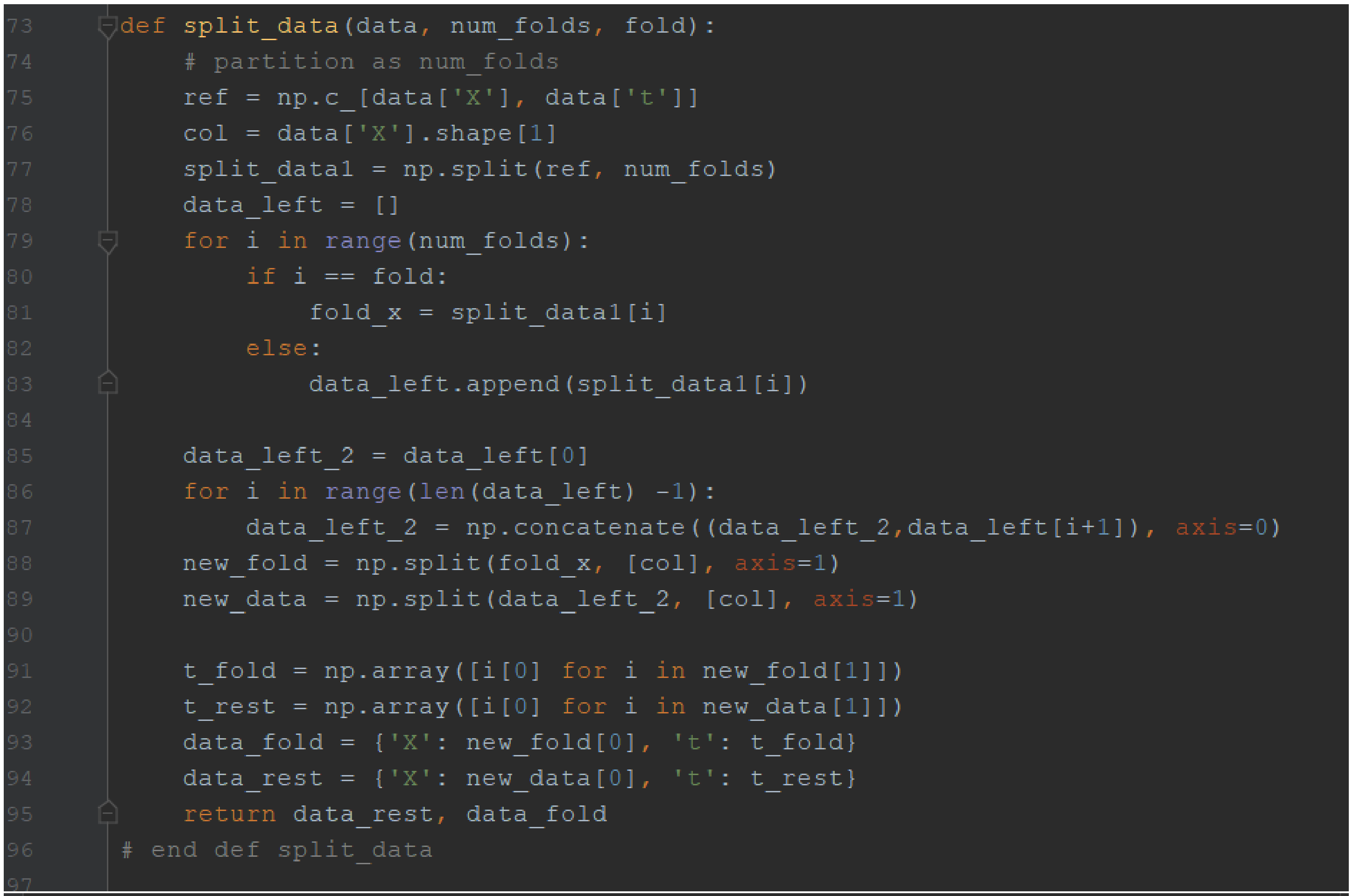
$$w_{MAP} = \left[X^T X + \frac{\sigma^2}{\tau^2}\right] [X^T t]$$

$$\tilde{w}_{MAP} = \left[X^T X + \lambda I\right)^{-1} [X^T t]$$

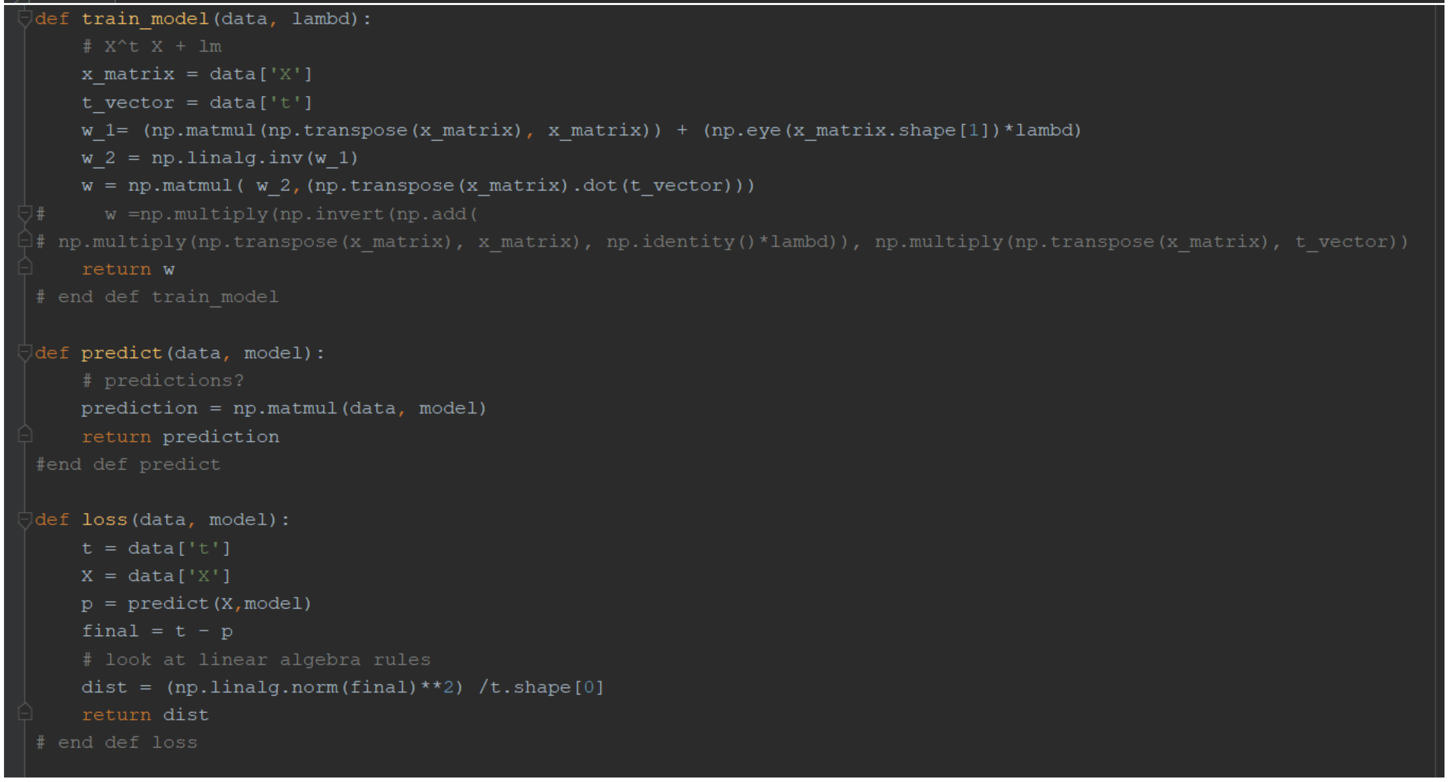
3.C



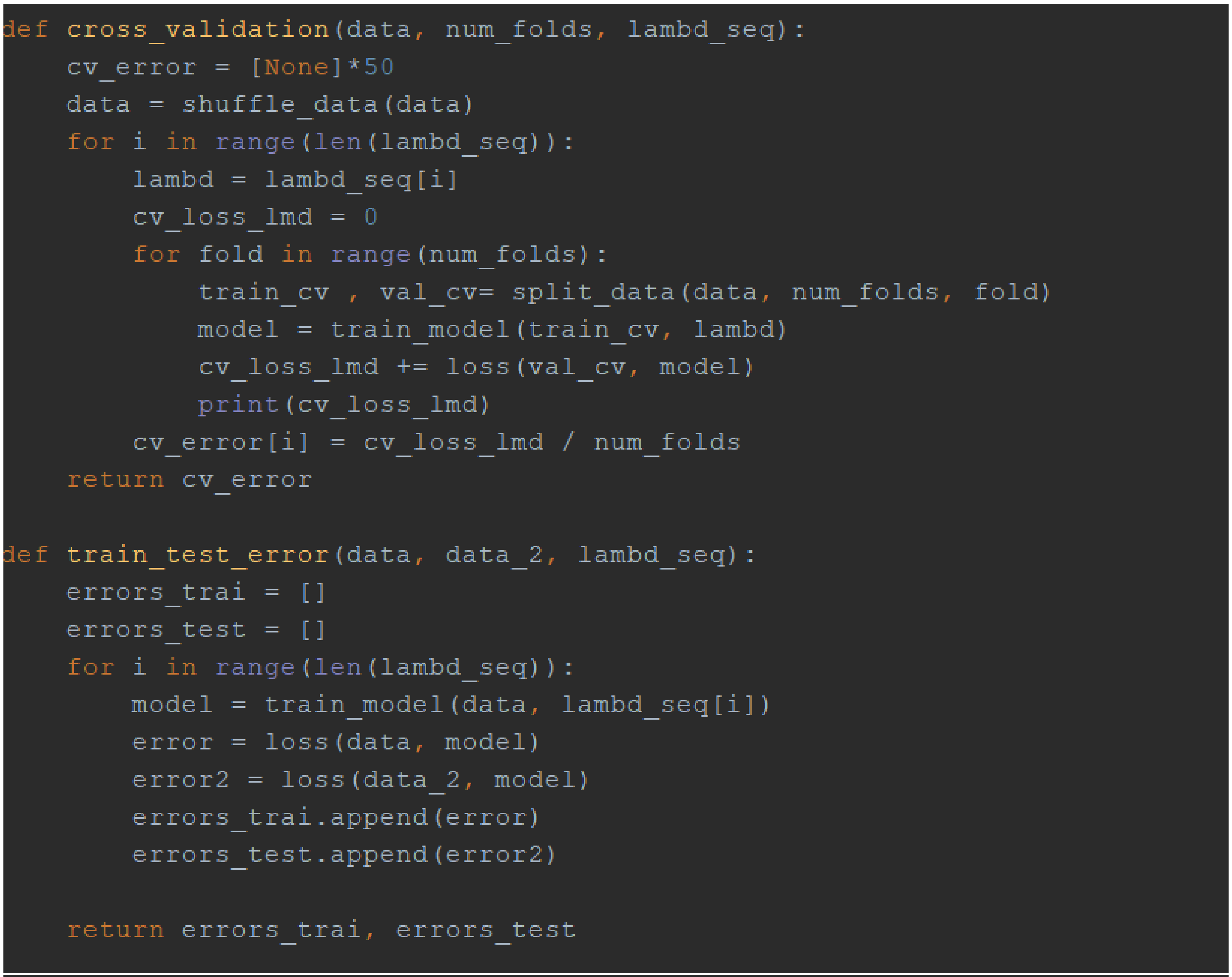
```
ef load_data_vector(target_name, feature_name):
   """Returns an array with vector and a matrix, target_vector and feature_matrix"""
   data = {'X': np.genfromtxt(feature_name, delimiter=','),
           't': np.genfromtxt(target_name, delimiter=',')}
   print(data['t'])
   # create ref x_i to t_1
   return data
 end of load_data_vector

ef shuffle_data(data):
   #np.shape , np.random.permutation , np._c (concat),
   # append t to x
   col = data['X'].shape[1]

   ref = np.c_[data['X'], data['t']]
   shuffled = np.random.permutation(ref)
   # shuffle_split = np.split(shuffled, [col], axis=1)
   # print("shuffle is {}".format(shuffle_split))
   t = shuffled[:,col]
   X = np.delete(shuffled, col, 1)
   # add them back together.
   # t = [i[0] for i in shuffle_split[1]]
   new_data = {'X': X, 't': t}


   return new_data
```



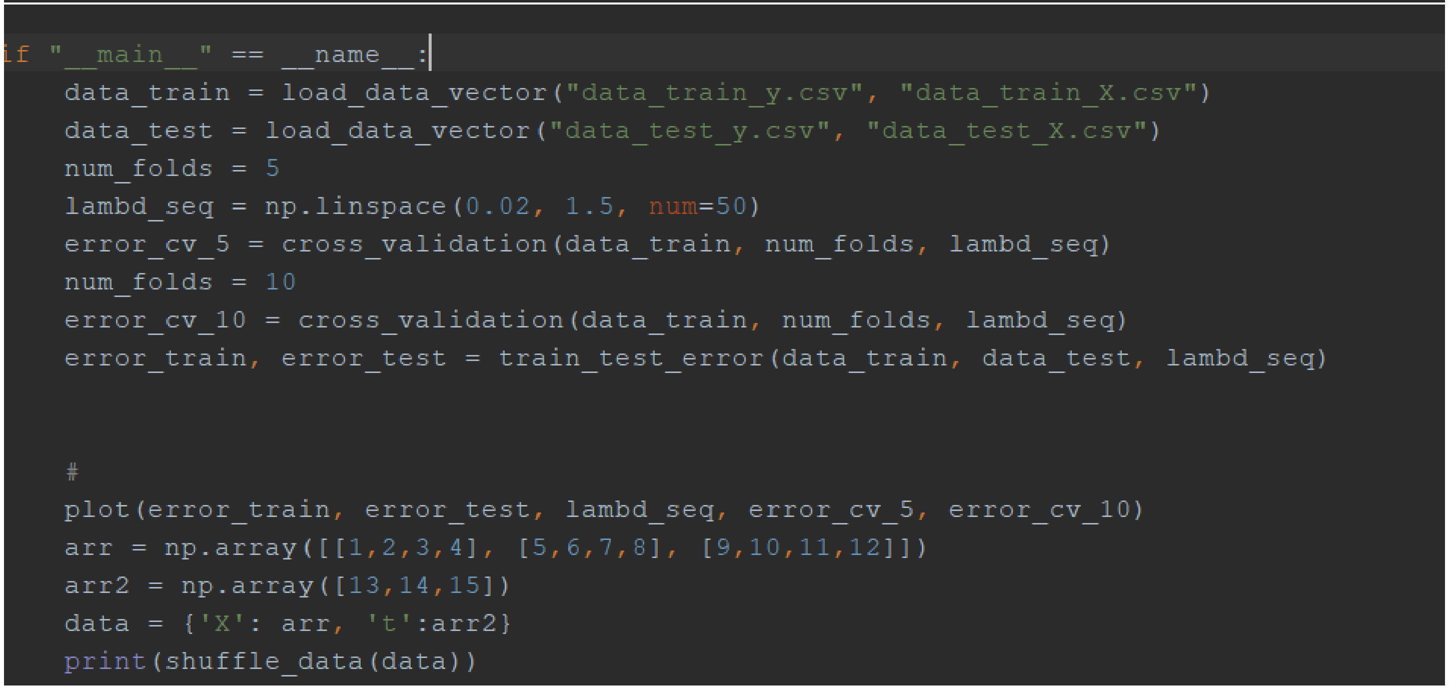
```python
def split_data(data, num_folds, fold):
    # partition as num_folds
    ref = np.c_[data['X'], data['t']]
    col = data['X'].shape[1]
    split_data1 = np.split(ref, num_folds)
    data_left = []
    for i in range(num_folds):
        if i == fold:
            fold_x = split_data1[i]
        else:
            data_left.append(split_data1[i])

    data_left_2 = data_left[0]
    for i in range(len(data_left) -1):
        data_left_2 = np.concatenate((data_left_2,data_left[i+1]), axis=0)
    new_fold = np.split(fold_x, [col], axis=1)
    new_data = np.split(data_left_2, [col], axis=1)

    t_fold = np.array([i[0] for i in new_fold[1]])
    t_rest = np.array([i[0] for i in new_data[1]])
    data_fold = {'X': new_fold[0], 't': t_fold}
    data_rest = {'X': new_data[0], 't': t_rest}
    return data_rest, data_fold
# end def split_data
```



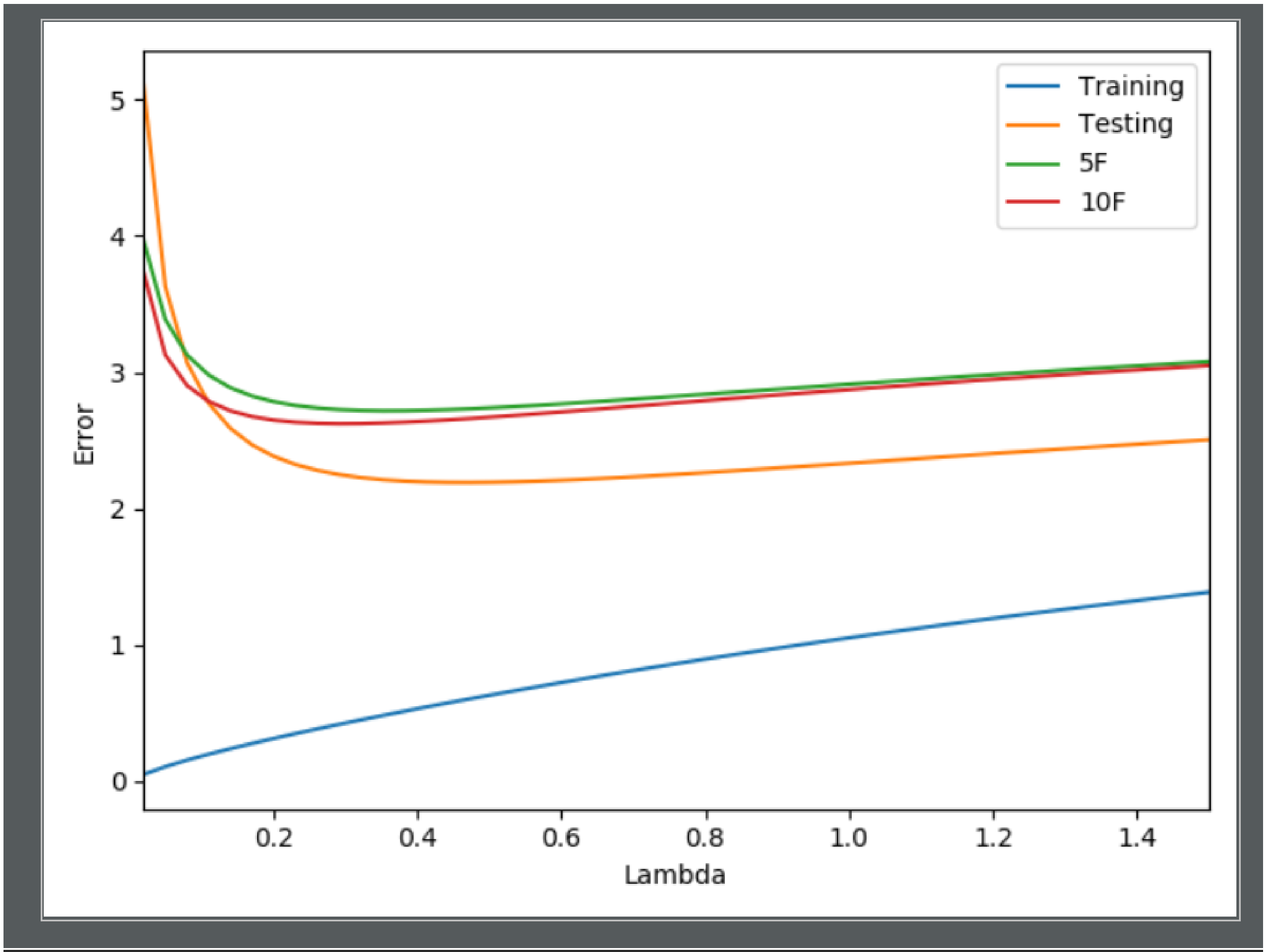
```python
def train_model(data, lambd):
    # X^t X + lm
    x_matrix = data['X']
    t_vector = data['t']
    w_1= (np.matmul(np.transpose(x_matrix), x_matrix)) + (np.eye(x_matrix.shape[1])*lambd)
    w_2 = np.linalg.inv(w_1)
    w = np.matmul( w_2,(np.transpose(x_matrix).dot(t_vector)))
#     w =np.multiply(np.invert(np.add(
# np.multiply(np.transpose(x_matrix), x_matrix), np.identity()*lambd)), np.multiply(np.transpose(x_matrix), t_vector))
    return w
# end def train_model

def predict(data, model):
    # predictions?
    prediction = np.matmul(data, model)
    return prediction
#end def predict

def loss(data, model):
    t = data['t']
    X = data['X']
    p = predict(X,model)
    final = t - p
    # look at linear algebra rules
    dist = (np.linalg.norm(final)**2) /t.shape[0]
    return dist
# end def loss
```

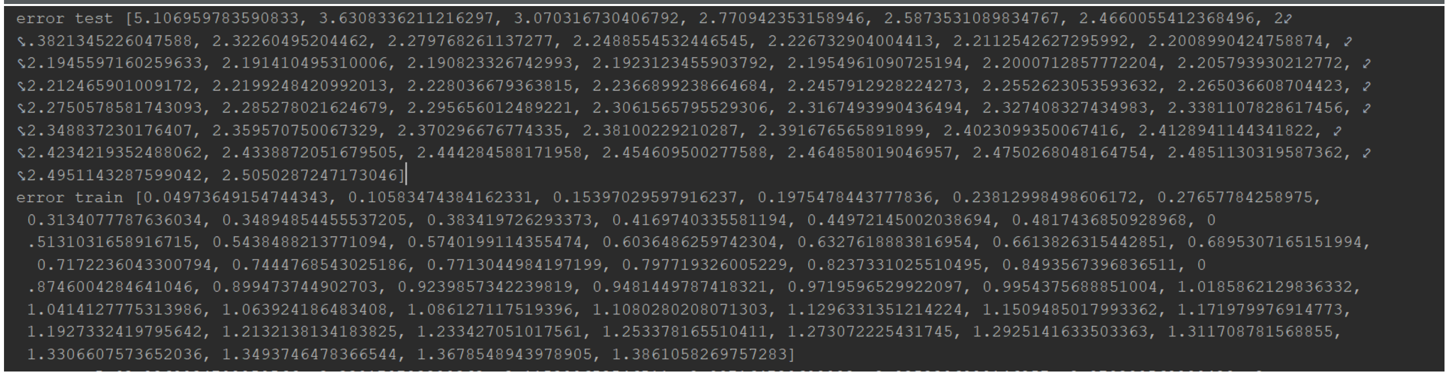


```
def cross_validation(data, num_folds, lambd_seq):
    cv_error = [None]*50
    data = shuffle_data(data)
    for i in range(len(lambd_seq)):
        lambd = lambd_seq[i]
        cv_loss_lmd = 0
        for fold in range(num_folds):
            train_cv , val_cv= split_data(data, num_folds, fold)
            model = train_model(train_cv, lambd)
            cv_loss_lmd += loss(val_cv, model)
            print(cv_loss_lmd)
        cv_error[i] = cv_loss_lmd / num_folds
    return cv_error

def train_test_error(data, data_2, lambd_seq):
    errors_trai = []
    errors_test = []
    for i in range(len(lambd_seq)):
        model = train_model(data, lambd_seq[i])
        error = loss(data, model)
        error2 = loss(data_2, model)
        errors_trai.append(error)
        errors_test.append(error2)

    return errors_trai, errors_test
```



```
if "__main__" == __name__:
    data_train = load_data_vector("data_train_y.csv", "data_train_X.csv")
    data_test = load_data_vector("data_test_y.csv", "data_test_X.csv")
    num_folds = 5
    lambd_seq = np.linspace(0.02, 1.5, num=50)
    error_cv_5 = cross_validation(data_train, num_folds, lambd_seq)
    num_folds = 10
    error_cv_10 = cross_validation(data_train, num_folds, lambd_seq)
    error_train, error_test = train_test_error(data_train, data_test, lambd_seq)


    #
    plot(error_train, error_test, lambd_seq, error_cv_5, error_cv_10)
    arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
    arr2 = np.array([13,14,15])
    data = {'X': arr, 't':arr2}
    print(shuffle_data(data))
```





```
error test [5.106959783590833, 3.6308336211216297, 3.070316730406792, 2.770942353158946, 2.5873531089834767, 2.4660055412368496, 2⤶
⤷.3821345226047588, 2.32260495204462, 2.279768261137277, 2.2488554532446545, 2.226732904004413, 2.2112542627295992, 2.2008990424758874, ⤶
⤷2.1945597160259633, 2.191410495310006, 2.190823326742993, 2.1923123455903792, 2.1954961090725194, 2.2000712857772204, 2.205793930212772, ⤶
⤷2.212465901009172, 2.2199248420992013, 2.228036679363815, 2.2366899238664684, 2.2457912928224273, 2.2552623053593632, 2.265036608704423, ⤶
⤷2.2750578581743093, 2.285278021624679, 2.295656012489221, 2.3061565795529306, 2.3167493990436494, 2.327408327434983, 2.3381107828617456, ⤶
⤷2.348837230176407, 2.359570750067329, 2.370296676774335, 2.38100229210287, 2.391676565891899, 2.4023099350067416, 2.4128941144341822, ⤶
⤷2.4234219352488062, 2.4338872051679505, 2.444284588171958, 2.454609500277588, 2.464858019046957, 2.4750268048164754, 2.4851130319587362, ⤶
⤷2.4951143287599042, 2.5050287247173046]
error train [0.04973649154744343, 0.10583474384162331, 0.15397029597916237, 0.1975478443777836, 0.23812998498606172, 0.27657784258975,
 0.3134077787636034, 0.34894854455537205, 0.383419726293373, 0.4169740335581194, 0.44972145002038694, 0.4817436850928968, 0
 .5131031658916715, 0.5438488213771094, 0.5740199114355474, 0.6036486259742304, 0.6327618883816954, 0.6613826315442851, 0.6895307165151994,
  0.7172236043300794, 0.7444768543025186, 0.7713044984197199, 0.797719326005229, 0.8237331025510495, 0.8493567396836511, 0
 .8746004284641046, 0.899473744902703, 0.9239857342239819, 0.9481449787418321, 0.9719596529922097, 0.9954375688851004, 1.0185862129836332,
 1.0414127775313986, 1.063924186483408, 1.086127117519396, 1.1080280208071303, 1.1296331351214224, 1.1509485017993362, 1.171979976914773,
 1.1927332419795642, 1.2132138134183825, 1.233427051017561, 1.253378165510411, 1.273072225431745, 1.2925141633503363, 1.311708781568855,
 1.3306607573652036, 1.3493746478366544, 1.3678548943978905, 1.3861058269757283]
```

Given this graph I would choose my lambda value to be 0.3 because it is where the error is reduce in my cross validation. The reason the error begins to increase again is that the model begins to over fit and hence fails at the validation part of the process because it is over fitted to the training data hence the error increases. Since the model was trained on the training data it would make sense that this graph has the lowest error. Since cross validation has truly unseen data at each point it will have a higher error during its hyper tuning, thus why the testing error is slight below the cv's error.