HOMEWORK 1

CSC311 Fall 2019

Nicole Xin Yue Wang
UtorID: wangnic8
#1004235339

1. Nearest Neighbors and the Curse of Dimensionality - 30 pts. In this question, you will verify the claim from lecture that "most" points in a high-dimensional space are far away from each other, and also approximately the same distance.

(a) *[10 pts]* First, consider two independent univariate random variables X and Y sampled uniformly from the unit interval $[0, 1]$. Determine the expectation and variance of the random variable Z, defined as the squared distance $Z = (X - Y)^2$.

SOLN $\mathbb{E}[Z] = \mathbb{E}[(X-Y)^2]$

$= \mathbb{E}[X^2 + Y^2 - 2XY]$

$= \mathbb{E}[X^2] + \mathbb{E}[Y^2] - 2\mathbb{E}[XY]$

$= \mathbb{E}[X^2] + \mathbb{E}[Y^2] - 2\mathbb{E}[X]\mathbb{E}[Y]$ # X, Y are independent

$\because X \sim \text{Uniform}(0,1), Y \sim \text{Uniform}(0,2)$

$\therefore f(x) = f(y) = \frac{1}{1-0} = 1$ ↳ pdf

$\Rightarrow \mathbb{E}[X^2] = \int_0^1 x^2 f(x)\,dx = \frac{1}{3}x^3\Big|_0^1 = \frac{1}{3}$

$\mathbb{E}[Y^2] = \int_0^1 y^2 f(y)\,dy = \frac{1}{3}y^3\Big|_0^1 = \frac{1}{3}$

$\mathbb{E}[X] = \int_0^1 x f(x)\,dx = \frac{1}{2}x^2\Big|_0^1 = \frac{1}{2}$

$\mathbb{E}[Y] = \int_0^1 y f(y)\,dy = \frac{1}{2}y^2\Big|_0^1 = \frac{1}{2}$

$\Rightarrow \mathbb{E}[Z] = \frac{1}{3} + \frac{1}{3} - 2\left(\frac{1}{2}\right)\left(\frac{1}{2}\right) = \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$

$\text{Var}(Z) = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2$

$= \mathbb{E}[(X-Y)^4] - \left(\frac{1}{6}\right)^2$

$= \mathbb{E}[X^4] + \mathbb{E}[Y^4] - 4\mathbb{E}[X^3]\mathbb{E}[Y] - 4\mathbb{E}[X]\mathbb{E}[Y^3] + 6\mathbb{E}[X^2]\mathbb{E}[Y^2] - \frac{1}{36}$

$= \int_0^1 x^4\,dx + \int_0^1 y^4\,dy - 4\int_0^1 x^3\,dx\left(\frac{1}{2}\right) - 4\left(\frac{1}{2}\right)\int y^3\,dy + 6\left(\frac{1}{3}\right)\left(\frac{1}{3}\right) - \frac{1}{36}$

$= \frac{1}{5} + \frac{1}{5} - 4\left(\frac{1}{4}\right)\left(\frac{1}{2}\right) - 4\left(\frac{1}{2}\right)\left(\frac{1}{4}\right) + \frac{2}{3} - \frac{1}{36}$

$= \frac{2}{5} - 1 + \frac{2}{3} - \frac{1}{36}$

$= \frac{72 - 180 + 120 - 5}{180}$

$= \frac{7}{180}$

ANS: $\mathbb{E}[Z] = 1/6$, $\text{Var}(Z) = \frac{7}{180}$

(b) *[10 pts]* Now suppose we sample two points independently from a unit cube in d dimensions. Observe that each coordinate is sampled independently from $[0, 1]$, i.e. we can view this as sampling random variables $X_1, \ldots, X_d, Y_1, \ldots, Y_d$ independently from $[0, 1]$. The squared Euclidean distance can be written as $R = Z_1 + \cdots + Z_d$, where $Z_i = (X_i - Y_i)^2$. Using the properties of expectation and variance, determine $\mathbb{E}[R]$ and $\mathrm{Var}[R]$. You may give your answer in terms of the dimension d, and $\mathbb{E}[Z]$ and $\mathrm{Var}[Z]$ (the answers from part (a)).

$$\mathbb{E}[R] = \mathbb{E}[Z_1 + \cdots + Z_d] = \mathbb{E}[Z_1] + \cdots + \mathbb{E}[Z_d] \quad \text{\# by properties of expectation}$$
$$= \underbrace{\tfrac{1}{6} + \cdots + \tfrac{1}{6}}_{d \text{ times}}$$
$$= d/6$$

$$\mathrm{Var}(R) = \mathrm{Var}(Z_1 + \cdots + Z_d) = \mathrm{Var}(Z_1) + \cdots + \mathrm{Var}(Z_d) \quad \text{\# by properties of variance}$$
$$= 7d/180$$

\# $\Rightarrow \mathrm{Var}(X+Y) = \mathrm{Var}(X) + 2\mathrm{Cov}(X,Y) + \mathrm{Var}(Y)$ while X and Y are independent $\Rightarrow \mathrm{Cov}(X,Y) = 0$

ANS: $\mathbb{E}[R] = d/6$, $\mathrm{Var}(R) = 7d/180$

(c) *[10 pts]* Based on your answer to part (b), compare the mean and standard deviation of R to the maximum possible squared Euclidean distance (i.e. the distance between opposite corners of the cube). Why does this support the claim that in high dimensions, "most points are far away, and approximately the same distance"?

$$\max(R) = \max\left(\sum_{i=1}^{d} Z_i\right) = \sum_{i=1}^{d} \max(Z_i)$$
$$= \sum_{i=1}^{d} \max\left((X_i - Y_i)^2\right)$$
$$= \sum_{i=1}^{d} (1)$$
$$= d$$

\# X and Y are sampled from unit interval $[0,1]$, the maximum difference will be $1 - 0 = 1$

In high-dimensions, since both mean and maximum possible values are multiples of d, the higher dimensions it get, the further the points get.

$\Rightarrow \lim_{d \to \infty} \mathbb{E}[R] = \lim_{d \to \infty} d/6 \to \infty$ $\Rightarrow$ therefore points could be very far away in high dimensions

If we define distance between "most" points to be within 3σ away from mean, the we have:

$$\sigma = \sqrt{\mathrm{Var}(R)} = \sqrt{\frac{7d}{180}}$$

$$\Rightarrow 3\sigma = 3\sqrt{\frac{7d}{180}}$$
$$\doteq 0.59\sqrt{d} \quad \Rightarrow \text{very small difference}$$

Therefore, in high-dimensions, "most" points are far away and approximately the same distance. ∎

2. (a) *[6 pts]* Write a function `load_data` which loads the data, preprocesses it using a vectorizer (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text, we suggest you use `CountVectorizer` as it is the simplest in nature), and splits the entire dataset randomly into 70% training, 15% validation, and 15% test examples.

```
def load_data():
    """ Load headlines from clean_real.txt and clean_fake.txt and seperate them
    into training set, validation set, and testing set. """
    # read clean_fake.txt and store each headline as an element into an array
    fake_file = open("clean_fake.txt", "r")
    fake_arr = fake_file.read().split('\n')
    fake_file.close()
    # read clean_real.txt and store each headline as an element into an array
    real_file = open("clean_real.txt", "r")
    real_arr = real_file.read().split('\n')
    real_file.close()
    #combine the two arrays into one sample array
    sample_arr = fake_arr + real_arr
    # target array, 0 means corresponding headline is fake, 1 means real
    target_arr = [0] * len(fake_arr) + [1] * len(real_arr)
    # split into three sets
    vectorizer = CountVectorizer()
    sample = vectorizer.fit_transform(sample_arr)
    x_train, x_test, y_train, y_test = train_test_split(
        sample.toarray(), target_arr, test_size=0.3)
    x_val, x_test, y_val, y_test = train_test_split(
        x_test, y_test, test_size=0.5)
    #return the dictionary
    return {"train":x_train, "train_target": y_train, "val":x_val,
            "val_target":y_val,  "test":x_test, "test_target": y_test,
            "feature_names":vectorizer.get_feature_names()}
```

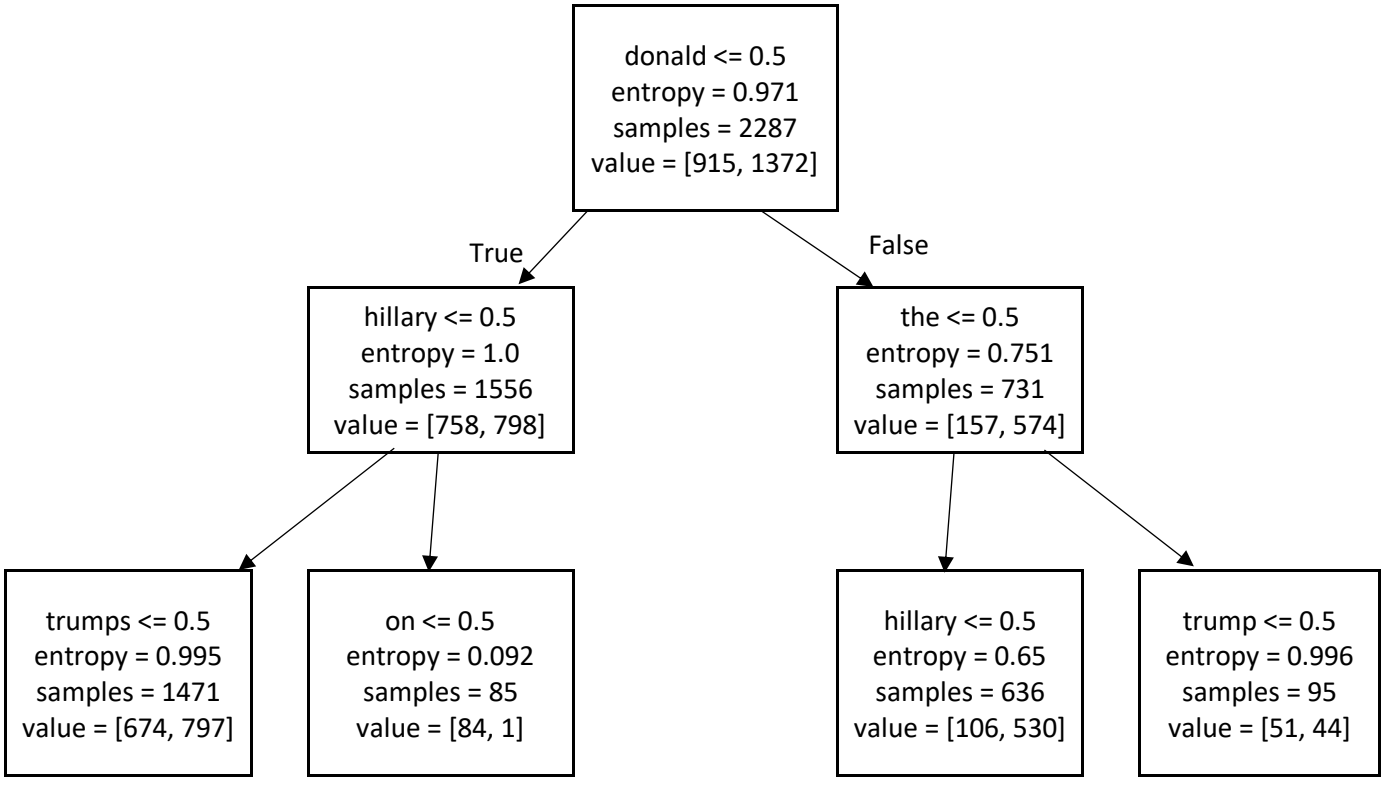
(b) *[6 pts]* Write a function `select_model` which trains the decision tree classifier using at least 5 different values of `max_depth`, as well as two different split criteria (information gain and Gini coefficient), evaluates the performance of each one on the validation set, and prints the resulting accuracies of each model. You should use `DecisionTreeClassifier`, but you should write the validation code yourself. Include the output of this function in your solution.

```
def select_model():
    """ Select the model with the most efficient hyperparameter. """
    # get data
    data = load_data()
    # initialize the variables for storing the model with best accuracy
    best_acc = 0
    best_depth = 0
    best_crit = ""
    # two criterion to choose
    for crit in ["gini", "entropy"]:
        # 5 different types of maximum depth of the tree
        for i in [4,8,12,16,20]:
            # train a tree with given criterion and depth
            clf = DecisionTreeClassifier(criterion=crit, max_depth=i)
            clf.fit(data["train"], data["train_target"])
            # accuracies testing on the validation set, store best accuracy
            acc = clf.score(data["val"], data["val_target"])
            (best_acc, best_depth, best_crit) = (
                best_acc, best_depth, best_crit
                ) if best_acc > acc else (acc, i, crit)
            print("Score (max_depth=", i, ", criterion=", crit, ") :", acc)
    print("Best Model(max_depth)=" , best_depth, ", criterion=",
          best_crit, "): Score=", best_acc)
```

(b) (cont'd) Output:

```
Score (max_depth= 4 , criterion= gini ) : 0.7081632653061225
Score (max_depth= 8 , criterion= gini ) : 0.7122448979591837
Score (max_depth= 12 , criterion= gini ) : 0.7285714285714285
Score (max_depth= 16 , criterion= gini ) : 0.746938775510204
Score (max_depth= 20 , criterion= gini ) : 0.7489795918367347
Score (max_depth= 4 , criterion= entropy ) : 0.6979591836734694
Score (max_depth= 8 , criterion= entropy ) : 0.7040816326530612
Score (max_depth= 12 , criterion= entropy ) : 0.7326530612244898
Score (max_depth= 16 , criterion= entropy ) : 0.7387755102040816
Score (max_depth= 20 , criterion= entropy ) : 0.753061224489796
Best Model(max_depth= 20 , criterion= entropy ): Score= 0.753061224489796
```

(c) *[6 pts]* Now let's stick with the hyperparameters which achieved the highest validation accuracy. Extract and visualize the first two layers of the tree. Your visualization does not have to be an image: it is perfectly fine to display text. It may also be hand-drawn. Include your visualization in your solution pdf.



* tree is generated by using *sklearn.tree.export_graphviz* to generate a dot file and visualized using http://www.webgraphviz.com/

(d) *[12 pts]* Write a function `compute_information_gain` which computes the information gain of a split on the training data. That is, compute $I(Y, x_i)$, where Y is the random variable signifying whether the headline is real or fake, and x_i is the keyword chosen for the split. Report the outputs of this function for the topmost split from the previous part, and for several other keywords.

```
def compute_information_gain(keyword):
    data = load_data()
    total = len(data["train_target"])   # total number of news
    # get the index of keyword in the list of features
    index = data["feature_names"].index(keyword)
    train = data["train"]
    real = 0            # number of real news
    real_key = 0        # number of real news with keyword
    key = 0             # number of news with keyword
    for i in range(total):
        if train[i][index] > 0:
            key += 1                        # count one news with keyword
            if data["train_target"][i] == 1 :
                real_key += 1               # count one real news with keyword
        if data["train_target"][i] == 1:
            real += 1                       # count one real news
    entropy = n_nlogn(real / total) + n_nlogn(
        (total - real)/ total)          # H(Y)
    p_e_r = real_key / key              # real news, with keyword
    p_e_f = (key-real_key) / key        # fake news, with keyword
    p_ne_r = (real-real_key) / (total-key)              # real news, no keyword
    p_ne_f = (total-real-(key-real_key)) / (total-key)  # fake news, no keyword
    entropy_keyword = (key/total) * (n_nlogn(p_e_r) + n_nlogn(p_e_f)) + (
        (total-key)/total) * (n_nlogn(p_ne_r) + n_nlogn(p_ne_f))    # H(Y|xi)
    IG = entropy - entropy_keyword      # information gain
    print("Information Gain in splitting at keyword '", keyword, "' is ", IG)

if __name__ == "__main__":
    for key in ["donald", "trumps", "hillary", "the", "trump",
                "breaking", "prime"]:
        compute_information_gain(key)
```

Output:

```
Information Gain in splitting at keyword ' donald ' is  0.051323670040992586
Information Gain in splitting at keyword ' trumps ' is  0.044374383775848236
Information Gain in splitting at keyword ' hillary ' is  0.03859640155331756
Information Gain in splitting at keyword ' the ' is  0.05584727379340548
Information Gain in splitting at keyword ' trump ' is  0.038198158834489626
Information Gain in splitting at keyword ' breaking ' is  0.010519717839329035
Information Gain in splitting at keyword ' prime ' is  3.0293720648333355e-05
```

3. Regression - 40 pts. In this question, you will derive certain properties of linear regression, and experiment with cross validation to tune its regularization parameter.

3.1. *Linear regression - 10 pts.* Suppose that $\mathbf{X} \in \mathbb{R}^{n \times m}$ with $n \geq m$ and $\mathbf{t} \in \mathbb{R}^n$, and that $\mathbf{t}|(\mathbf{X}, \mathbf{w}) \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$. We know that the maximum likelihood estimate $\hat{\mathbf{w}}$ of $\mathbf{w}$ is given by

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{t}.$$

(a) Write the log-likelihood implied by the model above, and compute its gradient w.r.t. $\mathbf{w}$. By setting it equal to 0, derive the above estimator $\hat{\mathbf{w}}$.

Given that $t|(X,w) \sim \mathcal{N}(Xw, \sigma^2 I)$, we know that:

$$p(t|(X,w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(t_i - x_i w)^2\right\}$$

likelihood

$$\Rightarrow \log p(t|X,w) = \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2 I}} \exp\left\{-\frac{1}{2\sigma^2 I}(t_i - x_i w)^2\right\}\right)$$

$$= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}} \exp\left\{-\frac{1}{2\sigma^2 I}(t_i - x_i w)^2\right\}\right)$$

$$= \sum_{i=1}^{n} \left(-\frac{1}{2\sigma^2 I}(t_i - x_i w)^2\right) \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right) \quad \text{\# by power rule of logarithms}$$

$$= -\frac{1}{2\sigma^2 I} \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right) \sum_{i=1}^{n} (t_i - x_i w)^2 \quad \text{\# by distributive property of summation and } -\frac{1}{2\sigma^2}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right) \text{ is a constant}$$

$$= -\frac{1}{2\sigma^2 I} \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right) \underbrace{(t - Xw)^T}_{=(t^T - w^T X^T)}(t - Xw)$$

$$= -\frac{1}{2\sigma^2 I} \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(t^T t - t^T X w - w^T X^T t + w^T X X w)$$

$$\Rightarrow \nabla[\log p(t|X,w)] = \left[-\frac{1}{2\sigma^2 I} \log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)\right](-t^T X - X^T t + 2X^T X w)$$

let $\nabla = 0$, then $-t^2 X - X^T t + 2X^T X w = 0$

$$2X^T X w = t^T X + X^T t$$

since $t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}$, then $t^T = [t_1, t_2, \ldots, t_n]$

since $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}$, then $X^T = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1m} & x_{2m} & \cdots & x_{nm} \end{bmatrix}$

$$\Rightarrow t^T X = [t_1 x_{11} + \cdots + t_n x_{n1}, \; t_1 x_{12} + \cdots + t_n x_{n2}, \; \cdots, \; t_1 x_{1m} + \cdots + t_n x_{nm}]$$

and $X^T t = \begin{bmatrix} x_{11}t_1 + \cdots + x_{n1}t_n \\ x_{12}t_1 + \cdots + x_{n2}t_n \\ \vdots \\ x_{1m}t_1 + \cdots + x_{nm}t_n \end{bmatrix}$

$\because$ both $t^T X$ and $X^T t$ are vectors

$\therefore t^T X = X^T t$

$$\Rightarrow \cancel{2}X^T X w = \cancel{2} X^T t$$

$$w = (X^T X)^{-1} X^T t \qquad \blacksquare$$

(b) Find the distribution of $\hat{\mathbf{w}}$, its expectation and covariance matrix. Hint: Read the property of multivariate Gaussian random vectors in preliminaries.pdf on course webpage.

Since $\hat{w} = (X^T X)^{-1} X^T t$, t is a normal ditributed random variable, and $(X^T X)^{-1} X^T$ is a constant matrix, by the property of a multivariate Gaussian random variable, $\hat{w}$ is also a multivariate Gaussian random vector.

mean: $[(X^T X)^{-1} X^T] Xw = \underbrace{(X^T X)^{-1}(X^T X)}_{I}(X^T X)^{-1} X^T t$

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

covariance: $[(X^T X)^{-1} X^T]\, \sigma^2 I [(X^T X)^{-1} X^T]^T = \sigma^2 [(X^T X)^{-1} X^T][(X^T X)^{-1} X^T]^T$

Let $A = (X^T X)^{-1} X^T$, then $\hat{w} \sim \mathcal{N}(At, \sigma^2 AA^T)$

3.2. *Ridge regression and MAP - 10 pts.* Suppose that we have $\mathbf{t}|(\mathbf{X},\mathbf{w}) \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$ and we place a normal prior on $\mathbf{w}|\mathbf{X}$, i.e., $\mathbf{w} \sim \mathcal{N}(0, \tau^2\mathbf{I})$. Recall from the first tutorial (also in preliminaries.pdf) that MAP estimate of $\mathbf{w}$ is given as the maximum of the posterior density

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmax}}\{p(\mathbf{w}|\mathbf{X},\mathbf{t}) \propto p(\mathbf{t}|\mathbf{X},\mathbf{w})p(\mathbf{w}|\mathbf{X})\}.$$

Here, $\propto$ notation means *proportional to*, and is used since we dropped the term $p(\mathbf{t}|\mathbf{X})$ in the denominator as it doesn't have $\mathbf{w}$ in it, thus it doesn't contribute to the maximization problem.

Show that the MAP estimate of $\mathbf{w}$ given $(\mathbf{t},\mathbf{X})$ in this context is

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{t} \tag{3.1}$$

where $\lambda = \sigma^2/\tau^2$.

SOLN

$$p(w|X,t) \propto p(t|X,w)\,p(w|X)$$

$$\Rightarrow \log p(w|X,t) \propto \log\left[p(t|Xw)\,p(w|X)\right]$$

$$= \log\left(\prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2 I_i}}\exp\left\{-\frac{1}{2\sigma^2 I}(t_i - x_i w)^2\right\}\right)\left(\prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\tau^2 I}}\exp\left\{-\frac{1}{2\tau^2 I}(w_i)^2\right\}\right)$$

$$= -\frac{1}{2\sigma^2 I}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(t^\top t - t^\top X w - w^\top X^\top t + w^\top X^\top X w) + \sum_{i=1}^{n}\log\frac{1}{\sqrt{2\pi\tau^2 I}}\exp\left\{-\frac{1}{2\tau^2 I}(w_i)^2\right\}$$

$$= -\frac{1}{2\sigma^2 I}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(t^\top t - t^\top X w - w^\top X^\top t + w^\top X^\top X w) + \sum_{i=1}^{n}\left[-\frac{1}{2\tau^2 I}(w_i)^2\right]\log\left(\frac{1}{\sqrt{2\pi\tau^2 I}}\right)$$

$$= -\frac{1}{2\sigma^2 I}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(t^\top t - t^\top X w - w^\top X^\top t + w^\top X^\top X w) - \frac{1}{2\tau^2 I}\log\left(\frac{1}{\sqrt{2\pi\tau^2 I}}\right)\sum_{i=1}^{n} w_i^2$$

$$= -\frac{1}{2\sigma^2 I}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(t^\top t - t^\top X w - w^\top X^\top t + w^\top X^\top X w) - \frac{1}{2\tau^2 I}\log\left(\frac{1}{\sqrt{2\pi\tau^2 I}}\right) w^\top w$$

$$\Rightarrow \nabla = -\frac{1}{2\sigma^2 I}\log\left(\frac{1}{\sqrt{2\pi\sigma^2 I}}\right)(-t^\top X - X^\top t + 2X^\top X w) - \frac{1}{2\tau^2 I}\log\frac{1}{\sqrt{2\pi\tau^2 I}}\,(2w)$$

let $\nabla = 0$, then:

$$-\frac{1}{2\sigma^2 I}\log\left[(2\pi\sigma^2 I)^{-\frac{1}{2}}\right](-t^\top X - X^\top t + 2X^\top X w) - \frac{1}{\tau^2 I}\log\left[(2\pi\tau^2 I)^{-\frac{1}{2}}\right] w = 0$$

$$+\frac{1}{4\sigma^2 I}\log(2\pi\sigma^2 I)(-t^\top X - X^\top t + 2X^\top X w) = \frac{-1}{2\tau^2 I}\log(2\pi\tau^2 I)\, w$$

$$(-t^\top X - X^\top t + 2X^\top X w) = -\frac{\sigma^2 I}{\tau^2 I}\frac{4\log(2\pi\sigma^2 I)}{2\log(2\pi\tau^2 I)} w$$

$$\frac{-2X^\top t}{w} + 2X^\top X = -\frac{\sigma^2}{\tau^2} I\,\frac{2\log(2\pi\sigma^2 I)}{\log(2\pi\sigma^2 I)}$$

$$\frac{2X^\top t}{w} = 2X^\top X + \frac{\sigma^2}{\tau^2} I\,\frac{2\log(2\pi\sigma^2 I)}{\log(2\pi\sigma^2 I)}$$

$$w = \left(X^\top X + \frac{\sigma^2}{\tau^2} I\,\frac{\log(2\pi\sigma^2 I)}{\log(2\pi\sigma^2 I)}\right)^{-1} X^\top t$$

$$\hat{w}_{MAP} = (X^\top X + \lambda I)^{-1} X^\top t$$

let $\lambda = \frac{\sigma^2}{\tau^2}$, and drop $\frac{\log(2\pi\sigma^2 I)}{\log(2\pi\sigma^2 I)}$ since it has very small affect comparing to $\frac{\sigma^2}{\tau^2}$, and we can adjust λ. ∎

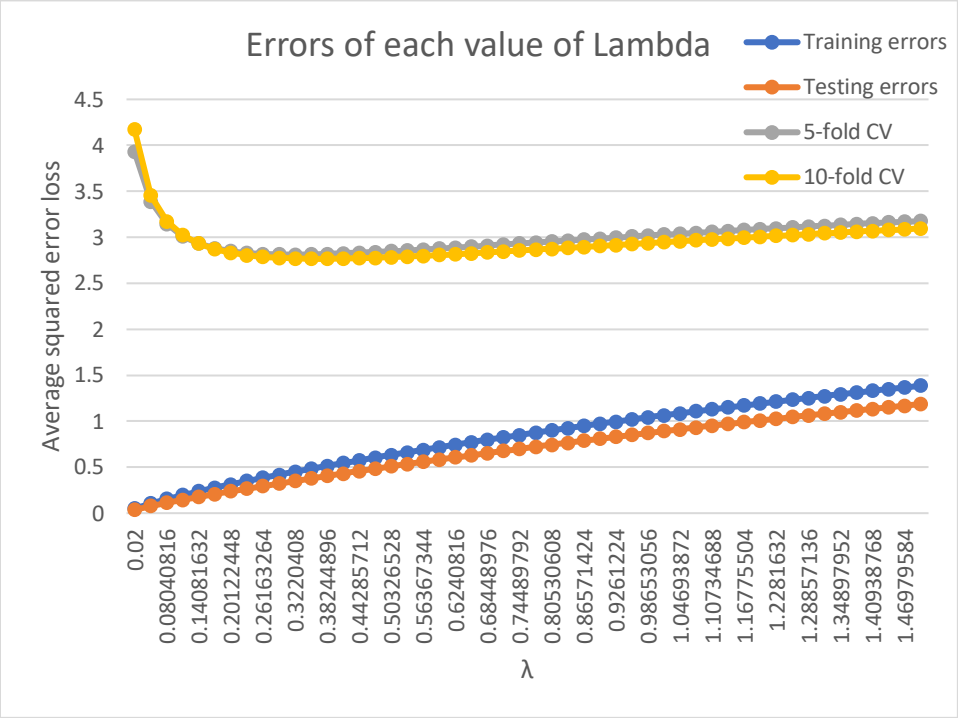
3.3. *Cross validation - 20 pts.*

(b) Code for the 6 functions:

```python
import numpy as np
from numpy.linalg import pinv

def shuffle_data(data):
    """ Takes in a data that contains a (t,X) pair and returns its randomly
    permuted version along the samples. """
    # generate an array that contains a random order of indices for data
    random_order = np.random.permutation(len(data["X"]))
    # return de reordered X and t
    return {"X": data["X"][random_order], "t": data["t"][random_order]}

```

```python
def split_data(data, num_fold, fold):
    """
    Takes in data that is a (t,X) pair, folds each row into num_fold equal folds
    and the foldth will be the validation fold. Returns the validation fold as
    a 1d array and the rest folds as a 2d array.
    """
    # fold the samples and target
    folded_x = np.array_split(data["X"], num_fold)
    folded_t = np.array_split(data["t"], num_fold)
    # pop the fold to be used as validation set
    data_fold = {"X": folded_x.pop(fold-1), "t": folded_t.pop(fold-1)}
    # add the rest of the folds into return value
    data_rest = {"X":np.concatenate(folded_x), "t":np.concatenate(folded_t)}
    return data_fold, data_rest


def train_model(data, lambd):
    """
    Takes a set of data and return the coefficient of rideg regression with
    penalty level lambd.
    """
    x = np.array(data["X"])
    t = data["t"]
    x_trans = x.transpose()
    return np.matmul(pinv(np.matmul(x_trans, x) +
             np.dot(lambd,np.identity(len(x_trans)))), np.matmul(x_trans, t))

def predict(data, model):
    """ Takes in data and model coefficient and predict the outcome. """
    return np.matmul(data["X"], model)

def loss(data, model):
    """ Takes in the data and model, and return the average sqaured error loss.
    """
    t_xw = data["t"] - predict(data, model)
    return t_xw.dot(t_xw) / len(data["t"])


def cross_validation(data, num_folds, lambd_seq):
    """ Takes in the training data, number of folds, and a sequence of λ and
    return a vector of 50 cross validation erros. """
    cv_error = []
    data = shuffle_data(data)
    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.append(cv_loss_lmd / num_folds)
    return cv_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 = np.linspace(0.02, 1.5, num=50)

```

(d) Ploting



Training errors and testing errors follows a similar pattern while the two cross validations follows a smilar patten while more fold makes slighty smaller errors than less folds. The training and testing erros are way smaller than cross validations. However, this is probably because error evaluations on traning and testing datas are on the exact same set of datas used for traning, while cross validations uses some set of sample datas for training, and then error losts are calculated by using a validation set which the model never seen before. If a new set of samples are to be tested on the model trained by the training and testing datas, the errors would increase.