CSC 2515: Introduction to Machine Learning

Homework 1

By: Alexis Bruneau

Collaborated with: Ramy ElMallah, Anton Korikov

Presented to Dr.Amir-massoud Farahmand

October 16th, 2022

Question 1 Nearest Neighbours and the Curse of Dimensionality – 15 pts

a) [5pts] 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 $Z = |X - Y|^2$, i.e, the squared distance between X and Y.

$$Z = |X - Y|^2$$

$$Z = X^2 - 2XY + Y^2$$

Since X and Y are independent, we can say the following:

$$E[Z] = E[X^2] - 2E[X]E[Y] + E[Y^2] \qquad [1.a1]$$

The following is known for uniform distribution for interval $[a, b]$. The mean can be calculated with the following:

$$\mathrm{E(X)} = \int_a^b xf(x)dx = \int_a^b \frac{x}{b-a}dx = \frac{a+b}{2}$$

$$E(X^2) = \int_a^b \frac{x^2}{b-a}dx$$

and the variance can be calculated by

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

$$Var(X) = \int_a^b \frac{x^2}{b-a}dx - \left(\frac{a+b}{2}\right)^2 = \frac{(b-a)^2}{12}$$

$$\sigma = \frac{(b-a)^2}{12}$$

Substituting back the values in [1.a1], we get:

$$E[Z] = \left|\int_a^b \frac{x^2}{b-a}dx - 2\left(\frac{a+b}{2}\right)\left(\frac{a+b}{2}\right) + \int_a^b \frac{y^2}{b-a}dy\right|$$

$$E[Z] = \left|\frac{1}{3} * \frac{1}{(b-a)} * (x^3)_a^b - 2\left(\frac{a+b}{2}\right)\left(\frac{a+b}{2}\right) + \frac{1}{3} * \frac{1}{(b-a)} * (y^3)_a^b\right|$$

$$E[Z] = \left|\frac{1}{3} * \frac{1}{(b-a)} * (b^3 - a^3) - 2\left(\frac{a+b}{2}\right)\left(\frac{a+b}{2}\right) + \frac{1}{3} * \frac{1}{(b-a)} * (b^3 - a^3)\right|$$

Replace a and b by upper and lower bound [0,1].

$$E[Z] = \left|\frac{1}{3} * \frac{1}{(1-0)} * (1-0) - 2 * \left(\frac{(1+1)}{2}\right) * \left(\frac{(1+1)}{2}\right) + \frac{1}{3} * \frac{1}{(1-0)} * (1-0)\right|$$

$$\boldsymbol{E[Z] = \frac{1}{6}}$$

The variance of Z can be calculated similarly.

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

Recall that

$$Z = |X^2 - 2XY + Y^2|$$

Hence

$$Z^2 = |X^4 + Y^4 + 6X^2Y^2 - 4X^3Y - 4XY^3|$$

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

$$E[Z^2] = \left| \int_a^b \frac{x^4}{b-a} dx + \int_a^b \frac{y^4}{b-a} dy + 6 \left(\int_a^b \frac{x^2}{b-a} dx \right) \left(\int_a^b \frac{y}{b-a} dy \right) - 4 \left(\int_a^b \frac{x^3}{b-a} dx \right) \left(\int_a^b \frac{y}{b-a} dy \right) - 4 \left(\int_a^b \frac{x}{b-a} dx \right) \left(\int_a^b \frac{y^3}{b-a} dy \right) \right|$$

$$E[Z^2] = \left| \frac{1}{5} * \frac{1}{(b-a)} * (x^5)_a^b + \frac{1}{5} * \frac{1}{(b-a)} * (y^5)_a^b + \frac{6}{9} * \frac{1}{(b-a)} * (x^3)_a^b * \frac{1}{(b-a)} * (y^3)_a^b - \frac{1}{2} * \frac{1}{(b-a)} * (x^4)_a^b * \frac{1}{(b-a)} * (y^2)_a^b - \frac{1}{2} * \frac{1}{(b-a)} * (x^4)_a^b * \frac{1}{(b-a)} * (y^2)_a^b \right|$$

Substituting b by 1 and a by 0, we get

$$E[Z^2] = \frac{2}{5} + \frac{6}{9} - 1 = \frac{1}{15}$$

$$\boldsymbol{Var[Z] = \frac{1}{15} - \left(\frac{1}{6}\right)^2 = \frac{7}{180}}$$

Hence, the expected value E[Z] is equal to $\frac{1}{6}$ and Var[Z] is equal to $\frac{7}{180}$

b) [5 pts] Now suppose we draw two d-dimensional points X and Y from a d-dimensional unit cube with a uniform distribution, i.e, X,Y$\epsilon[\mathbf{0},\mathbf{1}]^{d}$. Observe that each coordinate is sampled independently and uniformly from [0,1], that is, we can view this as drawing random variables $\boldsymbol{X_1, \ldots, X_d}$ and $\boldsymbol{Y_1, \ldots, Y_d}$ independently and uniformly from [0,1]. The squared Euclidean distance $\boldsymbol{\left|\left|X-Y\right|\right|_2^2}$ can be written as $\boldsymbol{R = Z_1 + \cdots + Z_d}$ where $\boldsymbol{Z_i = |X_i - Y_i|^2}$. Using the properties of expectation and variance, determine $\boldsymbol{E\left[\left|\left|X-Y\right|\right|_2^2\right] = E[R]}$ and $\boldsymbol{Var\left[\left|\left|X-Y\right|\right|_2^2\right] = Var[R]}$. You may give your answer in terms of the dimension d, and $\boldsymbol{E[Z]}$ and Var[Z] (the answers from part a.

Assuming linearity of expectation, which says the following:

$$E[A+B] = E[A] + E[B]$$

We can say the following for $E[R]$

$$E[R] = E[Z_1] + E[Z_2] + E[Z_3] \ldots + E[Z_n]$$

Hence using the value of E[Z] found in part a), we can say the following:

$$\boldsymbol{E[R] = \frac{d}{6}}$$

And for d = 2

$$E[R] = \frac{2}{6} = \frac{1}{3}$$

Similarly, since $Z_1, Z_i, \ldots, Z_d$ are independent and for independent random variables of X_i we can say the following:

$$Var\left(\sum_i a_i X_i\right) = \sum_i a_i^2 var(X_i)$$

Then we can say the following:

$$Var(R) = Var(Z_1) + Var(Z_i) + \cdots + Var(Z_d) = dVar(Z) = \frac{7}{180} * d$$

For the case where d = 2 we get the following for E[R] and Var(R)

$$E[R] = \frac{d}{6} = \frac{1}{3}$$

$$\boldsymbol{Var(R) = \frac{7}{180} * d = \frac{7}{90}}$$

c) ***Based on your answer to part (b), compare the mean and standard deviation of*** $\left||X-Y|\right|_2^2$ ***to the maximum possible squared Euclidean distance between two points within the d-dimensional unit cube (this would be 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 have the same distance".***

The maximum possible squared Euclidean distance between two points in a d-dimensional cube (length of 1) can be represented with the following equation:

$$D_{ij}^2 = \sum_{v=1}^{d} \left(X_{vi} - Y_{vj}\right)^2 = [(1-0)^2 + (1-0)^2 + \cdots] = d(1-0)^2 = d$$

From the previous question, we saw that the mean squared Euclidean distance between two points within the d-dimensional cube is $\frac{d}{6}$. Also using the variance found in part b $Var(R) = \frac{7}{180}d$, we can say that the standard deviation of the squared Euclidean distance between two points within the d dimensional cube is $\sqrt{(\frac{7}{180}d)}$.

Looking at the values mentioned, we can say that the mean squared Euclidean distance between two points within the d-dimensional cube is $\frac{1}{6}$ of the maximum squared Euclidean distance. We can also say that as d approaches infinity, the mean squared Euclidean distance goes to infinity as well. Hence, in high dimension, the mean squared Euclidean distance gets large since its value is proportional to the number of dimensions.

From the equations above, we can also say that the standard deviation is relatively small and does not grow as quickly as the expected distance. This can be said because of the square root and d being multiplied by $\frac{7}{180}$ inside the squared root. This means that in high dimensions, the expected mean distance will be much larger than the standard deviation. This results in points that are approximately the same distance in high dimensions.

Question 2 Limiting Properties of the Nearest Neighbour Algorithm – 10pts.

In this question, we will study the limiting properties of the k-nearest neighbour algorithm. Suppose that we are given n data points $X_1, \dots X_n$, sampled uniformly and independently from the unit interval [0,2] and one test point, y=1.

Let Z_i $|X_i - y|$ denote the distance of y to the data point X_i and $Z_i = |X_i - y|$ denote the distance of y its i-th nearest neighbour. Then $Z_1 < \cdots < Z_n$

a) [2pts] Show that the random variable Z_i is uniformly distribute between the unit interval [0,1]

We cab show that a random variable Z_i is uniformly distributed between the unit interval [0,1] in the following way. We can define the CDF of X_i by F_x. We know that the sum of the CDF needs to be equal to 1 and that X_i is defined by the unit interval [0,2]. Hence, we can say the following:

$$f_{X_i}(x) = \begin{cases} 0.5, & 0 \le x \le 2 \\ 0 & otherwise \end{cases}$$

We can then define F_x as 0.5x for $x\varepsilon[0,2]$, and 0 elsewhere. We can define F_{X_i-1} to be the CDF of $X_i - 1$. This can be represented as $0.5(x-1)$ for $x\varepsilon[0,2]$ and 0 elsewhere. Since $Z_i = |X_i - 1|$, the domain of Z is [0,1]. The CDF of Z which is implied by F_x and F_{X_i-1} is the following:

$$F_z = \begin{cases} F_Z = z, & z\epsilon[0,1] \\ 0, & otherwise \end{cases}$$

This is a uniform distribution between the interval [0,1] to which the PDF is

$$f_z(z) = 1$$

[4 pts] Show that $E[Z_1] = \frac{1}{n+1}$, i.e, the expected distance the 1st nearest neighbour is $\frac{1}{n+1}$.

Using the following statement can help show that $E[Z_1] = \frac{1}{n+1}$:

$$P\{Z_1 > t\} = P\{\min_{i=1,\dots,n} Z_i > t\}$$

We know that

$$Z_i = |X_i - y| = |X_i - 1|$$

It is also known that for $Z(k)$ to be between t and $t + dt$, we need the following:

- $k - 1$ samples smaller than t

- 1 sample between $(t, t + dt)$

- $n - k$ samples between $(t + dt, 1)$

Because Z_i is uniformly distributed on [0,1], we can say the following:

- $P(S_{k-1} < t) = t$
- $P(t < S_1 < t + dt) = dt$
- $P(t + dt < S_{n-k} < 1) = 1 - t - dt$

We can then say the following:

$$P\{Z_{(k)}\} = \frac{n!}{(k-1)!\,(n-k)!} t^{k-1}\, dt(1 - t - dt)^{n-k}$$

$$fz_k = \frac{n!}{(k-1)!\,(n-k)!} t^{k-1}(1-t)^{n-k}$$

$$E[Z_k] = \frac{n!}{(k-1)!\,(n-k)!} \int_0^1 t^k (1-t)^{n-k} dt$$

We can you use Beta function properties to solve the integral. The Beta properties are as such for p and q values bigger than 0:

$$B(p,q) = \int_0^1 t^{p-1}(1-t)^{q-1}\, dt$$

$$B(p,q) = \frac{(p-1)!\,(q-1)!}{(p+q-1)!}$$

If we replace p by k and q by (n+1-k), we get the beta function. The value of p and k will be positive numbers since $k > 0$, $n > 0$, and $k \leq n$.

$$B(k+1, n-k+1) = \frac{(k+1-1)!\,\big((n-k+1)-1\big)!}{\big((k+1+n-k+1)-1\big)!} = \frac{(k)!\,(n-k)!}{(n+1)!}$$

Hence, the expectation can be written as such:

$$E[Z_k] = \frac{n!}{(k-1)!\,(n-k)!}\left(\frac{(k)!\,(n-k)!}{(n+1)!}\right)$$

$$= \frac{n!}{(k-1)!}\left(\frac{(k)!}{(n+1)!}\right)$$

$$= \frac{n!\,k!}{(k-1)!\,(n+1)!} = \frac{k}{(n+1)}$$

By replacing the value of k for 1 for the 1st closest neighbour, the expected value is:

$$E[Z_1] = \frac{k}{(n+1)} = \frac{1}{n+1}$$

$$E(Z_k) = n!$$

c) Determine the expected value of the random variable Z_k , that is the expected distance to the k-th nearest neighbour.

$$f_{z_k}(t) = \frac{n!}{(k-1)!\,(n-k)!}\,t^{k-1}(1-t)^{n-k}, t\epsilon[0,1]$$

Using the pdf, we can find the expected value $Z_k(t)$ using the following equation.

$$E(Z_k) = \int_0^1 t\left(\frac{n!}{(k-1)!\,(n-k)!}t^{k-1}(1-t)^{n-k}\right)dt$$

Since n and k are constants, we can rewrite this equation like the following:

$$E(Z_k) = \frac{n!}{(k-1)!\,(n-k)!}\int_0^1 t^k(1-t)^{n-k}dt$$

Using the properties of the beta function we can computer the integral. The beta function takes the form

$$B(p,q) = \int_0^1 t^{p-1}(1-t)^{q-1}\,dt$$

Hence, we get the following values for p and q

$$Beta(k+1, n-k+1)$$

The beta function is closely related to the binomial coefficients. If m or n in the following equation is a positive integer, Beta(m,n) can be written the following way. In our case k and n are positive integers since $k > 0$, $n > 0$ and $k \leq n$:

$$B(p,q) = \frac{(p-1)!\,(q-1)!}{(p+q-1)!}$$

If we replace p by k and q by (n+1-k), we get the following:

$$B(k+1, n-k+1) = \frac{(k+1-1)!\,\big((n-k+1)-1\big)!}{\big((k+1+n-k+1)-1\big)!} = \frac{(k)!\,(n-k)!}{(n+1)!}$$

From this we can do the following substitution:

$$\int_0^1 t^k(1-t)^{n-k}dt = \frac{k!\,(n-k)!}{(n+1)!}$$

Which gives the following:

$$E(Z_k) = \frac{n!}{(k-1)!\,(n-k)!} * \frac{k!\,(n-k)!}{(n+1)!}$$

$$E(Z_k) = \frac{n!\,k!}{(k-1)!\,(n+1)!} = \frac{k}{(n+1)}$$

(d) [2 pts] Based on your answer to part (c), what can you say about the expected distance to the k-th nearest neighbour as $n \to \infty$ and $\frac{k}{n} \to 0$.

From the expected value found at c where $E(Z_k) = \frac{k}{(n+1)}$ we can say the following for the expected distance to the k-th nearest neighbour when $n \to \infty$. As the number of sample point approaches infinity, the **expected value will tend towards 0.** This can be shown with the following equation:

$$\lim_{n\to\infty} \frac{k}{n+1} = \frac{k}{\infty+1} = \frac{k}{\infty} = 0$$

as n is in the denominator of the expected value and dominates k. This makes sense as if there is an infinity number of sample points, the sample space is occupied by infinity number of points (even same location are repeated). Hence, the distance to the closest neighbour would be 0.

For the case when $\frac{k}{n} \to 0$, the expected value **will be 0**. The reason we can say that is that $k > 0$, $n > 0$, and $k \leq n$. Hence the only way that $\frac{k}{n} \to 0$ is if $n \to \infty$, or k is proportionally small compared to n. We evaluated this case previously. We can demonstrate the case when k =1 and n = 1000000

$$E(Z) = \frac{k}{n+1} = \frac{1}{1000000+1} \approx 0$$

Question 3 Information Theory – 15pts.

Recall the definition of the entropy of a discrete random variable X with probability mass function p:

$$H(X) = \sum_x p(x) \log_2 (\frac{1}{p(x)})$$

Here the summation is over all possible values of $x \in \chi$, which (for simplicity) we assume is finite. For example, χ might be $\{1,2, \dots, N\}$.

a)[3pt] Prove that the entropy H(X) is non-negative

$$H(X) = \sum_x p(x) \log_2 \left(\frac{1}{p(x)}\right)$$

$$H(X) = \sum_x p(x)(\log_2 1 - \log_2 p(x))$$

$$H(X) = \sum_x p(x)(0 - \log_2 p(x))$$

$$H(X) = -\sum_x p(x) \log_2 p(x))$$

Where p(x) represents the probability max function

$$0 \leq p(x) \leq 1$$

The three following cases can be calculated to prove that H(X) is always positive:

$$p(x) = 0$$

$$0 < p(x) < 1$$

$$p(x) = 1$$

When $p(x) = 0$, $H(X) = 0$

When $p(x) = 1$, $H(X) = 0$ since $\log_2 1 = 0$

When $0 < p(x) < 1$, $H(X)$ will always be positive because for any possible value of $p(x)$ between 0 and 1, $\log_2 p(x)$ will give a negative result. The result of the summation of negative results will give a negative result. The negative sign in front $\sum -result$ will change the sign and make $H(X)$ positive.

b)[3pt] If X and Y are independent random variables, show that $\boldsymbol{H(X,Y) = H(X) + X(Y)}$

From the chain rule we can say:

$$H(X,Y) = \sum_x \sum_y P(x,y) \log\left(\frac{1}{P(x,y)}\right) \qquad [3.b1]$$

We can say the following due to the independent random variables

$$P(x,y) = P(x)P(y)$$

$$H(X,Y) = -\sum_x \sum_y P(x)P(y) \log_2\big(P(x)P(y)\big)$$

$$= -\sum_{x,y} P(x)P(y) \log_2 P(x) - (-)\sum_{x,y} P(x)P(y) \log_2 P(y)$$

$$= -\sum_y P(y) \sum_x P(x) \log_2 P(x) - \sum_y P(x) \sum_x P(y) \log_2 P(y)$$

$$= -(1)\sum_x P(x) \log_2 P(x) - (1)\sum_x P(y) \log_2 P(y)$$

$$\boldsymbol{H(X,Y) = H(X) + H(Y)}$$

c)[3pt] Prove the chain rule for entropy $\boldsymbol{H(X,Y) = H(X) + H(Y|X)}$

We know that:

$H(X) = -\sum_{x\epsilon X} p(x) \log_2 p(x)$ 523

$$H(Y|X) = -\sum_{x\epsilon X} \sum_{y\epsilon Y} p(x,y) \log_2 p(y|x)$$

We also know the following:

$$H(X,Y) = -\sum_{x\epsilon X} \sum_{y\epsilon Y} p(x,y) \log_2 p(x,y)$$

$$= -\sum_{x\epsilon X} \sum_{y\epsilon Y} p(x,y) \log_2 p(y|x)p(x)$$

This can be rewritten as the following:

$$= -\sum_{x\epsilon X} \left(\sum_{y\epsilon Y} p(x,y) \right) \log_2 p(x) - \sum_{x\epsilon X} \sum_{y\epsilon Y} (p(x,y) \log_2 p(y|x)$$

$$= -\sum_{x \epsilon X} p(x) \log_2 p(x) - \sum_{x \epsilon X} \sum_{y \epsilon Y} (p(x,y) \log_2 p(y|x)$$

$$= H(X) + H(Y|X)$$

Hence

$$\boldsymbol{H(X,Y) = H(X) + H(Y|X)}$$

(d) [3pt] Prove that KL(p||q) is non-negative. Hint: You may want to use Jensen's inequality, which is described in the Appendix

We know the following and we can assume the following for all x:

$$KL(p||q) = \sum_x p(x) \log_2 \frac{p(x)}{q(x)} \; [3.d1]$$

$$p(x) > 0$$
$$q(x) > 0$$

This can be rewritten as the following:

$$KL(p||q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \; [3.d2]$$

Jensen's inequality will allow us to move the log outside the integral in equation 3.d2. The integral of the density function is equal to 1. This means that the log of the integral is 0. This will provide the lower bound of KL when:

$$KL(p||q) \geq 0$$

When

$$p(x) = q(x)$$

From Jensen's inequality, we know that given a convex function $f(\cdot)$ and a random variable X, then:

$$f(E[X]) \leq E[f(X)]$$

For continuous densities, this is equivalent to the following:

$$f(\int xp(x)dx) \leq \int f(x)p(x)dx$$

Since a log function is convex, we can apply Jensen's inequality to KL lower bound to prove it's non negativity.

$$KL(p||q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

$$KL(p||q) = -\int p(x)\log\left(\frac{q(x)}{p(x)}\right)dx$$

$$KL(p||q) = -\int p(x)[\log q(x) - \log p(x)]\,dx$$

$$\geq -log\int q(x)dx$$

$$= 0$$

From the equation above, we have proven that KL(p||q) is non-negative. To recap the steps, we flipped the fraction so that the $p(x)$ term s cancel. Then we applied Jensen's inequality. Finally, we know that log(1) = 0.

e)The information gain or mutual information between X and Y is $\boldsymbol{I(Y;X) = H(Y) - H(Y|X)}$. Show that:

$$\boldsymbol{I(Y;X) = KL(p(x,y)||p(x)p(y))},$$

Where p(x) is the marginal distribution of X and p(y) is the marginal distribution of Y

We can rewrite $I(Y;X)$ by the following equation:

$$I(Y;X) = \sum_{x,y} P(x,y)\log\frac{P(x,y)}{P(x)P(y)}\ [3.e1]$$

We know the following:

$$KL(p||q) = \sum_{x} p(x)\log_2\frac{p(x)}{q(x)}\ [3.e2]$$

Using conditional distributions, we can also write the following:

$$P(x,y) = P(x|y)P(y)\ [3.e3]$$

Substituting [3.e3] into [3.e1] we get:

$$I(Y;X) = \sum_{x}\sum_{y} P(x|y)P(y)log\frac{P(x|y)P(y)}{P(x)P(y)}$$

$$= \sum_{y} P(y)\sum_{x} P(x|y)log\frac{P(x|y)}{P(x)}$$

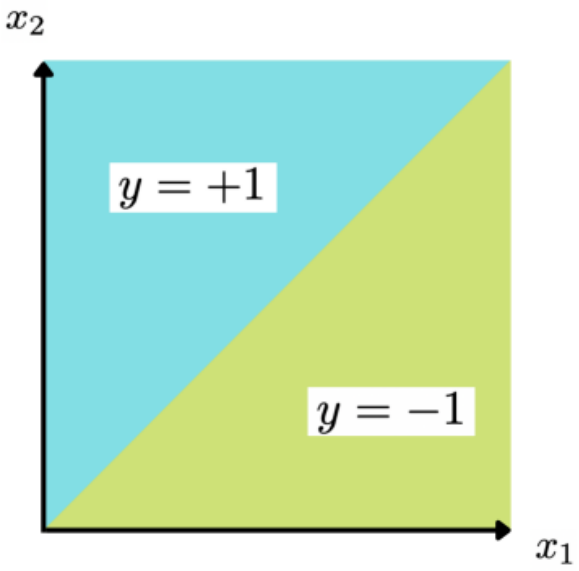
$$= \sum_{y} P(y)KL(p(x|y)||P(x))$$

Similarly, this equation could have been written for x

$$I(Y;X) = \sum_x P(x)KL(p(y|x)||P(y))$$

Question 4 Approximation Error in Decision Trees – 10 pts

We will study how we can use a decision tree to approximate a function and how the quality of the approximation improves as the depth increases. Consider the function $f^*: [0,1]^2 \rightarrow \{-1, +1\}$ as visualized below. This function takes the value of +1 on the upper left triangle and -1 on the lower right triangle.



We would like to approximate this function f^* using a decision tree with the maximum depth of d. We denote the best approximation with depth d as f_d.

a) [2 pts] Explain why f_d with a finite d cannot represent f^* exactly.

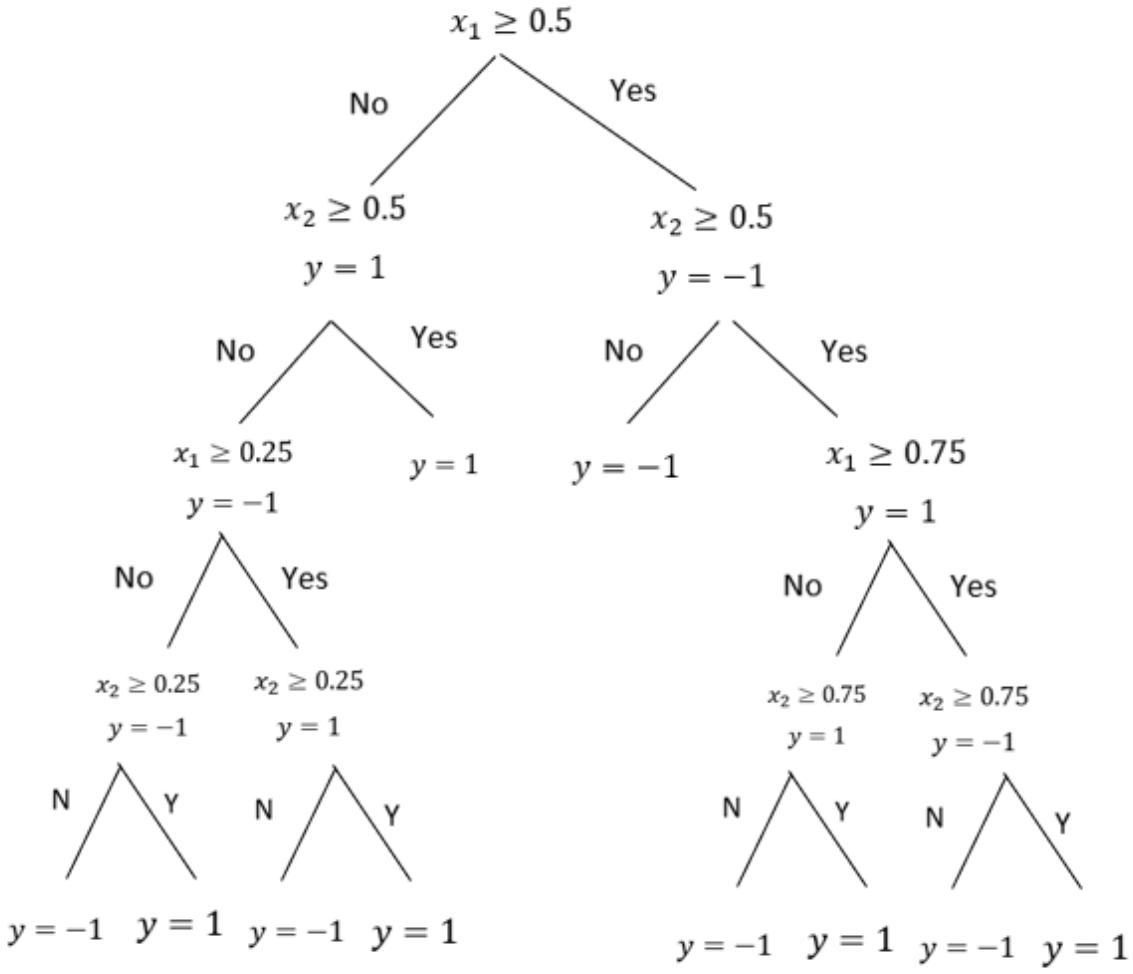
$\boldsymbol{f_d}$ with a finite d cannot represent $\boldsymbol{f^*}$ exactly. The geometry of the function makes it that in some region there is always going to be the two categories in one decision. The error will diminish as d increments and will approach 0, but it will not represent it exactly. We found in the following questions that the function that represent the error is

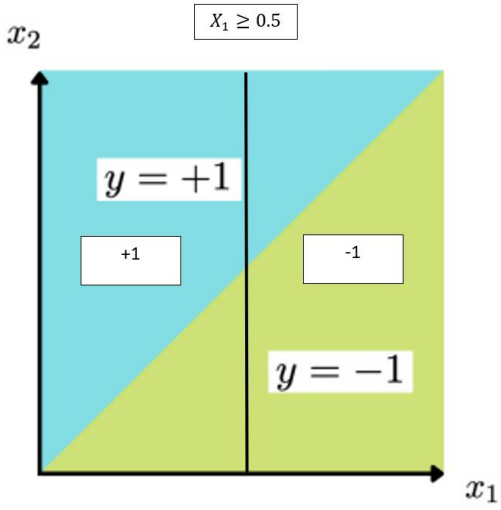
$$e_d = \frac{1}{2^{\frac{d}{2}+1}}, \mathrm{d} = 2\mathrm{k} \text{ with } \mathrm{k} \in \mathrm{N}$$

From this equation, we can see the there is always going to be an error, but it will approach 0 as N becomes larger. Another explanation as to why we cannot represent f^* exactly with a finite number of f_d is that the regions are separated parallel to both axes. However, the function splits the area at a 45-degree angle. Hence, there will always be some error.

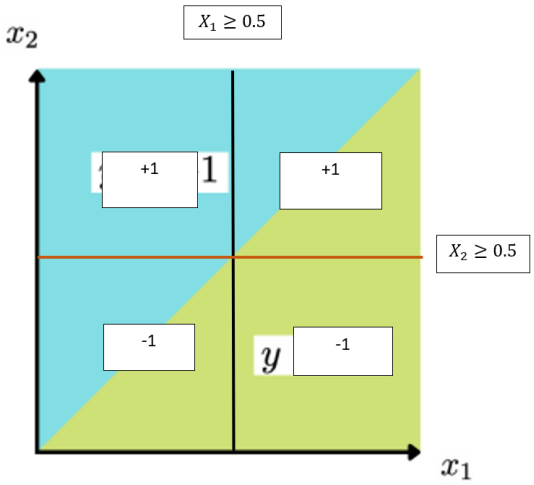
b) [2 pts] Show what f_4 is. You should draw the tree and include all the relevant information such as the attributes at each node, the splitting threshold, and the value of the leaves. You also need to show the regions that it induces.

The following figures illustrates how the function is being split at each level.

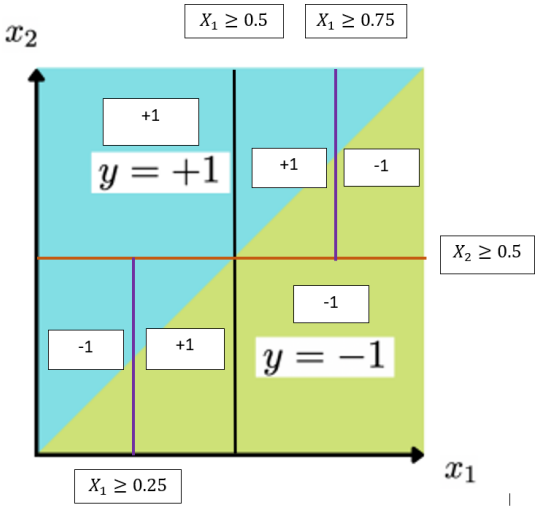




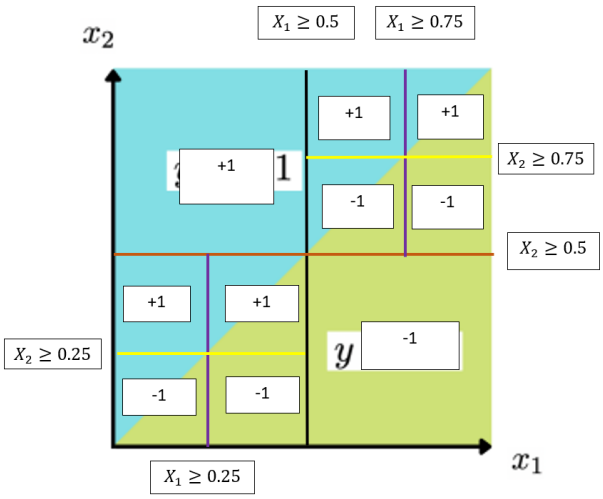
Level 1



Level 2



Level 3



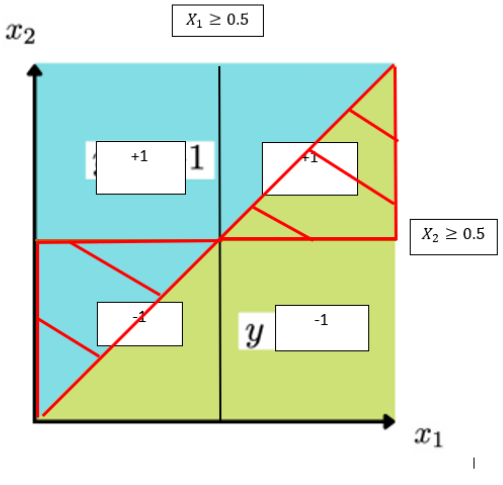
Level 4

c) [2 pts] What is the value of e_2 and e_4?

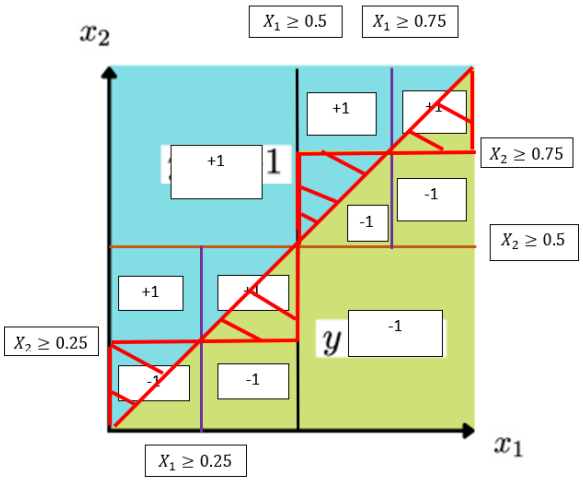
Using the following equation, we can calculate the error of e_2 and e_4:

$$e_d = \int_{[0,1]^2} \prod \{f_d(x) \neq f^*(x)\} dx$$

Looking at the illustration in part b, we can calculate the area of f_d that is different from f^*. For e_2 there are two regions that have some error (Top right corner and bottom left corner). The following image represents the area from the function f_d that is different from f^* for f_2.



Error for f_2



Error for f_4

From this image e_2 can be calculated with the following equation:

$$e_2 = \frac{1}{8} + \frac{1}{8} = \frac{1}{4}$$

Similarly, e_4 can be calculated in the following way.

$$e_4 = \frac{1}{32} + \frac{1}{32} + \frac{1}{32} + \frac{1}{32} = \frac{1}{8}$$

To recap, $e_2 = \frac{1}{4}$ and $e_4 = \frac{1}{8}$.

d) [2 pts] Provide the formula for e_d for any even d, that is, d = 2k with k ∈ N. You need to justify your formula, but your justification does not need to be detailed.

By induction the formula for e_d for any $d = 2k$ with $k \epsilon N$ was found and can be expressed with the following equation

$$\boldsymbol{e_d = \frac{1}{2^{\left(\frac{d}{2}+1\right)}}, \mathrm{d} = 2\mathrm{k} \text{ with } \mathrm{k} \in \mathrm{N}}$$

This equation can easily be seen from the illustration from the previous questions and the calculated error value. Originally, I thought the following formula represented e_d

$$e_d = \frac{1}{2 * d}, \mathrm{d} = 2\mathrm{k} \text{ with } \mathrm{k} \in \mathrm{N}$$

However, upon inspection, this formula does not work for $e_6 = \frac{1}{16}$. The error formula can be explained by the following. Every time the tree goes down two depths, it creates 2 time more areas that have errors in them. However, one single area from two levels higher is 4 times bigger than the smaller area.

Looking at e_2 and e_4 we can see the pattern. e_2 has 2 areas in which both are $\frac{1}{8}$, e_4 has 4 areas with each single one representing $\frac{1}{32}$. Without drawing the next depth, we can write e_6, e_8, e_{10} by the following:

$$e_2 = 2 * \left(\frac{1}{8}\right) = \frac{1}{4}$$

$$e_4 = 4 * \left(\frac{1}{32}\right) = \frac{1}{8}$$

$$e_6 = 8 * \left(\frac{1}{128}\right) = \frac{1}{16}$$

$$e_8 = 16 * \left(\frac{1}{512}\right) = \frac{1}{32}$$

From this, we can see that the function provided to calculate the error applies.

Question 5

The full code will be attached in hw1_code.py

a) [10 pts] Write a function load_data which loads the data, preprocesses it using a vectorizer, and splits the entire dataset randomly into 70% training, 15% validation, and 15% test examples. You may use train_test_split function of scikit-learn within this function.

The following figure is the code to load, preprocess the data and randomly split it into 70% training, 15% validation, and 15% test.

```
[1]  from google.colab import files
     import pandas as pd
     import numpy as np
     from sklearn import feature_extraction
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn import metrics
```

```
[2]  uploaded = files.upload()
```

Choose Files 2 files

- **DNE_climate.csv**(text/csv) - 123377 bytes, last modified: 9/30/2022 - 100% done
- **exists_climate.csv**(text/csv) - 335970 bytes, last modified: 9/30/2022 - 100% done

```
Saving DNE_climate.csv to DNE_climate.csv
Saving exists_climate.csv to exists_climate.csv
```

```
[3]  def load_data():
       DNE_climate = pd.read_csv("DNE_climate.csv", header=0)
       DNE_climate["label"] = 0 #"DNE"
       exists_climate = pd.read_csv("exists_climate.csv", header=0)
       exists_climate["label"] = 1 #"exists"
       corpus = pd.concat((exists_climate, DNE_climate))
       vectorizer = feature_extraction.text.CountVectorizer()
       tweets = corpus["tweet"].to_numpy()
       X = vectorizer.fit_transform(tweets)

       X = X.toarray()
       Y = corpus["label"].to_numpy()
       X_train, X_val_test, y_train, y_val_test = train_test_split(X, Y, test_size=0.3)
       X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_size=0.5)
       return X_train, X_val, X_test, y_train, y_val, y_test, vectorizer.get_feature_names_out()
```

```
     X_train, X_val, X_test, y_train, y_val, y_test, labels = load_data()
```

b) [10 pts] (Decision Tree) Write a function select_tree_model that trains the decision tree classifier using at least 5 different sensible 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.

The following is the code for the function select_tree_model and its accuracy is shown underneath. The chosen values for max_depths was 2,5,10,15, and 20. The two criteria chosen was gini and entropy.

```
[5] def calculate_accuracy(X_val, y_val, model_predict_fn):
      y_pred = model_predict_fn(X_val)
      matches = [1 if val==pred else 0 for val, pred in zip(y_val, y_pred)]
      total_correct = sum(matches)
      total = len(y_val)
      return total_correct/total
```

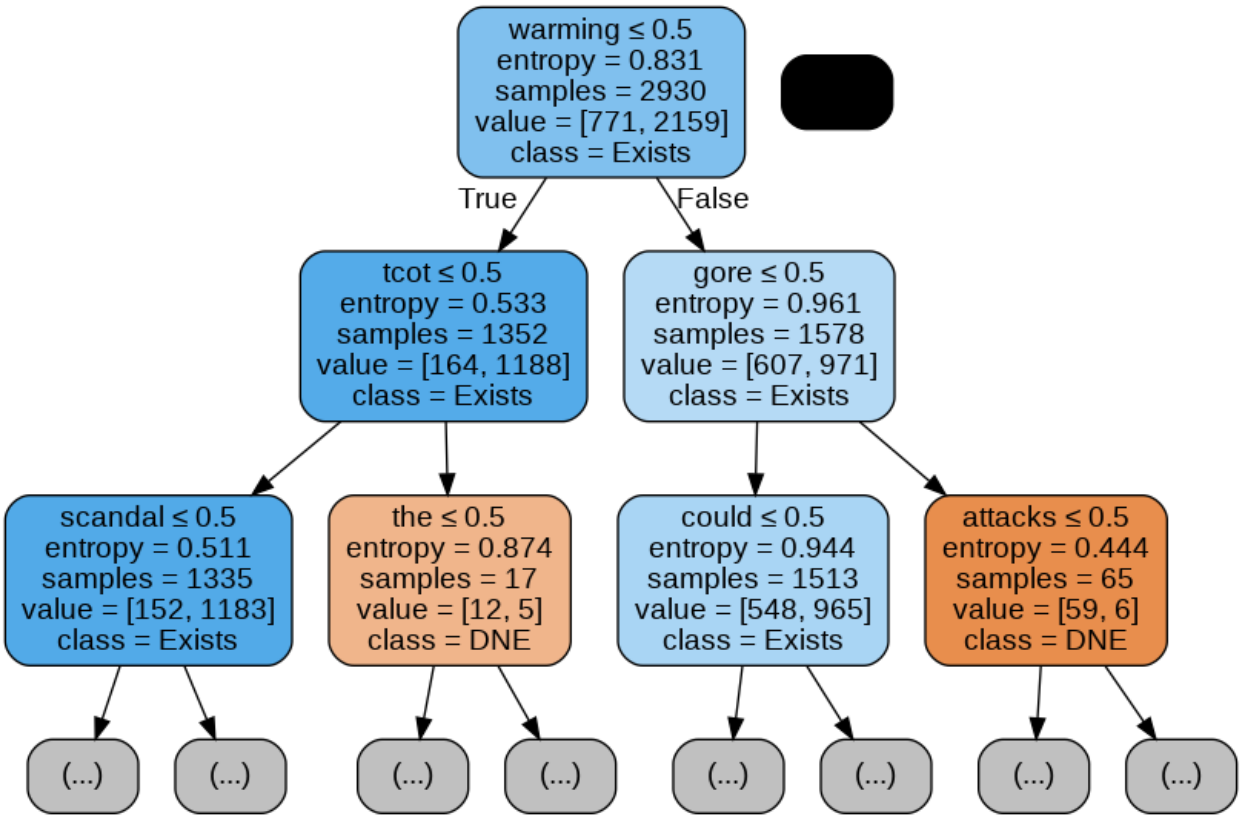
```
def select_tree_model(X_train, X_val, y_train, y_val):
  max_depths = [2, 5, 10, 15, 20]
  critera = ['gini', 'entropy']
  best_d = 0
  best_c = ''
  best_acc = 0
  best_model = None
  for d in max_depths:
    for c in critera:
      model_name = f"Critera={'Information Gain' if c=='entropy' else 'Gini Coefficient'}, Depth={d} model"
      classifier = DecisionTreeClassifier(criterion=c, max_depth=d)
      classifier.fit(X_train, y_train)
      y_pred = classifier.predict(X_val)

      acc = calculate_accuracy(X_val, y_val, classifier.predict)

      if acc > best_acc:
        best_d = d
        best_c = c
        best_acc = acc
        best_model = classifier
      print(f"{model_name:<45} Accuracy={acc*100:.2f}%")
  return best_model
```

```
model = select_tree_model(X_train, X_val, y_train, y_val)
```

```
Critera=Gini Coefficient, Depth=2 model       Accuracy=75.48%
Critera=Information Gain, Depth=2 model       Accuracy=75.48%
Critera=Gini Coefficient, Depth=5 model       Accuracy=76.11%
Critera=Information Gain, Depth=5 model       Accuracy=76.59%
Critera=Gini Coefficient, Depth=10 model      Accuracy=77.55%
Critera=Information Gain, Depth=10 model      Accuracy=77.23%
Critera=Gini Coefficient, Depth=15 model      Accuracy=80.41%
Critera=Information Gain, Depth=15 model      Accuracy=80.73%
Critera=Gini Coefficient, Depth=20 model      Accuracy=80.10%
Critera=Information Gain, Depth=20 model      Accuracy=80.25%
```

(c) [10 pts] (Decision Tree) Now let's stick with the hyperparameters which achieved the highest validation accuracy. Report its accuracy on the test dataset. Moreover, extract and visualize the first two layers of the tree. Your visualization may look something like what is shown below, but it 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.

With the hyperparameters which achieved the highest validation accuracy, the accuracy on the test set was 0.7726 = 77.26%. The figure underneath illustrates the first two layers of the tree and its code.



```
[8] # Calculate accuracy on the test set
    calculate_accuracy(X_test, y_test, model.predict)
```

```
0.7726550079491256
```

```
!pip install graphviz
!pip install pydotplus
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pydotplus in /usr/local/lib/python3.7/dist-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from pydotplus) (3.0.9)
```

```
[10] from sklearn.tree import export_graphviz
     from six import StringIO
     from IPython.display import Image
     import pydotplus

     dot_data = StringIO()
     export_graphviz(model, out_file=dot_data,
                     filled=True, rounded=True,
                     special_characters=True,
                     feature_names = labels,
                     class_names=['DNE','Exists'],
                     max_depth=2)
     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
     graph.write_png('cls.png')
     Image(graph.create_png())
```

(d) [10 pts] (Decision Tree) Write a function compute_information_gain which computes the information gain of a split on the training data. That is, compute I(Y, xi), where Y is the random variable signifying whether the tweet is climate change asserting or denying, and xi is the keyword chosen for the split. Your split should be based on whether the keyword xi exists (True) or does not exist (False). You should ignore the number of times that the keyword appears in the sentence. Report the outputs of this function for the topmost split from the previous part, and for several other keywords.

The function compute_infomration_gain can be found underneath with the output of the function. From the output, two words have significant higher information gain. Those two are warming with the highest information gain of 0.067 followed by scam with 0.015.

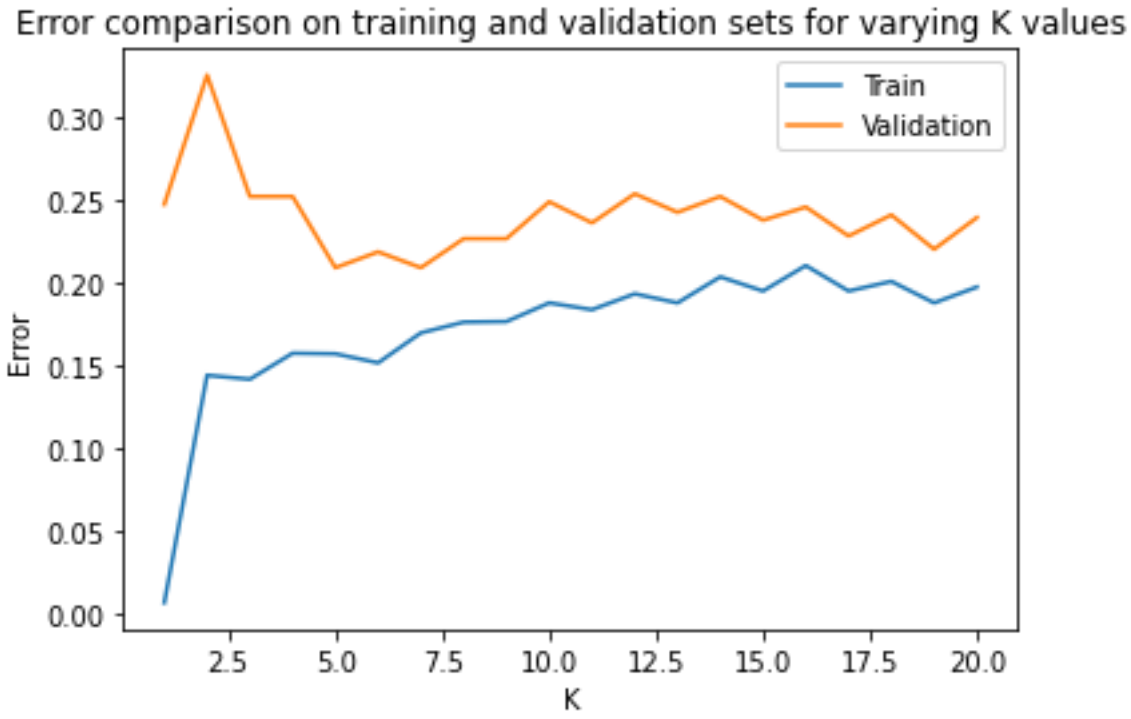
```
[11] import math
     def calculate_entropy(data):
       counts = np.bincount(data)
       probabilities = counts / len(data)
       entropy = 0
       for prob in probabilities:
           if prob > 0:
               entropy += prob * math.log(prob, 2)
       return -1*entropy

     def compute_information_gain(X_train, y_train, split_word):
       split_word = np.where(labels == split_word)[0][0]
       original_entropy = calculate_entropy(y_train)
       left_indices = [i for i, tweet in enumerate(X_train) if tweet[split_word] == 0]
       left_split = y_train[left_indices]
       right_indices = [i for i, tweet in enumerate(X_train) if tweet[split_word] >= 1]
       right_split = y_train[right_indices]
       new_entropy = 0
       for split in [left_split, right_split]:
           prob = len(split)/len(y_train)
           new_entropy += prob * calculate_entropy(split)
       return original_entropy-new_entropy
```

```
import random
words = ["warming", "scam", "thought", "great", "disprove"]
for _ in range(10):
  words.append(random.choice(labels))
for word in words:
  print(f"{word:<20} IG: {compute_information_gain(X_train, y_train, word):.10f}")
```

```
warming              IG: 0.0677538939
scam                 IG: 0.0152455979
thought              IG: 0.0005817005
great                IG: 0.0048208123
disprove             IG: 0.0012047571
asui2d               IG: 0.0001503841
timecomhttp          IG: 0.0000000000
auizpz               IG: 0.0001503841
d31puxtt             IG: 0.0001503841
considers            IG: 0.0001503841
11n727               IG: 0.0001503841
looooooooooooooooooooong IG: 0.0006576067
alumnus              IG: 0.0001503841
djovci               IG: 0.0000000000
free                 IG: 0.0004535568
```

e) ***[10 pts] (KNN) Write a function select_knn_model that uses a KNN classifier to classify between climate change asserting or denying tweets. Use a range of k values between 1 to 20 and compute both training and validation errors. You should generate a graph similar to the one on slide 44 of Lecture #1, which is Figure 2.4 of the Elements of Statistical Learning. You do not need to worry about the Bayes error or the Linear classifier in that figure. Report the generated graph in your report. Choose the model with the best validation accuracy and report its accuracy on the test data.***

After running the model for the value of k from 1 to 20, the accuracy on the test set was calculated. An accuracy of 0.777 = 77.7% was obtained. The following figure illustrates the error for the training and validation for each varying K values.



```
[13] def select_knn_model(X_train, X_val, y_train, y_val):
       ks = range(1, 21)
       history = {"k":ks, "train": [], "val":[]}
       best_k = 0
       best_acc = 0
       best_model = None
       for k in ks:
         model_name = f"k={k} model"
         classifier = KNeighborsClassifier(n_neighbors=k)
         classifier.fit(X_train, y_train)
         acc = calculate_accuracy(X_val, y_val, classifier.predict)
         history["val"].append(1-acc)
         acc_train = calculate_accuracy(X_train, y_train, classifier.predict)
         history["train"].append(1-acc_train)
         if acc > best_acc:
           best_k = k
           best_acc = acc
           best_model = classifier
         print(f"{model_name:<10} Accuracy={acc*100:.2f}%")
       return best_model, history
```

```
model, history = select_knn_model(X_train, X_val, y_train, y_val)
```

```
k=1 model  Accuracy=75.32%
k=2 model  Accuracy=67.52%
k=3 model  Accuracy=74.84%
k=4 model  Accuracy=74.84%
k=5 model  Accuracy=79.14%
k=6 model  Accuracy=78.18%
k=7 model  Accuracy=79.14%
k=8 model  Accuracy=77.39%
k=9 model  Accuracy=77.39%
k=10 model Accuracy=75.16%
k=11 model Accuracy=76.43%
k=12 model Accuracy=74.68%
k=13 model Accuracy=75.80%
k=14 model Accuracy=74.84%
k=15 model Accuracy=76.27%
k=16 model Accuracy=75.48%
k=17 model Accuracy=77.23%
k=18 model Accuracy=75.96%
k=19 model Accuracy=78.03%
k=20 model Accuracy=76.11%
```

```
[15] # Calculate accuracy on the test set
     calculate_accuracy(X_test, y_test, model.predict)
```

```
0.7774244833068362
```

[15]
```
# Calculate accuracy on the test set
calculate_accuracy(X_test, y_test, model.predict)
```

```
0.7774244833068362
```

[16]
```
import matplotlib.pyplot as plt
plt.plot(history["k"], history["train"], label= "Train")
plt.plot(history["k"], history["val"], label= "Validation")
plt.xlabel('K')
plt.ylabel('Error')
plt.title('Error comparison on training and validation sets for varying K values')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7e22ac8bd0>
```