Homework 1, Machine Learning, Fall 2022

***IMPORTANT* Homework Submission Instructions**

1. All homeworks must be submitted in one PDF file to Gradescope.
2. Please make sure to select the corresponding HW pages on Gradescope for each question
3. For all coding components, complete the solutions with a Jupyter notebook/Google Colab, and export the notebook (including both code and outputs) into a PDF file. Concatenate the theory solutions PDF file with the coding solutions PDF file into one PDF file which you will submit.
4. Failure to adhere to the above submission format may result in penalties.

All homework assignments must be your independent work product, no collaboration is allowed. You may check your assignment with others or the internet after you have done it, but you must actually do it by yourself. **Please copy the following statements at the top of your assignment file**:

Agreement 1) This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

Agreement 2) I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

1 Concepts of Learning (Ed Tam)

The goal of this question is to get you familiarized with the conceptual taxonomy of different types of machine learning tasks and to get you thinking about how ML could be applied in real life.
Some common types of machine learning tasks/problems are listed below:

- classification
- regression
- ranking
- clustering
- conditional probability estimation
- density estimation
- pattern mining

You are given a list of situations below. Assign one machine learning task from the list above to each situation below.

The situations are deliberately designed to simulate real-life applications and hence are open-ended. For each situation, there will be more than one answer that could be appropriate, depending on your interpretation of the data/task. **You only have to give ONE reasonable answer for each situation to get full credit.** Please limit any justification/explanation to at most two sentences.

1. You love betting on horse races. One popular betting mechanism is that if you guess the 1st, 2nd and 3rd place horses in a race correctly, you will win a grand prize. You have features about each horse, and would like to train a ML algorithm to create bets that would maximize the chances of winning a grand prize.

   Regression: Assuming the dataset has a label of the time it took for each horse to finish the race given the presence of other features, you could use a regression model to predict the time it takes for each horse to finish, and then use the predicted times to predict the finishing order of 1st, 2nd, and 3rd.

2. You are an astronomer studying galaxies. You have used the James Webb space telescope to collect 10000 images of different galaxies. Galaxies can be of 4 different types: spiral, elliptical, peculiar and irregular. You hand labelled 100 of these images yourself using your expert knowledge, and you would like an ML algorithm to distinguish what types of galaxies the remaining images correspond to.

   Classification: This can be constructed as a multiclass problem (spiral, elliptical, peculiar and irregular).

3. You are a labor economist interested in having an ML algorithm being able to predict individuals' post-college incomes. You have a dataset on 1000 people, with data on each person's characteristics (e.g., educational level, what state they are from) and their starting salary in their first post-college job.

   Regression: You are predicting a real value (post-college income).

4. You are the owner of a restaurant that is famous for your vegetable soup. You are trying to determine how many pounds of vegetables to buy for next week. If you buy too much, the leftover vegetables go to waste. If you buy too little, you will run out of vegetables prematurely. You have a database that contains data about all past weeks (whether there's a holiday coming up, what the weather is likely to be, and how many customers you had that week, etc.).

   Regression: You are predicting a real value (pounds of vegetables to buy).

5. You are a product analyst at Forever 21. You discover that different products sell at drastically different rates depending on the current fashion trends and the current season. You would like to use the available sales data to uncover some of the main current trends and correlations between different clothing products.

   Pattern mining: You are trying to find correlations and trends between clothing sold and the given season.

6. You are a UX researcher at a social network company. You are wondering whether the introduction of a new option will affect how much time users spend watching videos on the platform. In particular, you would like to learn about how the introduction of this new option will change the mean, kurtosis, and variance of the video-watch-time distribution. You start collecting data after the new option is implemented.

   Density estimation: You are trying to find information about the change in mean, kurtosis, and variance in the distribution of the dataset.

7. You are a salesperson that needs to pursue a set of customers. You have features about each of them, and would like to pursue them according to how likely it is that you will make a sale.

   Conditional Probability: You can frame the problem as assigning each customer a probability of their likelihood of purchasing, and then rank them from highest to lowest and pursue them in that order.

8. You are a mortgage specialist at Fannie Mae. You are trying to decide whether to extend a mortgage to a company for them to buy an office. You have a model that determines the likelihood of the company defaulting. You have some newly collected data which shows that the company has had a steady stream of cash flow for the past 5 years and has never defaulted on any loans before. In light of this new information, you would like the model to update its estimate of default.

   Conditional probability estimation: You are trying to assign a probability to a certain event (a given company defaulting).

9. You are a radiologist working on applying ML research in assisting medical diagnosis. In particular, you would like to develop an ML algorithm that can take in MRI scans of individual patients and output how likely it is that the patient is suffering from a malignant tumor.

   Conditional probability estimation: You are trying to assign a probability that a given classification is true (a patient has a malignant tumor).

10. You are a biologist studying the genetic lineage of different species. You have genomic data (a genome is the collection of all genes/genetic materials present in an organism) on 100 different species, and you would like group these species according to their "genetic distance" from one another.

    Clustering: You are trying to group species by genetic distance.

2 Information Theory (Ed Tam)

To understand how decision trees work, it helps to internalize some fundamentals of information theory. In particular, the notion of "information gain" that is used in decision trees (see Decision Tree Lecture Notes for reference) is closely related to the notion of Kullback-Leibler divergence (KL-Divergence), entropy and mutual information in information theory. In the subquestions below, we will explore some of these connections.

Consider a discrete random variable X with distribution P on a set $\mathcal{O}$ of all possible outcomes that X can take. We use $P(a)$ to denote the probability $\text{Prob}(X = a)$ for any $a \in \mathcal{O}$.

Suppose we are given another discrete random variable Y with distribution Q on the same set $\mathcal{O}$ of outcomes. We use $Q(b)$ to denote the probability $\text{Prob}(Y = b)$ for any $b \in \mathcal{O}$.

Recall from class that the entropy of a single random variable X is defined as

$$H(X) = \sum_{a \in \mathcal{O}} -P(a) \log P(a).$$

We can also define the entropy of pair of random variables jointly. Let X, Y be jointly distributed according to the distribution J. Let $J(a, b)$ denote the probability $\text{Prob}(X = a, Y = b)$ for $a, b \in \mathcal{O}$. The joint entropy of X, Y is defined as

$$H(X, Y) = \sum_{(a,b) \in (\mathcal{O}, \mathcal{O})} -J(a, b) \log J(a, b).$$

We can also define a notion of entropy via conditioning. The conditional entropy of X conditioning on Y is defined as

$$H(X|Y) = \sum_{(a,b) \in (\mathcal{O}, \mathcal{O})} -J(a, b) \log \frac{J(a, b)}{Q(b)}.$$

For simplicity, throughout the question, we can assume that $P(a) > 0$, $Q(a) > 0$ and $J(a, b) > 0$ for any outcomes $a, b \in \mathcal{O}$.

Given P and Q, we can define the KL Divergence between them as:

$$KL(P, Q) = \sum_{a \in \mathcal{O}} P(a) \log \frac{P(a)}{Q(a)}.$$

In other words, it is a function that takes in two distributions as input, and returns a real number as output.

You can assume that all logarithms in the question are natural.

2.1

Show that the KL divergence between any discrete distributions P and Q on $\mathcal{O}$ is always a non-negative quantity.

(Hint 1: Jensen's inequality says that $\mathbb{E}(f(X)) \geq f(\mathbb{E}(X))$ for any convex function f, where $\mathbb{E}$ denotes the expectation.

(Hint 2: Note that $-\log$ is a convex function.)

$$KL(P, Q) := \sum_{a \in O} P(a) \log \frac{P(a)}{Q(a)}$$

Say $f(X) = \log \frac{P(X)}{Q(X)} = -\log \frac{Q(X)}{P(X)}$ where $f(x) = -\log X$ and $X = \frac{Q(X)}{P(X)}$

For the right side of Jensen's:

$$f(E(X)) = -\log(E(\frac{Q(X)}{P(X)})) = -\log(\sum_{x \in O} \frac{Q(X)}{P(X)} P(X)) = -\log(\sum_{x \in O} Q(X)) = -\log(1) = 0$$

So we can update Jensen's to $E(f(X)) \geq 0$
For the left side of Jensen's:

$$E(f(X)) = E(-\log \frac{Q(X)}{P(X)}) = \sum_{x \in O} -\log \frac{Q(X)}{P(X)} P(X) = \sum_{x \in O} P(X) \log \frac{P(X)}{Q(X)} = KL(P, Q)$$

Therefore, $KL(P, Q) \geq 0$

2.2

Show that the KL Divergence between P and Q is equal to 0 if P and Q are the same distribution. (Note: this implication does not generally hold in the other direction)

$$KL(P, Q) := \sum_{a \in O} P(a) \log \frac{P(a)}{Q(a)}$$

If P and Q are the same distribution, then $\forall a \in O, P(a) = Q(a)$
Therefore,

$$\sum_{a \in O} P(a) \log(1) = \sum_{a \in O} P(a)0 = 0$$

$$KL(P, Q) = 0$$

2.3

Our results in 2.1 and 2.2 suggest that one intuitive way to think about the KL-Divergence is as a sort of "distance" between probability distributions. However, while this intuition is useful, it is well known that the KL-Divergence is not, strictly speaking, a distance. One reason that it is not a distance is that it is not always symmetric, i.e. $KL(P, Q)$ is not necessarily equal to $KL(Q, P)$ in general. A simple example can be shown using Bernoulli distributions. Come up with an example of two Bernoulli distributions and show that the KL between them is asymmetric.
Say

$$P = [\frac{1}{8}, \frac{7}{8}], Q = [\frac{1}{4}, \frac{3}{4}]$$

$$KL(P, Q) = \sum_{a \in O} P(a) \log \frac{P(a)}{Q(a)} = \frac{1}{8} \log \frac{\frac{1}{8}}{\frac{1}{4}} + \frac{7}{8} \log \frac{\frac{7}{8}}{\frac{3}{4}} = 0.0482$$

$$KL(Q, P) = \sum_{a \in O} Q(a) \log \frac{Q(a)}{P(a)} = \frac{1}{4} \log \frac{\frac{1}{4}}{\frac{1}{8}} + \frac{3}{4} \log \frac{\frac{3}{4}}{\frac{7}{8}} = 0.0577$$

2.4

Consider the joint distribution J of X and Y, and also consider the product distribution (which we denote as PQ) of X and Y, which is defined as $PQ(a, b) := P(a)Q(b)$. Show that $KL(J, PQ) = H(X) - H(X|Y)$.

$$KL(J, PQ) := \sum_{a,b \in O,O} J(a, b) \log \frac{J(a, b)}{PQ(a, b)} = \sum_{a,b \in O,O} J(a, b) \log \frac{J(a, b)}{P(a)Q(b)}$$

$$H(X) := \sum_{a \in O} -P(a) \log P(a)$$

$$H(X|Y) := \sum_{a,b \in O,O} -J(a, b) \log \frac{J(a, b)}{Q(b)}$$

Starting with the right side:

$$H(X) = H(X|Y) = \sum_{a \in O} -P(a) \log P(a) - \sum_{a,b \in O,O} -J(a,b) \log \frac{J(a,b)}{Q(b)}$$

$$= \sum_{a \in O} P(a) \log \frac{1}{P(a)} + \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{Q(b)}$$

If you keep a constant in a joint distribution $J(a,b)$ and sum over all b, then you get $P(a)$.

$$\sum_{a,b \in O,O} J(a,b) \log \frac{1}{P(a)} = \sum_{a \in O} \log \frac{1}{P(a)} \sum_{b \in O} J(a,b) = \sum_{a \in O} \log \frac{1}{P(a)} P(a) = \sum_{a \in O} P(a) \log \frac{1}{P(a)}$$

So,

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{1}{P(a)} + \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{Q(b)}$$

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{1}{P(a)} + J(a,b) \log \frac{J(a,b)}{Q(b)}$$

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{P(a)Q(b)} = KL(J, PQ)$$

Therefore, $KL(J, PQ) = H(X) - H(X|Y)$

2.5

Given X and Y, we can define a quantity called the mutual information between X and Y, denoted $I(X,Y)$. One way to define this is as $I(X,Y) := H(Y) - H(Y|X)$. Show that $I(X,Y) = KL(J, PQ)$.

$$I(X,Y) := H(Y) = H(Y|X) = \sum_{b \in O} -Q(b) \log Q(b) - \sum_{a,b \in O,O} -J(a,b) \log \frac{J(a,b)}{P(a)}$$

$$= \sum_{b \in O} Q(b) \log \frac{1}{Q(b)} + \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{P(a)}$$

From the previous question,

$$\sum_{a,b \in O,O} J(a,b) \log \frac{1}{Q(b)} = \sum_{b \in O} \log \frac{1}{Q(b)} \sum_{b \in O} J(a,b) = \sum_{b \in O} \log \frac{1}{Q(b)} Q(b) = \sum_{b \in O} Q(b) \log \frac{1}{Q(b)}$$

So,

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{1}{Q(b)} + \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{P(a)}$$

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{1}{Q(b)} + J(a,b) \log \frac{J(a,b)}{P(a)}$$

$$= \sum_{a,b \in O,O} J(a,b) \log \frac{J(a,b)}{P(a)Q(b)} = KL(J, PQ)$$

Therefore, $I(X,Y) = KL(J, PQ)$

2.6

Our work above has shown that $KL(J, PQ) = H(Y) - H(Y|X) = H(X) - H(X|Y) \geq 0$. Interpreting this in a decision tree context, we can see that splitting on an attribute decreases the entropy at a branch. This is a nice result, but you may wonder how entropy is related to our main goal of learning. It turns out that there is a close connection between entropy and misclassification error in decision trees.

For simplicity, consider a binary classification problem. You are inspecting the training samples at a node. Let p denoted the proportion of samples that are in class "0" at the node, and $1 - p$ denote the proportion of samples that are in class "1". You can treat p as a continuous variable that can range from 0 to 1.

The misclassification error at the node will be

$$\text{Error}(p) = \min(p, 1 - p).$$

The entropy at the node will be

$$H(p) = -p \log p - (1 - p) \log(1 - p).$$

For simplicity (and with no loss of generality), suppose you know that $p < 0.5$. Show that a smaller entropy H corresponds to a smaller misclassification error under this $p < 0.5$ setting.

By taking the derivatives of both the entropy and misclassification equations and showing that they both have a positive derivative, then it would follow that a smaller entropy would be correlated with a smaller misclassification error.

For $p < 0.5 : Error(p) = \min(p, 1 - p) = p$

$$\frac{d}{dp} Error(p)_{p<0.5} = \frac{d}{dp} p = 1$$

$$\frac{d}{dp} H(p) = \frac{d}{dp}(-p \log p - (1 - p) \log(1 - p)) = \log(1 - p) - \log(p)$$

$$\frac{d}{dp} H(p)_{p<0.5} > 0$$

$H(p)$ has one critical point at $p = 0.5$ For $p < 0.5$, $\frac{d}{dp} H(p)$ has a positive value as choosing any arbitrary $p < 0.5$ gives a positive value, and the critical point of $H(p)$ is at $p = 0.5$.

Both $Error(p)$ and $H(p)$ have positive derivatives for any $p < 0.5$. Therefore, a smaller entropy corresponds to a smaller misclassification error.

3 Classifiers and Metrics - Coding (Stark)

Age	likeRowing	Experience	Income	Y
20	1	0	20	0
18	1	1	33	0
11	0	1	21	1
31	0	0	18	1
19	1	1	7	1
21	1	0	10	0
44	1	0	23	1
15	1	1	16	0
16	0	1	15	1
17	1	0	6	0

You are given the dataset above with feature vector $\mathbf{x}$ including Age, likeRowing, Experience and Income, and the binary label Y. You are also given a linear classifier $g(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0$ and a non-linear classifier $f(\mathbf{x}) = \tanh(\boldsymbol{\theta}^\top \mathbf{x} + \theta_0)$, where $\boldsymbol{\theta} = (0.05, -3, 2.1, 0.008)$, $\theta_0 = 0.3$, and "tanh" function $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. (In this question, you are expected to write functions from scratch, but where packages including matplotlib and numpy are allowed.)

(3.1) First calculate the value of $g(\mathbf{x})$ for each data point. What choices of threshold would minimize misclassification error?

The threshold choices of [-0.316, 0.406, 1.994] lead to a misclassification error of 0.2

(3.2) Calculate the value $f(\mathbf{x})$ for each data point. What choice(s) of threshold would minimize misclassification error? Compute the confusion matrix, precision, recall, and F1 score for one such threshold.

The threshold choices of [-0.306, 0.385, 0.964] lead to a misclassification error of 0.2

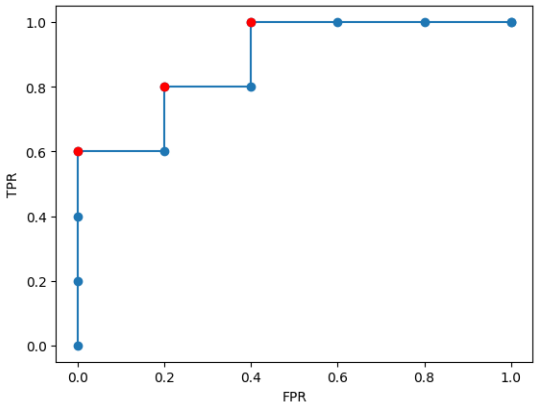
At threshold = -0.306:

	Predicted +	Predicted -
Actual +	5 TP	0 FN
Actual -	2 FP	3 TN

Precision: 0.714; Recall: 1.0; F1: 0.833

(3.3) For classifier $f(\mathbf{x})$, plot the ROC curve. Please plot the ROC curve as a continuous, connected set of lines. Plot the points on the ROC curve that represent decision points with the pminimum classification error.

ROC Curve of f(x) with pminimum classifcation error points in red:

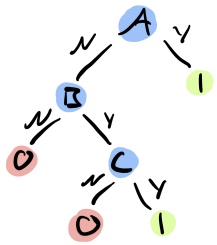


4 Calculate Splits (Stark)

As we learned from class, decision trees can be learned from data. The following table shows a dataset that forms the logical rule $A \vee (B \wedge C)$.

A	B	C	$A \vee (B \wedge C)$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

4.1 Decision trees can be used to represent logical rules. Please draw a decision tree to represent $A \vee (B \wedge C)$, and make sure that your decision tree has the smallest possible number of nodes.



4.2 If you had split according to Gini index, which feature would you choose to split first? (If you encounter multiple features with tied values of Gini index, follow alphabetical order. Please show your steps.)

$$\text{Gini} := 2p(1-p)$$

$$A : G_Y = 0; G_N = \frac{3}{8}; G_W = \frac{1}{2}G_Y + \frac{1}{2}G_N = 0 + \frac{3}{16} = \frac{3}{16}$$

$$B : G_Y = \frac{3}{8}; G_N = \frac{1}{2}; G_W = \frac{1}{2}G_Y + \frac{1}{2}G_N = \frac{3}{16} + \frac{1}{4} = \frac{7}{16}$$

$$C : G_Y = \frac{3}{8}; G_N = \frac{1}{2}; G_W = \frac{1}{2}G_Y + \frac{1}{2}G_N = \frac{3}{16} + \frac{1}{4} = \frac{7}{16}$$

Split over A!

4.3 If you had split according to information gain, is it the same as in 4.2? (If you encounter ties in information gain, follow alphabetical order. Please show your steps.)

$$\text{Info Gain} := H[\frac{\#p}{\#p+\#n}, \frac{\#n}{\#p+\#n}] - \sum_{j=1}^{J} \frac{\#p_j + \#n_j}{\#p+\#n} H[\frac{\#p_j}{\#p_j+\#n_j}, \frac{\#n_j}{\#p_j+\#n_j}]$$

$$H(X) := \sum_{a \in O} -P(a) \log P(a)$$

$$A : H[\frac{5}{8}, \frac{3}{8}] - [\frac{1}{2}H[0,1] + \frac{1}{2}H[\frac{1}{4}, \frac{3}{4}]] = 0.954 - [\frac{1}{2}0 + \frac{1}{2}0.811] = 0.549$$

$$B : H[\frac{5}{8}, \frac{3}{8}] - [\frac{1}{2}H[\frac{1}{2}, \frac{1}{2}] + \frac{1}{2}H[\frac{3}{4}, \frac{1}{4}]] = 0.954 - [\frac{1}{2}1 + \frac{1}{2}0.811] = 0.049$$

$$C : H[\frac{5}{8}, \frac{3}{8}] - [\frac{1}{2}H[\frac{1}{2}, \frac{1}{2}] + \frac{1}{2}H[\frac{3}{4}, \frac{1}{4}]] = 0.954 - [\frac{1}{2}1 + \frac{1}{2}0.811] = 0.049$$

Still split over A!

5 Classification with KNN and Decision Trees (Stark)

5.1 Coding In this problem you will experiment with the "carseat" dataset. This dataset contains 10 features, and 1 target ("Sales"). It is a binary classification task.

5.2 Coding Please use two of the following decision tree packages: sklearn.tree.DecisionTreeClassifier, GOSDT, chefboost, pydl8.5. For each package, utilizing the F1 score function implemented in Q3, report their F1 scores. For each of the trees, please experiment with tuning at least 1 parameter with K-Fold cross validation. (See hints below.)

Tuned *max_depth* parameter for both algorithms.

	F1 Scores	F1 Scores w/ K- Fold CV Tuned Parameter
sklearn Decision Tree	0.578	0.692
chefboost CART	0.643	0.643

5.3 Coding Implement your own KNN algorithm from scratch. It should support at least two different distance metrics (such as Euclidean distance, cosine distance, Manhattan Distance). For the KNN classifier you implemented, tune the number-of-neighbours parameter and choice of distance metric, evaluate them with the F1 score and report the best result in your notebook.

Tuned Number-of-Neighbors, Distance Metric by Grid-search: 1, Manhattan
F1 Score w/ Tuned Parameters: 0.554

5.4 For this dataset, which implementation performed the best?

The tuned sklearn Decision Tree performed the best!
If the chefboost maxdepth parameter actually worked, then maybe that would have been the best!

Hints: Available decision tree implementations include: sklearn CART, GOSDT+guesses (Pypi: "pip install gosdt" for mac/linux, does not yet work on windows), pydl8.5, (other implementations are also acceptable). The most powerful tools are GOSDT and DL8.5 so we suggest trying one of these if you are able to to install it (Sklearn and GOSDT will allow you to calculate tree sparsity). You are encouraged to try out GridSearchCV or other cross-validation parameter search functions, but be sure to implement a version from scratch by yourself for this problem. We have also provided a template, feel free to use and change it. You will have to make adjustments to encoding methods depending on package requirements.

(5.5) What is the difference between K-fold cross-validation and leave-one-out-cross-validation (LOOCV)? What are some of the disadvantages of the LOOCV? Please list two of them.

The difference is in the sizes of the train and validation sets. K-fold is partitioned into K folds (as per the name), while LOOCV is the extreme of K-fold with K equal to the size of the whole training set. Some disadvantages with LOOCV are the computational cost (you have to run a train/valid for every row of data) and when the number of rows is large, it can be time consuming. Also, when the test fold chosen is an outlier point, then the evaluation will be artificially low, dragging down the average performance score.

(5.6) For the carseat dataset, compared to accuracy, do you think F1 is a good evaluation metric for the dataset and why?

I think that F1 is a better evaluation metric than accuracy for the carseat dataset. The purpose is to classify whether someone has a carseat or not; the accuracy measures correct classifications while the F1 score is a better measure which accounts for false positives and negatives. The F1 score gives a more balanced

measure of model performance. The F1 score also does better with imbalanced datasets - which the carseat dataset is slightly.

6 Consistency and Curse of Dimensionality in K-Nearest Neighbors (Ed Tam)

A classical algorithm used in supervised learning (in both regression and classification) is the K-Nearest Neighbor (KNN) Algorithm.

In this question, we will explore some theoretical properties of the KNN algorithm under a toy setup. The goal is to have you gain intuitive understanding on why the KNN algorithm works, and under what situations it will fail to work well.

One way to evaluate whether an algorithm makes sense/work well is to look at what happens when it has access to an infinite amount of training data. Ideally, we would like the algorithm to have 0 prediction error in the limit as the number of training data n approaches infinity. This property is called consistency in statistics. In 6.1 and 6.2, we will guide you through a consistency proof.

Consider the following setting. Suppose we have d binary features. We can write our feature vector then as $\mathbf{x} \in \{0,1\}^d$. We can define distance between feature vectors that measures their "similarity" as:

$$\rho(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} I(\mathbf{x}_i \neq \mathbf{x}'_i).$$

ρ is called the Hamming distance.

Suppose we have training data $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$. Assume that the training features $x_1, x_2, \cdots, x_n$ are drawn I.I.D. from the uniform distribution over the d-dimensional hypercube $\{0,1\}^d$. Assume that the true relationship between the labels (y's) and the features (x's) is given by the formula $y = f(x)$, where f is an unknown but deterministic function that maps from $\{0,1\}^d \to [0,1]$.

Assume that f satisfies the following property

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq L \cdot \rho(\mathbf{x}, \mathbf{x}')$$

where L is a positive constant. Such a function is called a Lipschitz function.

6.1

Consider the simplified case where we run a KNN algorithm with $K = 1$. This means that when presented with an observed test point x_{test}, this algorithm will find the nearest neighbor of x_{test} (i.e., the closest training point) and use the label of $NN(x_{test})$ as the prediction for x_{test}. We will use $NN(x_{test})$ to denote the closest training point of x_{test}.

Show that as $n \to \infty$, $NN(x_{test})$ converges in probability to x_{test} (under the Hamming distance).

(Review: A sequence of random variable Z_n converges in probability to a target T under the Hamming distance ρ if for any $\epsilon > 0$, $\lim_{n\to\infty} P(\rho(Z_n, T) > \epsilon) = 0$. The target T can be a deterministic constant or a random variable.)

$NN(x_{test})$ is the nearest point, out of n points, to x_{test}. All $i \in n$ and x_{test} are drawn IID from $\{0,1\}^d$. So, we can think of $NN(x_{test})$ as a random variable, Z_n. For $\lim_{n\to\infty} P(\rho(Z_n, T) > \epsilon) = 0$, then $NN(x_{test})$

must be the same point as x_{test}.

As all the points are drawn from the same distribution, Then $0 < P(x_{test} \neq Z_n) < 1$, or $P(x_{test} \neq Z_n) = \frac{1}{k}$ where $k \in \mathbb{N}$.

As $n \to \infty$,

$$\lim_{n\to\infty} \prod^{n} P(x_{test} \neq Z_n) = \lim_{n\to\infty} \prod^{n} \frac{1}{k} = 0$$

Therefore, as $n \to \infty$, x_{test} must be drawn. When x_{test} is drawn,

$$\lim_{n\to\infty} \rho(Z_n, x_{test}) = 0$$

Therefore, for any $\epsilon > 0$,

$$\lim_{n\to\infty} P(\rho(Z_n, x_{test}) > \epsilon) = 0$$

Or,

$$\lim_{n\to\infty} P(\rho(NN(x_{test}), x_{test}) > \epsilon) = 0$$

In other words, as $n \to \infty$, $NN(x_{test})$ converges in probability to x_{test}!

6.2

Based on the result in 6.1, argue that the absolute error loss

$$|f(NN(x_{test})) - f(x_{test})|$$

converges to 0 in probability as n goes to infinity.

By definition:

$$|f(x) - f(x') \leq L * \rho(x, x')$$

Or,

$$|f(NN(x_{test})) - f(x_{test}) \leq L * \rho(NN(x_{test}), x_{test})$$

From above,

$$\lim_{n\to\infty} P(\rho(NN(x_{test}), x_{test}) > \epsilon) = 0 \text{ for any } \epsilon > 0$$

As our $NN()$ is set to $K = 1$, that means that for the above statement to be true, the ϵ ball must be small enough to where the only point within the ball is x_{test}. Then, it follows that

$$\lim_{n\to\infty} \rho(NN(x_{test}), x_{test}) = 0$$

Taking the limit of the inequality,

$$\lim_{n\to\infty} |f(x) - f(x')| \leq \lim_{n\to\infty} L * \rho(x, x')$$

$$\lim_{n\to\infty} |f(NN(x_{test})) - f(x_{test})| \leq L * \lim_{n\to\infty} \rho(NN(x_{test}), x_{test})$$

$$\lim_{n\to\infty} |f(NN(x_{test})) - f(x_{test})| \leq L * 0$$

As $|f(NN(x_{test})) - f(x_{test})|$ is an absolute value, can't be negative, and must be less than or equal to 0:

$$0 \geq \lim_{n\to\infty} |f(NN(x_{test})) - f(x_{test})| = 0$$

In other words, $|f(NN(x_{test})) - f(x_{test})|$ converges to 0 as n goes to infinity.

6.3

We have shown in 6.1 and 6.2 that KNN (when $K = 1$) is a consistent algorithm under a regression setting with categorical features. (The argument is very similar for general K if you are interested).

KNN is a powerful algorithm, and it can be often be used in a latent representation setting in conjunction with other models like neural networks to tackle difficult, high-dimensional machine learning problems such as image classification. However, KNN is not directly useful for high dimensional problems in the original feature space. The reason is that the KNN algorithm suffers from the curse of dimensionality, i.e., when the dimension d of the features is big, the algorithm performs poorly. Below, we will do some quick math to gain intuition on why this is true.

We are given an observed test point on the d-dimensional hypercube $\mathbf{x}_{test} \in \{0,1\}^d$. One natural way to think about nearest neighbor algorithms is to consider any point within some Hamming distance r of $\mathbf{x}_{test}$ as a reasonable neighbor of $\mathbf{x}_{test}$. This can be formalized via the notion of a Hamming neighborhood around $\mathbf{x}_{test}$, denoted as $N_r(\mathbf{x}_{test}) := \{\mathbf{x} \in \{0,1\}^d : \rho(\mathbf{x}_{test}, \mathbf{x}) \leq r\}$.

You are given n points $x_1, x_2, \cdots, x_n$ drawn I.I.D. from the uniform distribution on the d-dimensional hypercube $\{0,1\}^d$. Let K' denote the expected number of points that will lie within the Hamming neighborhood $N_r(\mathbf{x}_{test})$. For simplicity, assume $r = 2$. Find an expression of K' in terms of d and n.

$$K' = E(|N_r(x_{test})|) = \sum^{n} 1 * P(N_r(x_{test}))$$

Deriving $P(N_r(x_{test}))$:
When $r = 0$:

$$\text{By definition, this must be } x_{test}; |N_{r=0}(x_{test})| = 1$$

When $r = 1$:

$$\text{Hamming distance of 1, only one different binary dimensional value;} |N_{r=1}(x_{test})| = \binom{d}{1} = d$$

When $r = 2$:

$$\text{Hamming distance of 2, only one different binary dimensional value;} |N_{r=2}(x_{test})| = \binom{d}{2}$$

$$P(N_r(x_{test})) = \frac{\text{Number of points in } N_r(x_{test}}{\text{Total number of possible points}} = \frac{1 + d + \binom{d}{2}}{2^d}$$

Therefore,

$$K' = \sum^{n} P(N_r(x_{test})) = \sum^{n} \frac{1 + d + \binom{d}{2}}{2^d} = n(\frac{1 + d + \binom{d}{2}}{2^d})$$

6.4

For fixed sample size n, what happens to K' as $d \to \infty$? Considering your solution to 6.3 as a fraction, how fast (polynomial or exponential) does the numerator grow as you let d go to infinity? How fast does the denominator grow?

$$\lim_{d \to \infty} K' = \lim_{d \to \infty} n(\frac{1 + d + \binom{d}{2}}{2^d}) = 0$$

The numerator grows polynomially; the denominator grows exponentially.

6.5

Based on your answer to 6.4, give an intuitive explanation for why KNN does not work well in high dimensions in no more than three sentences.

As the number of dimensions gets higher, the $E(|N_r(x_{test})|)$ for a given r gets exponentially lower. In order to get the K nearest neighbors, r has to get higher, meaning the ϵ ball gets larger. This increases the probability of misclassification as the probability that nearest neighbor is x_{test} gets exponentially lower.

This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

CS 671: Homework 1

Alex Kumar

Question 3

In [ ]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
# Define global variables
DATA = np.array([[20, 1, 0, 20, 0],
                 [18, 1, 1, 33, 0],
                 [11, 0, 1, 21, 1],
                 [31, 0, 0, 18, 1],
                 [19, 1, 1, 7, 1],
                 [21, 1, 0, 10, 0],
                 [44, 1, 0, 23, 1],
                 [15, 1, 1, 16, 0],
                 [16, 0, 1, 15, 1],
                 [17, 1, 0, 6, 0]])
COEF = np.array([0.05, -3, 2.1, 0.008])
K = 0.3
```

In [ ]:
```
# Base functions
def g(x):
    return np.dot(COEF, x) + K

def f(x):
    return np.tanh(g(x))
```

In [ ]:
```
# Data Helper Functions

# Calculate data values
def getData():
    gs, fs, ys = [], [], []
    for x in DATA:
        ys.append(x[-1])
        gs.append(round(g(x[:-1]), 3))
        fs.append(round(f(x[:-1]), 3))
    return gs, fs, ys

# Make matrix with function and y values
def makePairs(x, y):
```

```
    x, y = np.vstack(x), np.vstack(y)
    return np.concatenate((x, y), axis=1)

# Sort A by f(x) values
def sorted(a):
    temp = a.view(np.ndarray)
    a_sort = temp[np.lexsort((temp[:, 0],))]
    return a_sort, a_sort[:, 0]
```

In [ ]:
```
# Threshold Helper Functions

# Return list of thresholds to iterate over
def makeThresh(thresh):
    thresh = np.insert(thresh, 0, thresh[0]-1)
    thresh = np.insert(thresh, len(thresh), thresh[-1]+1)
    return thresh

# Calculate misclassification error for each thresh over the sorted dat
def find_best(a, thresh):
    error, total = [], len(a[:,1])
    for t in thresh:
        misclass = 0
        for x in a:
            if x[0] >= t:       # class pos
                if x[1] != 1:   misclass += 1
            else:               # class neg
                if x[1] != 0:   misclass += 1
        error.append(misclass/total)
    return np.concatenate((np.vstack(thresh), np.vstack(error)), axis=1)

# Find threshold(s) with lowest misclassification error
def find_min(a):
    curr_min, best = 1, []
    for x in a:
        if x[1] < curr_min:
            curr_min = x[1]
            best = [x[0]]
        elif x[1] == curr_min:
            best.append(x[0])
    return best, curr_min
```

In [ ]:
```
# Evalutation Helper Functions

# Calculate confusion matrix for best threshold(s)
def confusionMatrix(a, t):
    confusion = [0, 0, 0, 0]    # [TP, FP, FN, TN]
    for x in a:
        if x[0] >= t:       # class pos
                if x[1] != 1:   confusion[1] += 1   # FP
                else:           confusion[0] += 1   # TP
        else:               # class neg
            if x[1] != 0:       confusion[2] += 1   # FN
            else:               confusion[3] += 1   # TN
    return confusion

# Precision
def calcPrecision(c):
```

```
    return c[0] / (c[0] + c[1])     # [TP, FP, FN, TN]

# Recall
def calcRecall(c):
    return c[0] / (c[0] + c[2])

# F1
def calcF1(c):
    p = calcPrecision(c)
    r = calcRecall(c)
    return 2 * ((p * r) / (p + r))
```

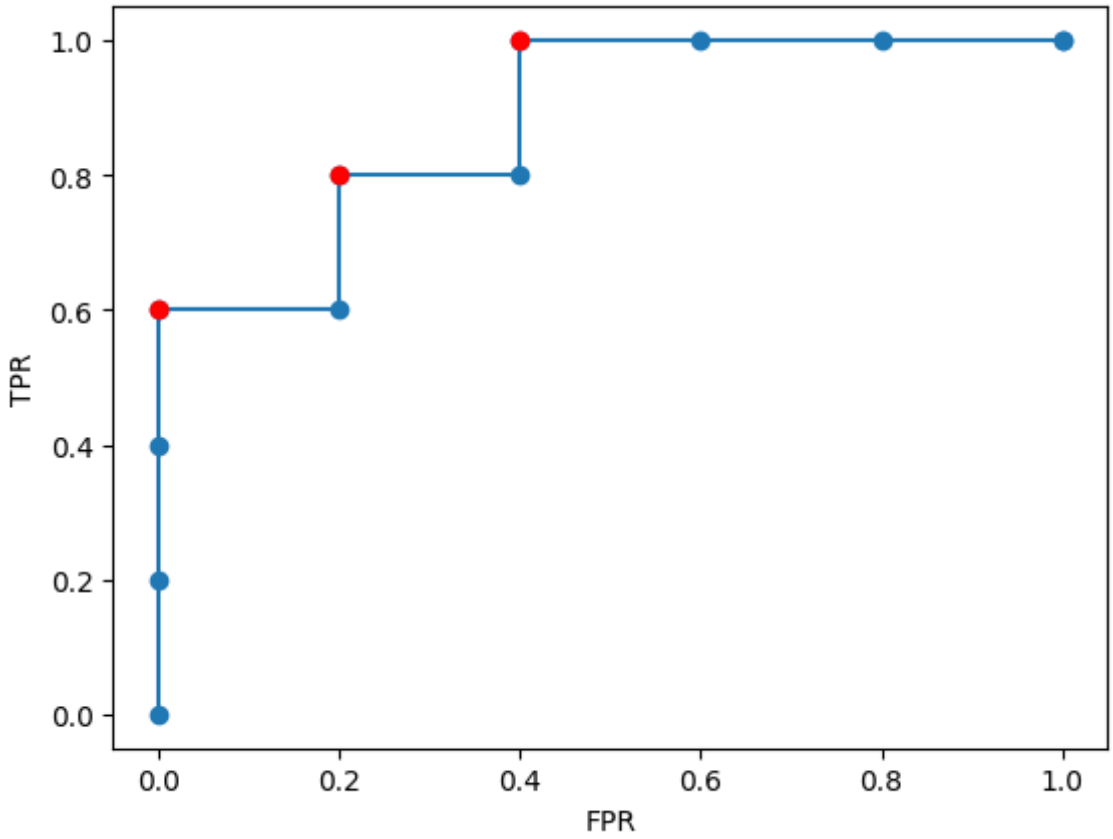
In [ ]:
```
# ROC Helper Functions

# Find TPR and FPR for each threshold
def findRates(a, thresh):
    tprs, fprs = [], []
    for t in thresh:
        confusion = confusionMatrix(a, t)
        tprs.append(calcRecall(confusion))
        fprs.append(confusion[1] / (confusion[1] + confusion[3]))
    return np.concatenate((np.vstack(thresh), np.vstack(tprs), np.vstack(fprs))

# Generate ROC Curve
def makeROC(rates, min_ts):
    plt.plot(rates[:, 2], rates[:, 1], "-o")
    for p in rates:
        if p[0] in min_ts:
            plt.plot(p[2], p[1], "ro")
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.show()
    return
```

In [ ]:
```
# Subroutine that runs for g(x) and f(x)
def subroutine(x, y):
    xy = makePairs(x, y)                            # prep data
    x_sort, x_thresh = sorted(xy)                   # sort data
    x_thresh = makeThresh(x_thresh)                 # make thresholds
    x_error = find_best(x_sort, x_thresh)           # get misclass error forall
    x_min, min_val = find_min(x_error)              # find best t's
    print("Min thresh: ", x_min, "---> @ error of: ", min_val)

    c_matrix = confusionMatrix(x_sort, x_min[0])    # ex confus matrix
    precision = calcPrecision(c_matrix)
    recall = calcRecall(c_matrix)
    f1 = calcF1(c_matrix)
    print("[TP, FP, FN, TN]: ", c_matrix)
    print("Precision: ", round(precision, 3), "  |  Recall: ", round(recall, 3)

    roc_matrix = findRates(x_sort, x_thresh)        # tpr and fpr forall t
    makeROC(roc_matrix, x_min)
    return
```

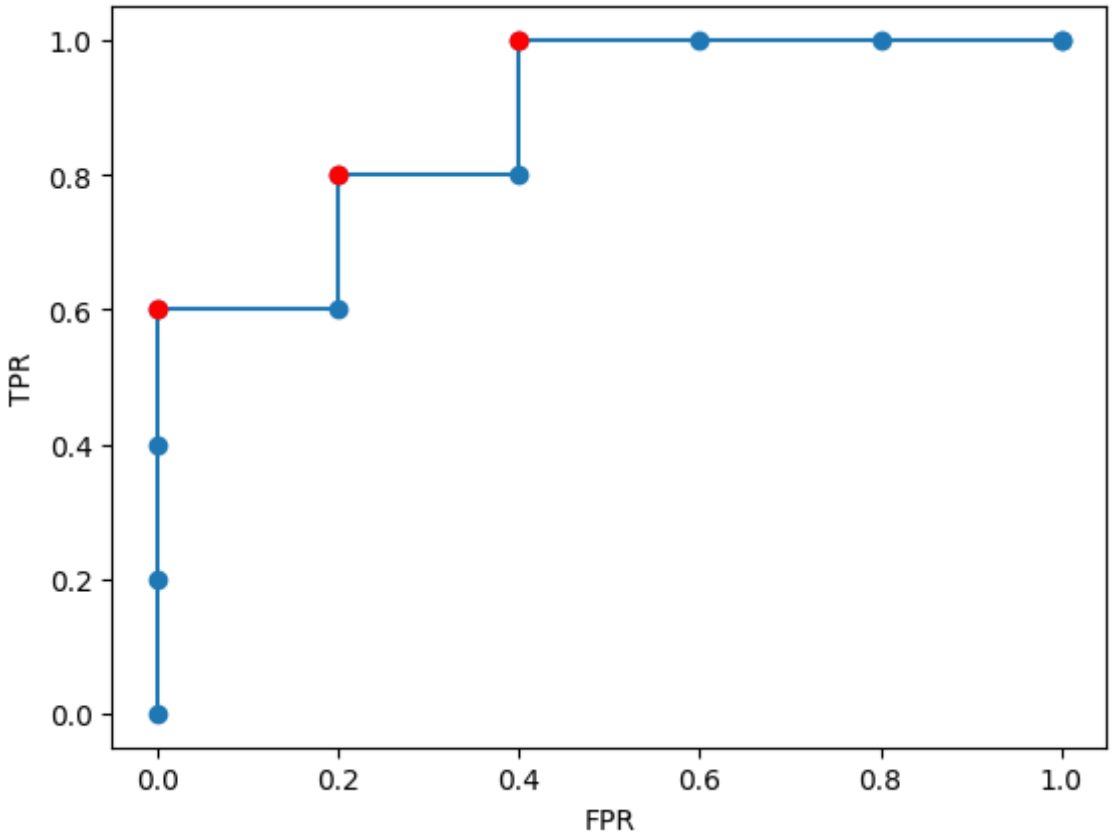
In [ ]:
```
# Main function
def main():
```

```
    gs, fs, ys = getData()
    print("\n G time!")
    subroutine(gs, ys)
    print("\n F time!")
    subroutine(fs, ys)
    return

# Run main
if __name__ == "__main__":
    main()
```

```
 G time!
Min thresh:  [-0.316, 0.406, 1.994] ---> @ error of:  0.2
[TP, FP, FN, TN]:  [5, 2, 0, 3]
Precision:  0.714   |  Recall:  1.0   |  F1:  0.833
```




```
 F time!
Min thresh:  [-0.306, 0.385, 0.964] ---> @ error of:  0.2
[TP, FP, FN, TN]:  [5, 2, 0, 3]
Precision:  0.714   |  Recall:  1.0   |  F1:  0.833
```


1.0
0.8
0.6
0.4
0.2
0.0
TPR
0.0
0.2
0.4
0.6
0.8
1.0
FPR

This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

CS 671: Homework 1

Alex Kumar

Question 5.

In [ ]:
```
# Imports
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from chefboost import Chefboost as chef
```

In [ ]:
```
### General Data Helper Functions

# Returns train and test data
def getData():
    return pd.read_csv("carseats_train.csv"), pd.read_csv("carseats_test.csv")

# Separates the feature vector from the label associated with it
def splitXY(data):
    return data.iloc[:, 1:].to_numpy(), data.iloc[:, 0].to_numpy()

# Change categorical data to numerical
def cleanX(data):
    old, new = ["Bad", "Medium", "Good", "No", "Yes"], [0, 1, 2, 0, 1]
    for i in range(len(old)):
        data[data == old[i]] = new[i]
    return data

# Renames and relocates the label column; changes numerical to categorical
def chefPrep(data):
    data = data.rename(columns={"Sales": "Decision"})
    dec = data.pop("Decision")
    data.insert(len(data.columns), "Decision", dec)

    data.loc[data["Decision"] == 0, "Decision"] = "No"
    data.loc[data["Decision"] == 1, "Decision"] = "Yes"
    return data


### For CV
# Split data into K folds as np array for CV
def kFolds(data_X, data_Y, K=5):
    folds_X, folds_Y = [], []
    bucket = len(data_X) // K
```

```
    start, end = 0, bucket
    for i in range(K):
        if i+1 == K:
            folds_X.append(data_X[start:])
            folds_Y.append(data_Y[start:])
        else:
            folds_X.append(data_X[start: end])
            folds_Y.append(data_Y[start: end])
        start += bucket
        end += bucket
    return folds_X, folds_Y

# Prepare train / valid pairs for CV
def cvSplit(data_X, data_Y, K=5):
    validX, validY, trainX, trainY = [], [], [], []
    folds = list(range(K))
    for i in range(K):
        vX, vY, tX, tY = [], [], [], []
        temp = folds.copy()
        temp.remove(i)
        vX, vY = data_X[i], data_Y[i]
        for j in temp:
            if len(tX) == 0:
                tX, tY = data_X[j], data_Y[j]
            else:
                tX = np.concatenate((data_X[j], tX))
                tY = np.concatenate((data_Y[j], tY))
        validX.append(vX)
        validY.append(vY)
        trainX.append(tX)
        trainY.append(tY)
    return validX, validY, trainX, trainY

# Split data into K folds as pd dataframe for CV
def kFoldsChef(data, K):
    folds = []
    bucket = len(data) // K
    start, end = 0, bucket
    for i in range(K):
        if i+1 == K:
            folds.append(data.iloc[start:])
        else:
            folds.append(data.iloc[start: end])
        start += bucket
        end += bucket
    return folds

# Prepare train / valid pairs for CV
def cvSplitChef(data, K):
    valid, train = [], []
    folds = list(range(K))
    for i in range(K):
        v, t = [], []
        temp = folds.copy()
        temp.remove(i)
        v = data[i]
        for j in temp:
```

```
            if len(t) == 0:
                t = data[j]
            else:
                t = pd.concat([data[j], t])
        valid.append(v)
        train.append(t)
    return valid, train
```

In [ ]:
```
### Evalutation Helper Functions

# Calculate confusion matrix for 1/0
def confusionMatrix(preds, labels):
    confusion = [0, 0, 0, 0]    # [TP, FP, FN, TN]
    for i in range(len(preds)):
        p, l = preds[i], labels[i]
        if p == 1:                                  # pred pos
            if l == 1: confusion[0] += 1            # TP
            else: confusion[1] += 1                 # FP
        else:                                       # pred neg
            if l == 0: confusion[3] += 1            # TN
            else: confusion[2] += 1                 # FN
    return confusion

# Calculate confusion matrix for Yes/No
def chefConfusion(preds, labels):
    confusion = [0, 0, 0, 0]    # [TP, FP, FN, TN]
    for i in range(len(preds)):
        p, l = preds[i], labels[i]
        if p == "Yes":                              # pred pos
            if l == "Yes": confusion[0] += 1        # TP
            else: confusion[1] += 1                 # FP
        else:                                       # pred neg
            if l == "No": confusion[3] += 1         # TN
            else: confusion[2] += 1                 # FN
    return confusion

# Precision
def calcPrecision(c):
    return c[0] / (c[0] + c[1])     # [TP, FP, FN, TN]

# Recall
def calcRecall(c):
    return c[0] / (c[0] + c[2])

# F1
def calcF1(c):
    p = calcPrecision(c)
    r = calcRecall(c)
    return 2 * ((p * r) / (p + r))
```

In [ ]:
```
# KNN Helper Functions

# Returns euclidean distance
def euclidDist(x, y):
    dist = []
    for i in range(len(x)):
        dist.append((x[i] - y[i]) ** 2)
```

```
    return np.sqrt(sum(dist))

# Returns manhattan distance
def manhatDist(x, y):
    dist = []
    for i in range(len(x)):
        dist.append(np.abs(x[i] - y[i]))
    return sum(dist)

# Finds distance from p to all points in train, returns closest k
def distToTrain(train_X, train_Y, p, dist_measure, k):
    distances = []
    for i in range(len(train_X)):
        if dist_measure == "euclidean":
            dist = euclidDist(train_X[i], p)
        elif dist_measure == "manhattan":
            dist = manhatDist(train_X[i], p)
        distances.append((dist, train_Y[i]))
    sorted_dist = sorted(distances)
    return sorted_dist[:k]

# Returns majority label from closest k
def getMajority(distances):
    g1, g2 = 0, 0
    for d in distances:
        if d[1] == 0: g1 += 1
        elif d[1] == 1: g2 += 1
    return 0 if g1 > g2 else 1
```

In [ ]:

```
# Cross Validation Helper Functions

######### COMBINE INTO 1 FUNCT WITH CALL TO FUNCT TO RUN MODEL
# Get summed f1 scores across K folds for each hyper for Decision Tree
def rotateCV(trainX, trainY, validX, validY, hyper, K):
    fold_vals = {}
    for i in range(K):
        for h in hyper:
            cv_tree = DecisionTreeClassifier(max_depth=h, random_state=None)
            cv_tree = cv_tree.fit(trainX[i], trainY[i])

            cv_predict = cv_tree.predict(validX[i])
            confusion_cv = confusionMatrix(cv_predict, validY[i])
            f1_cv = calcF1(confusion_cv)
            # print("{a}th fold f1 score for h value {b}: ".format(a=i, b=h), f
            if h not in fold_vals.keys():
                fold_vals[h] = f1_cv
            else:
                fold_vals[h] += f1_cv
    return fold_vals

# Get summed f1 scores across K folds for each hyper for chefboost
def rotateCVChef(train, valid, hyper, K):
    fold_vals = {}
    for i in range(K):
        for h in hyper:
            config = {"algorithm": "CART", "max_depth": h}
            chef_cv = chef.fit(train[i], config=config)
```

```
            chef_predict = []
            for j in range(len(valid[i])):
                chef_predict.append(chef.predict(chef_cv, valid[i].iloc[j]))
            confusion = chefConfusion(chef_predict, valid[i]["Decision"].to_lis
            f1_cv = calcF1(confusion)

            if h not in fold_vals.keys():
                fold_vals[h] = f1_cv
            else:
                fold_vals[h] += f1_cv
    return fold_vals

# Find hyperparam with best average score across K folds
def findBestK(fold_vals, K):
    for k in fold_vals.keys():
        fold_vals[k] = fold_vals[k] / K

    return max(fold_vals, key=fold_vals.get)

# Gets F1 score for current setting of hyperparams on valid set
def KNNCV(train_X, train_Y, valid_X, valid_Y, k, dist_meas):
    preds = []
    for i in range(len(valid_X)):
        dist = distToTrain(train_X, train_Y, valid_X[i], dist_meas, k)
        vote = getMajority(dist)
        preds.append(vote)

    knn_confusion = confusionMatrix(preds, valid_Y)
    return calcF1(knn_confusion)
```

In [ ]:

```
# Decision Tree Main Function
def decTree():
    # Get and clean the data
    train, test = getData()
    train_X, train_Y = splitXY(train)
    test_X, test_Y = splitXY(test)
    train_X = cleanX(train_X)
    test_X = cleanX(test_X)

    # Train the model
    decision_tree = DecisionTreeClassifier(max_depth=3, random_state=None)
    decision_tree = decision_tree.fit(train_X, train_Y)

    # Get test F1 score
    dt_predict = decision_tree.predict(test_X)
    confusion = confusionMatrix(dt_predict, test_Y)
    f1 = calcF1(confusion)
    # print("Decision Tree F1 Score: ", f1)

    # CV
    K, hyper = 5, [1, 2, 3, 4]
    folds_X, folds_Y = kFolds(train_X, train_Y, K)
    # Split into sets that are rotated
    validX, validY, trainX, trainY = cvSplit(folds_X, folds_Y, K)

    # Train over all h
    fold_vals = rotateCV(trainX, trainY, validX, validY, hyper, K)
```

```
    # Find best h
    max_key = findBestK(fold_vals, K)
    print("Best value of hyperparmeter being tuned: ", max_key)

    # Make model with optimal hyperparam
    optimal_tree = DecisionTreeClassifier(max_depth=max_key, random_state=None)
    optimal_tree = optimal_tree.fit(train_X, train_Y)

    # Get F1
    optimal_predict = optimal_tree.predict(test_X)
    confusion_optimal = confusionMatrix(optimal_predict, test_Y)
    f1_optimal = calcF1(confusion_optimal)
    print("Normal Decision Tree F1 Score: ", f1)
    print("F1 score after CV tuning: ", f1_optimal)
    return
```

In [ ]:

```
# Chefboost Main Function
def chefTime():
    # Get and clean the data
    train, test = getData()
    train = chefPrep(train)
    test = chefPrep(test)

    # Train the model
    config = {"algorithm": "CART", "max_depth": 5}
    chef_model = chef.fit(train, config=config)

    # Get test F1 Score
    chef_predict = []
    for i in range(len(test)):
        chef_predict.append(chef.predict(chef_model, test.iloc[i]))
    confusion = chefConfusion(chef_predict, test["Decision"].to_list())
    f1 = calcF1(confusion)
    # print("Chef F1 Score: ", f1)

    # CV
    K, hyper = 5, [1, 2, 3, 4]
    folds = kFoldsChef(train, K)
    # Split into sets that are rotated
    valid_cv, train_cv = cvSplitChef(folds, K)
    # Train over all h
    fold_vals = rotateCVChef(train_cv, valid_cv, hyper, K)

    # Find best h
    max_key = findBestK(fold_vals, K)
    print("Best value of hyperparmeter being tuned: ", max_key)

    # Make model with optimal hyperparam
    config = {"algorithm": "CART", "max_depth": max_key}
    chef_optimal = chef.fit(train, config=config)

    # Get F1
    optimal_predict = []
    for i in range(len(test)):
        optimal_predict.append(chef.predict(chef_optimal, test.iloc[i]))
    confusion_opt = chefConfusion(optimal_predict, test["Decision"].to_list())
```

```
    f1_optimal = calcF1(confusion_opt)
    print("Normal Chef F1 Score: ", f1)
    print("Chef F1 Score after CV Tuning: ", f1_optimal)
    return
```

In [ ]:
```
# KNN Main Function
def KNN():
    # Get and clean data
    train, test = getData()
    train_X, train_Y = splitXY(train)
    test_X, test_Y = splitXY(test)
    train_X = cleanX(train_X)
    test_X = cleanX(test_X)

    # Run KNN CV
    dist_measures = ["euclidean", "manhattan"]
    num_neighbors = [1, 3, 5, 7, 9]
    K = 5
    folds_X, folds_Y = kFolds(train_X, train_Y, K)
    # Split into sets that are rotated
    validX, validY, trainX, trainY = cvSplit(folds_X, folds_Y, K)

    fold_vals = {}
    for d in dist_measures:
        for n in num_neighbors:
            for i in range(len(trainX)):
                cv_f1 = KNNCV(trainX[i], trainY[i], validX[i], validY[i], n, d)
                if (d,n) not in fold_vals.keys():
                    fold_vals[(d,n)] = cv_f1
                else:
                    fold_vals[(d,n)] += cv_f1
                # print("Dist {d}; NumNeih {n}; Fold {i} F1 score: ".format(d=d

    max_key = findBestK(fold_vals, K)
    print("Best distance and NN parameters: ", max_key)

    # Performance of Optimal KNN
    preds = []
    for i in range(len(test_X)):
        dist = distToTrain(train_X, train_Y, test_X[i], max_key[0], max_key[1])
        vote = getMajority(dist)
        preds.append(vote)

    knn_confusion = confusionMatrix(preds, test_Y)
    knn_f1 = calcF1(knn_confusion)
    print("KNN F1 score after CV tuning: ", knn_f1)
    return
```

In [ ]:
```
# Run Decision Tree Function
decTree()
```

```
Best value of hyperparmeter being tuned:  4
Normal Decision Tree F1 Score:  0.5777777777777777
F1 score after CV tuning:  0.6923076923076924
```

In [ ]:
```
# Run Chefboost Function
chefTime()
```

```
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.641408920288086  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  97.16312056737588 % on  282  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[100, 3], [5, 174]]
Precision:  97.0874 %, Recall:  95.2381 %, F1:  96.1539 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.431269884109497  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[82, 0], [3, 141]]
Precision:  100.0 %, Recall:  96.4706 %, F1:  98.2036 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.3990559577941895  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[82, 0], [3, 141]]
Precision:  100.0 %, Recall:  96.4706 %, F1:  98.2036 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.5331168174743652  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[82, 0], [3, 141]]
Precision:  100.0 %, Recall:  96.4706 %, F1:  98.2036 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.4706659317016602  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[82, 0], [3, 141]]
Precision:  100.0 %, Recall:  96.4706 %, F1:  98.2036 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
```

```
--------------------------
finished in  1.582603931427002  seconds
--------------------------
Evaluate  train set
--------------------------
Accuracy:  96.90265486725664 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[79, 2], [5, 140]]
Precision:  97.5309 %, Recall:  94.0476 %, F1:  95.7576 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
--------------------------
finished in  1.5586268901824951  seconds
--------------------------
Evaluate  train set
--------------------------
Accuracy:  96.90265486725664 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[79, 2], [5, 140]]
Precision:  97.5309 %, Recall:  94.0476 %, F1:  95.7576 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
--------------------------
finished in  1.6172471046447754  seconds
--------------------------
Evaluate  train set
--------------------------
Accuracy:  96.90265486725664 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[79, 2], [5, 140]]
Precision:  97.5309 %, Recall:  94.0476 %, F1:  95.7576 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
--------------------------
finished in  1.5485520362854004  seconds
--------------------------
Evaluate  train set
--------------------------
Accuracy:  96.90265486725664 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[79, 2], [5, 140]]
Precision:  97.5309 %, Recall:  94.0476 %, F1:  95.7576 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
--------------------------
finished in  1.577951192855835  seconds
--------------------------
Evaluate  train set
--------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 1], [2, 140]]
Precision:  98.8095 %, Recall:  97.6471 %, F1:  98.2249 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
--------------------------
finished in  1.581491231918335  seconds
```

```
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 1], [2, 140]]
Precision:  98.8095 %, Recall:  97.6471 %, F1:  98.2249 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.581228256225586  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 1], [2, 140]]
Precision:  98.8095 %, Recall:  97.6471 %, F1:  98.2249 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.6014139652252197  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  98.67256637168141 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 1], [2, 140]]
Precision:  98.8095 %, Recall:  97.6471 %, F1:  98.2249 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.2408387660980225  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  96.01769911504425 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 5], [4, 134]]
Precision:  94.3182 %, Recall:  95.4023 %, F1:  94.8572 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.2702221870422363  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  96.01769911504425 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 5], [4, 134]]
Precision:  94.3182 %, Recall:  95.4023 %, F1:  94.8572 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.2755241394042969  seconds
-------------------------
Evaluate  train set
```

```
-------------------------
Accuracy:  96.01769911504425 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 5], [4, 134]]
Precision:  94.3182 %, Recall:  95.4023 %, F1:  94.8572 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.27650785446167  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  96.01769911504425 % on  226  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[83, 5], [4, 134]]
Precision:  94.3182 %, Recall:  95.4023 %, F1:  94.8572 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.4284229278564453  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  99.55357142857143 % on  224  instances
Labels:  ['No' 'Yes']
Confusion matrix:  [[145, 1], [0, 78]]
Precision:  99.3151 %, Recall:  100.0 %, F1:  99.6564 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.4346230030059814  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  99.55357142857143 % on  224  instances
Labels:  ['No' 'Yes']
Confusion matrix:  [[145, 1], [0, 78]]
Precision:  99.3151 %, Recall:  100.0 %, F1:  99.6564 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.4274659156799316  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  99.55357142857143 % on  224  instances
Labels:  ['No' 'Yes']
Confusion matrix:  [[145, 1], [0, 78]]
Precision:  99.3151 %, Recall:  100.0 %, F1:  99.6564 %
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.427408218383789  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  99.55357142857143 % on  224  instances
```

```
Labels:  ['No' 'Yes']
Confusion matrix:  [[145, 1], [0, 78]]
Precision:  99.3151 %, Recall:  100.0 %, F1:  99.6564 %
Best value of hyperparmeter being tuned:  1
[INFO]:  5 CPU cores will be allocated in parallel running
CART  tree is going to be built...
-------------------------
finished in  1.619767665863037  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  97.16312056737588 % on  282  instances
Labels:  ['Yes' 'No']
Confusion matrix:  [[100, 3], [5, 174]]
Precision:  97.0874 %, Recall:  95.2381 %, F1:  96.1539 %
Normal Chef F1 Score:  0.6434782608695652
Chef F1 Score after CV Tuning:  0.6434782608695652
```

In [ ]:
```
# Run KNN Function
KNN()
```

```
Best distance and NN parameters:  ('manhattan', 1)
KNN F1 score after CV tuning:  0.5535714285714286
```