(I was enrolled to the course only after the first QA was due. I have emailed Professor Krishnan about this as well. Thus I will include the make-up of 4 QAs from the first lectures.)

W1. KNN

- Q1. KNN can be sensitive to outliers and noisy data because it uses the closest neighbors for prediction. What should we do if we have noisy data?
    - A. We may use Z-score or interquartile range to identify and remove outliers that may distort the predictions. We can also implement feature weighting that assigns lower weights to those features that are prone to noise and higher weights for those that are more reliable.
- Q2. One weakness of KNN is that it suffers from 'curse of dimensionality', which is shown in what we attempt in Assignment 1 question 1. In assignment 1 question 5, it also asks us to predict text classification using KNN. The problem here is that we vectorize all the texts and have >10000 features after preprocessing. What would be a great way to deal with this dimension problem if we need to train a KNN classifier on a high-dimension dataset?
    - A. We can implement some dimensionality reduction techniques e.g. principal component analysis (PCA). It transforms a dataset with many features (high dimension) into a smaller set of uncorrelated features, which are called the principal components.

W2. Decision Tree

- Q1. In the lecture slide we only talk about information gain (entropy) as the criterion. In fact, there is another criterion called gini index. How does it work and what is the difference between them?
    - A. Gini index measures the probability of misclassifying a random element, where lower gini values indicating purer nodes. One difference is that in terms of computation speed, gini is faster to compute.
- Q2. Beyond decision trees, there is an ensemble model called random forest. How does the random forest method combine multiple decision trees to improve model performance?
-
    - A. We create an ensemble of decision trees by growing each tree on an independent bootstrap sample from the data. At each node, we randomly select a subset of features from the full set of features, and find the best split based on these features. Once all the trees are trained, we aggregate the predictions by either voting (for classification) or averaging (for regression) the outputs of all the trees to make a final prediction for a new example.

W3 & 4. Linear Regression

- Q1. In linear regression models, all features are considered independent. What if there is multicollinearity in some features? E.g., to predict housing prices, we commonly use 1)

the size of the house, and 2) the number of bedrooms as predictors. These two, however, are obviously highly related because larger houses tend to have more rooms as well.
    - A. Feature engineering can be used to deal with multicollinearity. One simple approach could be to combine the two correlated features into one. For example, we could analyze the relationship between the size of the house and the number of bedrooms, then create a new feature that captures both (e.g., size per bedroom or a combined metric). This reduces multicollinearity while retaining the information from both features.
- Q2. In assignment 1, we worked on the Twitter text dataset and used KNN/decision trees to classify whether a tweet asserts global warming exists/does not exist. Would linear regression work well on this too?
    - A. To preprocess the data, we used count vectorization (Bag of Words) to convert tweets into numerical form. The dataset has the following characteristics: 1) The values are discrete (0 or integers, representing word counts in each tweet), and 2) The dataset is high-dimensional, with more than 10,000 sparse columns (each representing a unique word). Given these characteristics, linear regression may not work well for several reasons: 1) It is designed for continuous target variables, not binary classification, and 2) The high-dimensionality and sparsity of the data can lead to overfitting and unstable coefficient estimates. Models specifically designed for classification, such as logistic regression or decision trees, are more suitable.
- Q3. In the probabilistic interpretation of the squared error section, we mentioned MLE. This reminded me of the Bayesian modeling class from my undergraduate course earlier this year (where we covered concepts like prior, likelihood, and posterior). What are some major differences between linear models and Bayesian models?
    - A. A key advantage of Bayesian models is that they can perform well with limited data. In Bayesian modeling, we continuously update our parameter estimates (posterior distribution) by combining prior beliefs with new data (likelihood). This iterative updating allows us to make robust predictions even with small sample sizes. In contrast, linear models typically require a larger dataset to provide reliable point estimates. Linear models focus on producing a single set of optimal parameters (point estimates), whereas Bayesian models provide a distribution of possible parameter values, reflecting uncertainty.
- Q4. When working in a real-world setting (e.g. using sklearn), how would we implement gradient descent?
    - A. We won’t need to implement gradient descent manually in sklearn. However, there is a linear model called SGDRegressor, which is optimized using stochastic gradient descent (SDG). SGDRegressor is useful for large datasets, as it updates the model's parameters using one or a few data points at a time rather than the entire dataset. You can customize the learning rate (alpha), apply regularization (L1 or L2), and adjust the number of iterations to control the convergence of the model.