final

December 5, 2024

1 Final Questions

1.1 Instructions

For each of the chosen questions:

- A brief (1-2 sentence) plain English answer to the question
- A simple data example or sketch that illustrates your answer
- A mathematical derivation that supports your answer

1.2 Example

Question: In KNN, sometimes a large K is a better choice and sometimes a small K is. Why?

Short answer:

- With a small K, we are using only the closest neighbors to predict. In some data this might work well, while in other data this might be overfitting, and we need to average more neighbors for less variable predictions.

Data-y answer:

- Imagine we wanted to predict performance of marathon runners based on their times in previous races. Because there are very few elite top runners, we might achieve better performance with a low K, so that we predict fast times for this small group of runners.
- Now imagine we want to predict ratings of a movie coming out soon, based on the genre and actors involved. Since there is a lot of variability in movie quality even for a particular genre and cast, we might want to use a larger K to average performance over a larger set of similar movies, to avoid extreme predictions

Math-y answer:

- Consider precision and recall in a classification problem with KNN. When K gets very large, most of the neighbors will be from whatever the most common class is. Thus, we will predict the largest class for every observation.
- Suppose we consider the largest class, Class A, to be the target class. This means the False Negative rate will be low, because we predict Class A every time. But it also means the False Positive rate will be high, because we predict Class A for the non-A classes.
- The precision is equal to TP/(FP + TP) - and since FP is high, this number will be relatively small.

1.3 Questions

1.3.1 First Half Questions

1. Sometimes, standardizing some or all of the predictors helps. Sometimes it does not. Why?
   - Standardizing predictors can enhance model performance by ensuring all features contribute equally, especially for algorithms sensitive to feature scales. However, it may not be beneficial for models that are inherently scale-invariant or when the original scales carry meaningful information.
   - Consider a dataset with two features: height in centimeters and weight in kilograms. Standardizing both ensures neither feature dominates due to its larger numerical range, benefiting algorithms like Support Vector Machines (SVM) or gradient descent-based methods. However, for decision trees, which are not affected by feature scaling, standardization offers no advantage.
   - Consider two features x_1 and x_2 with different variances. In linear regression, the coefficients are estimated as $\beta = (X^T X)^{-1} X^T y$. If x_1 has a larger variance than x_2, the condition number of $X^T X$ increases, making the estimation of β unstable. By standardizing: $z_i = \frac{x_i - \mu_i}{\sigma_i}$, the variance of each feature becomes 1, improving numerical stability and ensuring that regularization penalties treat all feature equally. For algorithms such as decision trees, split based on order and not magnitude, standardization is unnecessary.
2. Why might we choose a metric that is not the same as our loss function (or vice versa)?
   - Choosing a metric different from the loss function allows for optimizing one aspect of performance during training while evaluating another aspect that better aligns with real-world objectives or business needs.
   - Consider a binary classification problem where the data is imbalanced. Cross-entropy as the loss function can be used to train the model but the metric to evaluate the performance should be F1 score, since it better captures the balance between precision and recall. Another example could be using Mean Squared Error (MSE) as the loss function for regression tasks but using Mean Absolute Error (MAE) as the evaluation metric since it is more interpretable (model to predict house prices).
   - Cross-entropy: $L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$. F1 score: $F1 = 2 \times \frac{precision \times recall}{precision + recall}$. Optimizing L minimizes the loss, improving estimates, where the metric measures the model's performance. The separation allows training to focus on probability estimates while evaluation focuses on the model's ability to classify correctly.
3. Many of the models we studied involve a linear combination of features. How do we go about choosing a loss function to fit coefficients?
   - Choosing a loss function for linear models depends on the problem type such as regression or classification and the desired properties of the solution. It needs to ensure that the coefficients are optimized to minimize the chosen error metric.
   - Take the model for predicting house prices, where MSE would be the loss function to penalize large errors more severely, which leads to optimal coefficients for features such as size and bedrooms.
   - For a linear model such as $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$, the MSE loss function is $L(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$. Taking the derivative with respect to β_j and setting it to zero for minimization gives $\frac{\partial L}{\partial \beta_j} = -\frac{2}{n} \sum_{i=1}^{n} x_{ij}(y_i - \hat{y}_i) = 0$. Solving these equations gives the coefficients that minimize the MSE, ensuring the best linear fit according to the chosen loss function.
4. Why do some models work well in higher dimensions, and some do not?

- Some models work well in higher dimensions because they can focus on important features or relationships, while others struggle due to the curse of dimensionality, where the data becomes sparse and noisy. Models that can handle high-dimensional data effectively often have built-in mechanisms to reduce the impact of irrelevant features.
- Consider an example where we need to classify emails as spam or not spam. Models like KNN would struggle in this high dimensional data because all the emails would be far apart since every word is a dimension. However, models like SVM can find a hyperplane that separates the emails effectively, as long as the data is linearly separable.
- In high dimensions the distance between points grows and becomes less meaningful. Take the euclidean distance between two points in d dimensions: $d(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$. As d gets large all the distances start to look similar, making it harder for some models to distinguish between nearby and far points. SVM avoids this by finding a hyperplane like $w \cdot x + b = 0$ that separates the classes.

5. What are three possible stopping conditions for gradient descent, and why might we prefer or not prefer each?
   - Three possible stopping conditions for gradient descent are when the gradient is small, when the loss function stops improving significantly, and when a fixed number of iterations is reached. The choice of stopping condition depends on the computational resources available, the desired accuracy, and the convergence properties of the optimization algorithm.
   - Consider a scenario where you are filling a bucket with water. You can stop when the bucket is almost full (small gradient), when adding more water does not change the level significantly (loss function improvement), or after a fixed number of bucket refills (iterations). If you stop too soon then the bucket will not be full, but if you wait too long you might overflow.
   - For fixed iterations, the algorithm would stop after a predefined number of iterations $(N) : t \geq N$. This ensures computational efficiency but might stop before convergence. For change in loss function, the algorithm stops when the absolute diff between loss at consecutive steps becomes negligible: $|J(\theta^{(t)}) - J(\theta^{(t-1)})| < \epsilon$. For the small gradient, the algorithm stops when the gradient's magnitude is below a small threshold: $||\nabla J(\theta)|| < \epsilon$.

1.3.2 Second Half Questions

6. One approach to classification problems is to establish a decision boundary. Why might some boundaries be "better" choices than others, and how do we measure that?
   - A good decision boundary cleanly separates classes and performs well on new data. Some boundaries are better because they strike a balance between capturing patterns in the training data and staying flexible enough to handle unseen examples
   - Consider an example of classifying images of cats and dogs. A boundary drawn tightly around cat images might work well for the training set but fails on test data if a dog image looks similar. A better boundary would leave more "margin" around each class, ensuring correct classification even for unambiguous examples.
   - Margin is the gap between the boundary and the closest points (support vectors from class). A larger margin makes the boundary more robust. The margin is calculated as $\frac{2}{||w||}$ where w is the weight vector of the decision boundary. Maximizing the margin is equivalent to minimizing $||w||$ which is the objective of the SVM algorithm.
7. There are several ways to add more complexity to a Neural Net. Discuss each of these, and explain why you might prefer or not prefer to add more complexity in this way.

- Increasing the number of layers, adding more nodes per layer, and different activation functions are ways to add complexity to a Neural Net. More complexity can improve the model's ability to capture complex patterns but at the risk of overfitting, computational cost, and interpretability.
- Take a neural network to recognize handwritten digits. Adding more layers can help the network capture more than just basic shapes such as combining edges into digits and the recognizing complete digit shapes. Adding more nodes per layer can help the network start to learn increasingly subtle details such as the curvature of a line. Different activation functions can help the network learn different types of patterns such as edges, shapes, and textures.
- $TotalParameters = (n_{inputs} \times n_{nodes}) + n_{nodes} + (n_{nodes} \times n_{nodes}) + n_{nodes}$. Adding more layers or nodes increases the parameter count, making the model capable of fitting more complex functions. However this can lead to overfitting as the effective capacity of the model outgrows the data.

8. Suppose you wanted to fit a Neural Net, but you wanted the input function to be a non-linear combination of the predictors. Why might you want to do this, and what would you need to change in the Neural Net implementation process to make it happen?
   - You would want to do this to capture the non-linear patterns in the data that linear models cannot. To implement this you would need either a initial custom layer for the non-linear transformations or transform the input data before feeding it into the network.
   - Take an example such as predicting the success of a startup with features like the age of the founder and initial funding. These features might seem independent but non-linear transformation could reveal other details. For example, the squared funding might capture better exponential growth potential or the interaction between age and funding might reveal success patterns.
   - Let x be the input data and $f(x)$ be the non-linear transformation. This means that the traditional linear model would be $y = w \cdot x + b$ and the non-linear model would be $y = w \cdot f(x) + b$. $f(x)$ could be a polynomial, kernel, or log transformation. The goal in the end would be expand the feature space to make linearly inseparable data separable.
9. Why do we apply an activation function between layers of our Neural Network?
   - Activation functions introduce non-linearity into the network, which enables it to model complex, non-linear patterns. Without activation functions neural networks would be no more powerful than a linear regression model.
   - For a conceptual sketch, activation functions are like a creative translator in a conversation. Linear communication would be the raw inputs and the activation function would be the translation that adds context and meaning to the conversation. This means that different activation functions mean different translations. Another example would be recognizing dog images where a linear model would only see basic pixel values, while non-linear activation functions would learn to detect edges, shapes, and complex features like fur patterns.
   - Consider the output of a single layer network: $z = w_1 x + b_1$. Adding an activation function such as sigmoid: $a = \sigma(z) = \frac{1}{1+e^{-z}}$ prevents the network from becoming a purely linear combination of the inputs. This means that $f(x) = g(w_2 g(w_1 x + b_1) + b_2)$ can capture non-linear patterns in the data.
10. Suppose we fit a Neural Net with multiple outputs, which we want to use to predict a multiclass problem. How do we make sure those outputs are interpretable and can be used for classification?

- We can make sure those outputs are interpretable and can be used for classification by applying a softmax activation to the outputs. This converts them into probabilities and the predicted class corresponds to the highest probability.
- Take for example a movie genre classifier where the input is the movie features such as actors, genre, and director. The output would be a vector of probabilities for each genre like comedy, drama, and action. Softmax activation transforms raw scores to comedy: 0.6, drama: 0.3, action: 0.1. In this case the predicted class would be comedy.
- The softmax function is defined as: $p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ where x is the raw output of the network and n is the number of classes. The sum of the probabilities is 1 and the highest probability is $\hat{y} = \text{argmax}(p_i)$. Loss function for a multi-class problem would be the cross-entropy: $L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$.