Machine Learning Practice Problems
(MFML)

SEC9

September 6, 2025

1 Decision Tree Learning

Question 1 (Numerical)

Problem: Given the following dataset for predicting whether a startup will be profitable, calculate the Information Gain for splitting on the 'Funding' attribute and the 'Team Size' attribute. Which attribute would the ID3 algorithm choose as the root of the decision tree?

ID	Funding	Team Size	Profitable
1	High	Small	Yes
2	High	Large	Yes
3	Low	Small	No
4	Medium	Small	Yes
5	Medium	Large	No
6	Low	Large	No
7.	High	Large	Yes
8	Medium	Small	No

Solution

Step 1: Calculate the entropy of the entire dataset (Entropy(S)).

- Total instances: 8
- Profitable (Yes): 4
- Not Profitable (No): 4
- $P(\text{Yes}) = 4/8 = 0.5$
- $P(\text{No}) = 4/8 = 0.5$
- $Entropy(S) = -P(\text{Yes})\log_2(P(\text{Yes})) - P(\text{No})\log_2(P(\text{No}))$
- $Entropy(S) = -0.5\log_2(0.5) - 0.5\log_2(0.5) = -0.5(-1) - 0.5(-1) = 1$ bit.

Step 2: Calculate the Information Gain for the 'Funding' attribute.

- **Funding = High**: 3 instances (3 Yes, 0 No). Entropy = 0.
- **Funding = Medium**: 3 instances (1 Yes, 2 No). $E = -(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}) \approx 0.918$.

- **Funding = Low**: 2 instances (0 Yes, 2 No). Entropy = 0.
- Weighted average entropy: $Gain(S, \text{Funding}) = E(S) - \left[\frac{3}{8}E(\text{High}) + \frac{3}{8}E(\text{Medium}) + \frac{2}{8}E(\text{Low})\right]$
- $Gain(S, \text{Funding}) = 1 - \left[\frac{3}{8}(0) + \frac{3}{8}(0.918) + \frac{2}{8}(0)\right] = 1 - 0.344 = 0.656$.

Step 3: Calculate the Information Gain for the 'Team Size' attribute.

- **Team Size = Small**: 4 instances (2 Yes, 2 No). $E = -(\frac{2}{4}\log_2\frac{2}{4} + \frac{2}{4}\log_2\frac{2}{4}) = 1$.
- **Team Size = Large**: 4 instances (2 Yes, 2 No). $E = -(\frac{2}{4}\log_2\frac{2}{4} + \frac{2}{4}\log_2\frac{2}{4}) = 1$.
- Weighted average entropy: $Gain(S, \text{Team Size}) = E(S) - \left[\frac{4}{8}E(\text{Small}) + \frac{4}{8}E(\text{Large})\right]$
- $Gain(S, \text{Team Size}) = 1 - \left[\frac{4}{8}(1) + \frac{4}{8}(1)\right] = 1 - 1 = 0$.

Step 4: Compare Information Gains.

- $Gain(S, \text{Funding}) = 0.656$
- $Gain(S, \text{Team Size}) = 0$
- Since $Gain(S, \text{Funding}) > Gain(S, \text{Team Size})$, the ID3 algorithm would choose **Funding** as the root node for the decision tree.

Question 2 (Scenario)

Problem: Your team is building a decision tree to predict customer churn. One of the features is 'AverageMonthlySpend', which is a continuous-valued attribute. How would an algorithm like C4.5 handle this attribute? Describe the process of finding the best split threshold.

Solution

C4.5 handles continuous attributes by creating a binary split based on a threshold value. The process is as follows:

Step 1: Sort the Data

- First, all unique values of the 'AverageMonthlySpend' attribute are collected from the training data.
- The data points are then sorted in ascending order based on this attribute's value.

Step 2: Identify Candidate Split Points

- The algorithm identifies candidate split points. A common method is to take the midpoint between every pair of adjacent sorted values where the target class label changes.
- For example, if we have sorted spending values \$50 (No Churn), \$65 (Churn), \$70 (Churn), the candidate split point between the first two would be $(\$50 + \$65)/2 = \$57.5$. A split point is only necessary when the class label changes between adjacent values.

Step 3: Calculate Information Gain for Each Candidate Split

- For each candidate split point (threshold T), the data is partitioned into two subsets:
  - Subset 1: 'AverageMonthlySpend' $\leq T$
  - Subset 2: 'AverageMonthlySpend' $> T$

- The Information Gain (or Gain Ratio, in the case of C4.5) is calculated for each of these potential binary splits.

Step 4: Select the Best Split

- The threshold T that results in the highest Information Gain (or Gain Ratio) is selected as the best split point for the 'AverageMonthlySpend' attribute at that node.
- This continuous attribute can then be used again for further splits deeper in the tree, with different thresholds.

Question 3 (Scenario)

Problem: You are given a dataset where 15% of the values for the attribute 'CustomerAge' are missing. When building a decision tree, how would you handle these missing values during the training phase using a probabilistic splitting approach (like the one used in C4.5)?

Solution

The C4.5 algorithm uses a sophisticated probabilistic approach to handle missing values during both training (calculating gain) and prediction (classifying a new instance).

Handling Missing Values During Training (Gain Calculation):

1. **Fractional Gain Calculation:** When calculating the Information Gain for an attribute like 'CustomerAge', the instances with missing values are not ignored. Instead, they are fractionally distributed down each branch of the potential split.
2. **Example:** Suppose we are considering a split on 'CustomerAge' > 30. Let's say we have 100 instances at the current node. 85 have a value for 'CustomerAge', and 15 are missing.
   - Of the 85 instances with age, suppose 60 are > 30 and 25 are ≤ 30.
   - The 15 instances with missing 'CustomerAge' are split proportionally. A fraction of 60/85 of each of these 15 instances is sent down the '¿ 30' branch, and a fraction of 25/85 is sent down the '≤ 30' branch.
3. **Entropy Calculation:** The Information Gain is then calculated using these fractional counts in the entropy formulas. This allows the algorithm to use the information from the non-missing attributes of those instances without unfairly penalizing the attribute for having missing values.

Handling Missing Values During Classification:

1. When an instance with a missing 'CustomerAge' needs to be classified, it also travels down all possible branches of the split on 'CustomerAge'.
2. Each path leads to a leaf node with a class probability distribution. The final prediction is a weighted average of the predictions from all the leaves reached, weighted by the proportion of training instances that went down each path.

Question 4 (Conceptual/Applied)

Problem: Compare pre-pruning and post-pruning in decision trees. Describe a scenario where post-pruning would be significantly more effective than pre-pruning.

Solution

Comparison:

- **Pre-pruning (Early Stopping):** This strategy involves stopping the tree's growth early during the training process. A node is not split if it fails to meet a certain criterion. Common criteria include:
  - Maximum tree depth is reached.
  - The number of samples in a node is below a minimum threshold.
  - The information gain from a split is below a certain threshold.
  - A chi-squared test on the split's significance is not met.
- **Post-pruning (Pruning a Fully Grown Tree):** This strategy allows the tree to grow to its full complexity (potentially overfitting the training data) and then prunes it back. It replaces subtrees with leaf nodes if the replacement results in a better generalization performance (usually measured on a validation set). Common methods are Reduced Error Pruning and Cost Complexity Pruning.

Scenario for Post-pruning's Effectiveness:

- Consider a dataset where the target concept has an "XOR-like" feature interaction. For example, a customer is valuable (*Class=Yes*) if they are either (*Feature A=High* AND *Feature B=Low*) OR (*Feature A=Low* AND *Feature B=High*). Otherwise, they are not valuable.
- **Why pre-pruning fails:** When considering the first split on either *Feature A* or *Feature B* alone, the information gain might be very low or zero. This is because splitting on just one feature doesn't separate the classes well. A pre-pruning algorithm with a minimum information gain threshold would likely stop right at the root, creating a useless tree. It cannot see the benefit of the future split on the second feature.
- **Why post-pruning succeeds:** Post-pruning would first allow the tree to grow fully. It would make the low-gain split on *Feature A* and then, in the subsequent nodes, it would make the highly informative split on *Feature B*. This would correctly capture the XOR relationship. After the tree is fully built, the pruning process would evaluate this structure on a validation set and see that this subtree is highly effective and should not be pruned. It overcomes the "myopic" nature of the greedy splitting process that hampers pre-pruning in such cases.

Question 5 (Numerical/Scenario)

Problem: Given the following data with a continuous feature 'TestScore', demonstrate how the C4.5 algorithm would find the best binary split for this feature.

TestScore	**Pass**
55	No
60	No
75	Yes
80	Yes
85	No
95	Yes

Solution

Step 1: Sort the data by 'TestScore' and identify changes in the 'Pass' class. The data is already sorted. Let's list it with the class labels:

- 55 (No)
- 60 (No) $\rightarrow$ Class changes here $\rightarrow$ 75 (Yes)
- 75 (Yes)
- 80 (Yes) $\rightarrow$ Class changes here $\rightarrow$ 85 (No)
- 85 (No) $\rightarrow$ Class changes here $\rightarrow$ 95 (Yes)

Step 2: Identify candidate split thresholds. The thresholds are the midpoints between values where the class label changes.

- Threshold 1 (T1): $(60 + 75)/2 = 67.5$
- Threshold 2 (T2): $(80 + 85)/2 = 82.5$
- Threshold 3 (T3): $(85 + 95)/2 = 90.0$

Step 3: Calculate Information Gain for each threshold. First, calculate the entropy of the parent node (all 6 instances).

- $P(\text{Yes}) = 3/6 = 0.5$, $P(\text{No}) = 3/6 = 0.5$
- $Entropy(\text{Parent}) = -0.5\log_2(0.5) - 0.5\log_2(0.5) = 1$

Now, evaluate each split:

- **Split at T1 = 67.5:**
  - Left ($<= 67.5$): $\{55, 60\} \rightarrow$ 2 No, 0 Yes. $Entropy_{Left} = 0$.
  - Right (> 67.5): $\{75, 80, 85, 95\} \rightarrow$ 3 Yes, 1 No. $Entropy_{Right} = -(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}) \approx 0.811$.
  - $Gain(T1) = 1 - [\frac{2}{6}(0) + \frac{4}{6}(0.811)] = 1 - 0.541 = 0.459$.
- **Split at T2 = 82.5:**
  - Left ($<= 82.5$): $\{55, 60, 75, 80\} \rightarrow$ 2 No, 2 Yes. $Entropy_{Left} = 1$.
  - Right (> 82.5): $\{85, 95\} \rightarrow$ 1 No, 1 Yes. $Entropy_{Right} = 1$.
  - $Gain(T2) = 1 - [\frac{4}{6}(1) + \frac{2}{6}(1)] = 1 - 1 = 0$.
- **Split at T3 = 90.0:**
  - Left ($<= 90.0$): $\{55, 60, 75, 80, 85\} \rightarrow$ 3 No, 2 Yes. $Entropy_{Left} = -(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}) \approx 0.971$.
  - Right (> 90.0): $\{95\} \rightarrow$ 1 Yes. $Entropy_{Right} = 0$.
  - $Gain(T3) = 1 - [\frac{5}{6}(0.971) + \frac{1}{6}(0)] = 1 - 0.809 = 0.191$.

Step 4: Select the best split. Comparing the gains: $Gain(T1) = 0.459$, $Gain(T2) = 0$, $Gain(T3) = 0.191$. The algorithm would choose the split 'TestScore ¡= 67.5' because it provides the highest information gain.

2 Instance-based Learning

Question 6 (Numerical)

Problem: Given the following 2D dataset, classify the new point $P_{new}(3, 5)$ using the k-Nearest Neighbor algorithm with $k = 3$. Calculate the result using both Euclidean distance and Manhattan distance.

Point	Coordinates (x, y)	Class
P_1	(1, 2)	A
P_2	(2, 4)	A
P_3	(4, 1)	B
P_4	(5, 4)	B
P_5	(4, 6)	A
P_6	(6, 2)	B

Solution

The new point is $P_{new}(3, 5)$.

Part 1: Using Euclidean Distance The Euclidean distance is $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

- $d(P_{new}, P_1) = \sqrt{(3-1)^2 + (5-2)^2} = \sqrt{4+9} = \sqrt{13} \approx 3.61$
- $d(P_{new}, P_2) = \sqrt{(3-2)^2 + (5-4)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$
- $d(P_{new}, P_3) = \sqrt{(3-4)^2 + (5-1)^2} = \sqrt{1+16} = \sqrt{17} \approx 4.12$
- $d(P_{new}, P_4) = \sqrt{(3-5)^2 + (5-4)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$
- $d(P_{new}, P_5) = \sqrt{(3-4)^2 + (5-6)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$
- $d(P_{new}, P_6) = \sqrt{(3-6)^2 + (5-2)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$

Ranking by distance (smallest first):

1. P_2 (1.41, Class A)
2. P_5 (1.41, Class A)
3. P_4 (2.24, Class B)
4. P_1 (3.61, Class A)
5. P_3 (4.12, Class B)
6. P_6 (4.24, Class B)

The 3 nearest neighbors are P_2, P_5, and P_4. Their classes are A, A, B. By majority vote (2 A's, 1 B), P_{new} is classified as **Class A**.

Part 2: Using Manhattan Distance The Manhattan distance is $d = |x_2 - x_1| + |y_2 - y_1|$.

- $d(P_{new}, P_1) = |3-1| + |5-2| = 2+3 = 5$
- $d(P_{new}, P_2) = |3-2| + |5-4| = 1+1 = 2$
- $d(P_{new}, P_3) = |3-4| + |5-1| = 1+4 = 5$

- $d(P_{new}, P_4) = |3 - 5| + |5 - 4| = 2 + 1 = 3$
- $d(P_{new}, P_5) = |3 - 4| + |5 - 6| = 1 + 1 = 2$
- $d(P_{new}, P_6) = |3 - 6| + |5 - 2| = 3 + 3 = 6$

Ranking by distance (smallest first):

1. P_2 (2, Class A)
2. P_5 (2, Class A)
3. P_4 (3, Class B)
4. P_1 (5, Class A)
5. P_3 (5, Class B)
6. P_6 (6, Class B)

The 3 nearest neighbors are P_2, P_5, and P_4. Their classes are A, A, B. By majority vote (2 A's, 1 B), P_{new} is classified as **Class A**.

In this case, both distance metrics lead to the same classification.

Question 7 (Scenario)

Problem: You are developing a system to recommend news articles to users. The articles are represented as high-dimensional TF-IDF vectors. Why would Euclidean distance be a poor choice for measuring similarity between articles in this context? What distance metric would be more appropriate and why?

Solution

Why Euclidean Distance is a Poor Choice:

1. **Curse of Dimensionality:** In high-dimensional spaces, like those created by TF-IDF vectors where each dimension corresponds to a word in the vocabulary, the concept of distance becomes less meaningful. The distance between any two points tends to become very similar, making it hard to find "nearest" neighbors.
2. **Ignores Direction/Orientation:** Euclidean distance measures the magnitude of the difference between two vectors. Two articles could have TF-IDF vectors that are far apart in Euclidean space simply because one article is much longer than the other, even if they discuss the exact same topic. For example, a short article about "machine learning" and a very long, comprehensive article about "machine learning" would be treated as very dissimilar.
3. **Sparsity:** TF-IDF vectors are typically very sparse (most entries are zero). Euclidean distance doesn't handle this sparsity well; the many zero-valued dimensions can dominate the calculation.

More Appropriate Metric: Cosine Similarity

1. **What it Measures:** Cosine similarity measures the cosine of the angle between two vectors. It is calculated as:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||}$$

2. **Why it's Better:**
   - **Focuses on Orientation, Not Magnitude:** Cosine similarity is concerned with the direction of the vectors, not their length. In the context of text, this means it measures the similarity of the topics/content regardless of the document's length. The two "machine learning" articles from the previous example would have a very high cosine similarity (close to 1) because their vectors point in the same direction in the vector space, indicating they use a similar distribution of words.
   - **Robust to High Dimensions and Sparsity:** It is computationally efficient and works well in high-dimensional, sparse settings like TF-IDF, as the dot product is only affected by the non-zero dimensions.

Therefore, for document similarity tasks, **cosine similarity** is the standard and more appropriate choice.

Question 8 (Conceptual/Applied)

Problem: Explain the core idea behind Locally Weighted Regression (LWR). How does it differ from standard linear regression, and in what kind of problem would it be more suitable?

Solution

Core Idea of Locally Weighted Regression (LWR): LWR is a non-parametric regression method. Unlike standard linear regression, which tries to find a single global model (a single line or hyperplane) that fits all the data, LWR computes a unique local model for every new query point you want to make a prediction for.

The process for predicting a value for a query point x_q is:

1. **Assign Weights:** For the query point x_q, assign a weight to every training point (x_i, y_i) in the dataset. The weight is highest for points close to x_q and decreases as points get farther away. A common weighting function is a Gaussian (or Radial Basis) kernel.

2. **Fit a Weighted Model:** Fit a linear regression model to the dataset, but minimize a *weighted* sum of squared errors. The training points that are closer to x_q (and thus have higher weights) will have a much greater influence on the resulting regression line than points far away.

3. **Make a Prediction:** Use the resulting locally-fit regression model to predict the output for the query point x_q.

4. **Discard the Model:** This local model is then discarded. A new one is computed from scratch for the next query point.

Differences from Standard Linear Regression:

- **Global vs. Local:** Standard linear regression computes one global set of parameters (θ) for the entire dataset. LWR computes a new set of parameters for each local prediction.

- **Training Phase:** Standard linear regression has a distinct training phase to find the optimal parameters. LWR has no "training" phase in the traditional sense; all the computation happens at prediction time. This is why it's called a "lazy" learning algorithm.

- **Model Complexity:** Standard linear regression models a simple linear relationship. LWR can model complex, non-linear functions because the local approximations can stitch together to form a complex curve.

When is LWR More Suitable? LWR is more suitable when:

- The underlying relationship between features and the target is known to be **non-linear and complex**, and a simple global model would have high bias.
- The dataset is **not excessively large**, as the computational cost of LWR at prediction time is very high (it needs to access the entire dataset for every prediction).
- You have enough data density across the feature space, as LWR needs sufficient "local" neighbors to make a good prediction.

Question 9 (Numerical)

Problem: You are using k-NN regression with $k = 3$ to predict the price of a house based on its size (in sq. ft.). Given the following training data, what is the predicted price for a new 2000 sq. ft. house?

Size (sq. ft.)	**Price ($1000s)**
1500	300
1800	350
1900	400
2200	500
2500	520
3000	600

Solution

Step 1: Identify the query point. The query point is a house of size $x_q = 2000$ sq. ft.

Step 2: Calculate the distance from the query point to all training points. Since this is a 1D problem, the distance is simply the absolute difference in size.

- $|2000 - 1500| = 500$
- $|2000 - 1800| = 200$
- $|2000 - 1900| = 100$
- $|2000 - 2200| = 200$
- $|2000 - 2500| = 500$
- $|2000 - 3000| = 1000$

Step 3: Find the k-nearest neighbors. For $k = 3$, we need the 3 houses with the smallest distance in size.

1. The house of 1900 sq. ft. (distance 100)
2. The house of 1800 sq. ft. (distance 200)
3. The house of 2200 sq. ft. (distance 200)

Step 4: Predict the value for the query point. In k-NN regression, the prediction is typically the average of the values of the k-nearest neighbors.

- The prices of the 3 nearest neighbors are $400k, $350k, and $500k.
- Predicted Price = $(400 + 350 + 500)/3 = 1250/3 \approx 416.67$

The predicted price for a 2000 sq. ft. house is **$416,670**.

Question 10 (Scenario)

Problem: Describe a scenario where choosing a very small value of 'k' (e.g., k=1) in k-NN would be problematic. What does this decision relate to in terms of the bias-variance tradeoff?

Solution

Scenario: Medical Diagnosis from Patient Data Imagine you are building a k-NN classifier to predict whether a patient has a certain disease based on features like blood pressure, cholesterol level, age, etc. The dataset contains some natural noise and outliers. For instance, there might be a few healthy patients who happen to have measurement values similar to those of sick patients due to data entry errors or unusual temporary conditions.

Problem with k=1 (1-NN): If you use $k = 1$, the model's prediction for a new patient is determined entirely by the class of the single closest patient in the training data.

- **High Sensitivity to Noise:** If a new patient's data is closest to one of these noisy or outlier points, the model will make an incorrect prediction. For example, a new healthy patient might be incorrectly classified as "sick" simply because their nearest neighbor in the feature space is a single, anomalous sick patient.
- **Overfitting:** The decision boundary created by a 1-NN classifier is highly irregular and complex, as it contorts itself to perfectly classify every single point in the training data. This is a classic example of overfitting. The model has learned the noise in the training data, not the underlying pattern.

Relation to Bias-Variance Tradeoff: This choice directly relates to the bias-variance tradeoff:

- **Low Bias:** A model with $k = 1$ has very low bias. It has high capacity and can fit the training data perfectly, making very few errors on the data it has already seen.
- **High Variance:** The model has extremely high variance. A small change in the training data (e.g., adding or removing a single outlier) could drastically change the decision boundary and the model's predictions. The model does not generalize well to new, unseen data.

Conclusion: Choosing a very small 'k' leads to a model with low bias but high variance, making it susceptible to noise and causing it to overfit the training data. Increasing 'k' increases the bias (as the decision boundary becomes smoother and less flexible) but decreases the variance (as the prediction is averaged over more neighbors, making it more stable and robust to outliers).

3 Support Vector Machines (SVMs)

Question 11 (Numerical)

Problem: Given the following linearly separable 2D data, find the equation of the maximum margin hyperplane. Identify the support vectors.

- Class +1: (2, 3), (3, 3)
- Class -1: (1, 1), (2, 1)

Solution

Step 1: Visualize the data. Plotting the points, we can see they are linearly separable. The separating hyperplane will be a line that passes between the two classes. By inspection, a horizontal line seems appropriate.

Step 2: Determine the form of the hyperplane. The hyperplane is defined by $w \cdot x + b = 0$. For a 2D line, this is $w_1 x_1 + w_2 x_2 + b = 0$. The margins are defined by $w \cdot x + b = 1$ and $w \cdot x + b = -1$. The points on these lines are the support vectors.

Step 3: Identify candidate support vectors and solve for w and b. Let's assume the support vectors are (2, 3) from class +1 and (2, 1) from class -1. The margin planes must pass through these points.

1. For (2, 3) [class +1]: $w \cdot (2, 3) + b = 1 \implies 2w_1 + 3w_2 + b = 1$
2. For (2, 1) [class -1]: $w \cdot (2, 1) + b = -1 \implies 2w_1 + 1w_2 + b = -1$

By inspection, a horizontal separating line would have a weight vector $w = (0, w_2)$. Let's substitute $w_1 = 0$ into the equations:

1. $3w_2 + b = 1$
2. $w_2 + b = -1 \implies b = -1 - w_2$

Substitute (2) into (1): $3w_2 + (-1 - w_2) = 1 \implies 2w_2 = 2 \implies w_2 = 1$. Now find b: $b = -1 - w_2 = -1 - 1 = -2$. So, the weight vector is $w = (0, 1)$ and the bias is $b = -2$.

Step 4: Write the equation of the hyperplane. The equation is $w \cdot x + b = 0$. $(0, 1) \cdot (x_1, x_2) - 2 = 0 \implies x_2 - 2 = 0 \implies \boldsymbol{x_2 = 2}$.

Step 5: Identify the support vectors. The support vectors are the points that lie on the margin hyperplanes $x_2 - 2 = 1$ (i.e., $x_2 = 3$) and $x_2 - 2 = -1$ (i.e., $x_2 = 1$).

- From class +1, the points with $x_2 = 3$ are (2, 3) and (3, 3).
- From class -1, the points with $x_2 = 1$ are (1, 1) and (2, 1).

All four points are support vectors in this case.

Step 6: Calculate the margin. The margin is $2/\|w\| = 2/\sqrt{0^2 + 1^2} = 2$.

Question 12 (Scenario)

Problem: You are tasked with building an image classifier to distinguish between cats and dogs. The images are high-dimensional (e.g., 256x256 pixels). Explain how an SVM with the "kernel trick" could be effective for this non-linearly separable problem. What specific kernel might be a good starting point and why?

Solution

The Challenge: Images of cats and dogs are not linearly separable in the raw pixel space. The boundary that separates "cat pixels" from "dog pixels" is incredibly complex and non-linear. A simple linear SVM would fail spectacularly.

How the Kernel Trick Helps: The kernel trick is the core idea that allows SVMs to solve non-linear problems.

1. **Implicit High-Dimensional Mapping:** Instead of trying to find a complex boundary in the original, low-dimensional pixel space, the kernel trick implicitly maps the data to a much higher-dimensional feature space. The key insight is that data which is not linearly separable in a low-dimensional space can become linearly separable in a higher-dimensional one.

2. **Avoiding Explicit Computation:** The "trick" is that we never actually have to compute the coordinates of the data points in this high-dimensional space. The SVM algorithm only requires dot products between data points. A kernel function is a function that computes this dot product in the high-dimensional space directly from the original vectors. This is computationally much cheaper.

3. **Finding a Linear Separator:** Once the data is implicitly mapped to this high-dimensional space, the SVM simply finds the maximum-margin linear hyperplane there. This linear separator in the high-dimensional space corresponds to a complex, non-linear decision boundary back in the original pixel space.

Good Starting Kernel: Radial Basis Function (RBF) Kernel The RBF kernel (also called the Gaussian kernel) is often the default choice for problems like image classification.

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

Why it's a good choice:

1. **Infinite-Dimensional Space:** The RBF kernel corresponds to mapping the data into an infinite-dimensional feature space. This provides immense flexibility to create a very complex decision boundary, which is necessary for tasks like distinguishing cats from dogs.

2. **Locality:** The kernel's value depends on the squared Euclidean distance between two points (x_i, x_j). This means its notion of "similarity" is based on how close two images are in the pixel space. It creates a localized and smooth decision boundary.

3. **Fewer Hyperparameters:** The RBF kernel primarily has one key hyperparameter, γ (gamma), which controls the "width" of the Gaussian bumps. A small γ gives a broader influence, while a large γ gives a more localized influence. Along with the SVM's regularization parameter C, this is a manageable number of hyperparameters to tune.

Question 13 (Conceptual/Applied)

Problem: Explain the role of the regularization parameter 'C' and the slack variables 'ξ_i' (xi) in a soft-margin SVM. How does tuning 'C' affect the model's bias and variance?

Solution

In real-world data, perfect linear separability is rare. A soft-margin SVM is introduced to handle non-linearly separable data or data with outliers by allowing some misclassifications.

Role of Slack Variables (ξ_i):

- A slack variable $\xi_i \geq 0$ is introduced for each data point x_i.
- ξ_i measures the degree to which the point x_i violates the margin.
  - If $\xi_i = 0$, the point is correctly classified and is either on or outside the correct margin.

  - If $0 < \xi_i \leq 1$, the point is correctly classified but is inside the margin (a margin violation).
  - If $\xi_i > 1$, the point is on the wrong side of the hyperplane (a misclassification).

- The SVM optimization goal is modified to not only maximize the margin but also to minimize the total amount of slack, $\sum \xi_i$.

Role of the Regularization Parameter (C):

- The parameter $C > 0$ is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error (represented by the sum of slack variables).
- The optimization objective looks something like:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

- C can be thought of as a "penalty" for misclassification.

Effect of Tuning C on Bias and Variance:

- **Small C:**
  - The penalty for misclassification is low. The optimization prioritizes finding a large margin, even if that means misclassifying more points.
  - The resulting decision boundary is simpler and smoother.
  - This leads to a model with **high bias** (it might underfit the data) and **low variance** (it's less sensitive to individual data points).

- **Large C:**
  - The penalty for misclassification is high. The optimization will try very hard to classify every point correctly, even if it means choosing a smaller margin.
  - The resulting decision boundary can be highly complex and contorted to fit the training data, including outliers.
  - This leads to a model with **low bias** (it fits the training data very well) and **high variance** (it's very sensitive to the training data and may not generalize well). This is overfitting.

Question 14 (Numerical/Scenario)

Problem: Imagine you have the following 1D dataset which is not linearly separable.

- Class +1: {-4, 4}
- Class -1: {-1, 1}

Show how mapping this data into a 2D feature space using the mapping function $\phi(x) = (x, x^2)$ makes the data linearly separable.

Solution

Step 1: Understand the initial data. The data points on a number line are arranged as [+1, -1, -1, +1]. You cannot draw a single point (a 0D hyperplane) to separate the +1 class from the -1 class.

Step 2: Apply the mapping function $\phi(x) = (x, x^2)$ to each data point. We will transform each 1D point x into a 2D point (x_1, x_2) where $x_1 = x$ and $x_2 = x^2$.

- **For Class +1:**
  - $x = -4 \rightarrow \phi(-4) = (-4, (-4)^2) = (-4, 16)$
  - $x = 4 \rightarrow \phi(4) = (4, (4)^2) = (4, 16)$
- **For Class -1:**
  - $x = -1 \rightarrow \phi(-1) = (-1, (-1)^2) = (-1, 1)$
  - $x = 1 \rightarrow \phi(1) = (1, (1)^2) = (1, 1)$

Step 3: Analyze the new data in the 2D feature space. The transformed points are:

- Class +1: {(-4, 16), (4, 16)}
- Class -1: {(-1, 1), (1, 1)}

If we plot these points on a standard 2D graph, we can see that the Class +1 points are high up on the y-axis (at $y = 16$), and the Class -1 points are low on the y-axis (at $y = 1$).

Step 4: Find a linear separator in the new space. The data is now clearly linearly separable. We can easily draw a horizontal line between the two classes. For example, the line $x_2 = 8$ (where x_2 is the second dimension of our new space) perfectly separates the classes. All points with $x_2 > 8$ belong to Class +1, and all points with $x_2 < 8$ belong to Class -1.

This demonstrates the core concept of the kernel trick: a non-linear problem in a lower-dimensional space can be transformed into a linear problem in a higher-dimensional space.

Question 15 (Scenario)

Problem: Your company wants to use SVMs for real-time fraud detection on millions of transactions per day. What are the potential challenges regarding scalability and efficiency, and how might they be addressed?

Solution

Challenges with SVM Scalability and Efficiency:

1. **Training Complexity:** The computational complexity of training a standard SVM is typically between $O(n^2 d)$ and $O(n^3 d)$, where n is the number of training samples and d is the number of features. For millions of transactions, this is computationally infeasible. The memory requirement to store the kernel matrix ($O(n^2)$) is also prohibitive.
2. **Prediction (Inference) Complexity:** For a kernelized SVM, the prediction time for a new data point is proportional to the number of support vectors. In complex problems, the number of support vectors can be a significant fraction of the training set size, making real-time prediction slow. If prediction for one transaction requires comparing it to thousands of support vectors, it won't meet real-time requirements.

Potential Solutions and Mitigations:

1. **Use Linear SVMs:** If the data is high-dimensional and might be linearly separable (or close to it), a linear SVM (with no kernel) is much faster to train. The training complexity can be reduced to approximately $O(nd)$. For text or other high-dimensional sparse data, this is often a very strong baseline.
2. **Stochastic Gradient Descent (SGD) Solvers:** Instead of using the standard quadratic programming solvers, algorithms like 'SGDClassifier' in scikit-learn can be used to train linear SVMs (or other linear models) efficiently on very large datasets. It processes one instance (or a mini-batch) at a time, making it scalable.
3. **Kernel Approximation / Random Projections:** For non-linear problems, instead of using the full kernel trick, we can use techniques like Random Fourier Features or Nystroem method. These methods create an explicit, approximate mapping to a lower-dimensional feature space where a fast linear SVM can then be trained. This allows you to get the power of non-linear kernels with the speed of linear models.
4. **Subsampling / Data Reduction:** Train the SVM on a smaller, representative subset of the data. This is a simple but often effective strategy. Advanced techniques can be used to intelligently select the most informative samples to include in the training set.
5. **Online Learning:** For a stream of transactions, an online SVM approach can be used. The model is continuously updated as new data arrives, without needing to be retrained from scratch on the entire historical dataset.
6. **Model Distillation:** Train a large, complex SVM offline. Then, train a much smaller, faster model (like a small neural network or a linear model) to mimic the predictions of the large SVM. This smaller "student" model is then deployed for real-time inference.

4 Bayesian Learning

Question 16 (Numerical)

Problem: You flip a thumbtack and observe the sequence: Up, Down, Up, Up, Up.

1. What is the Maximum Likelihood Estimation (MLE) for the probability of the thumbtack landing 'Up' (θ)?
2. Suppose you have a prior belief that θ follows a Beta distribution, $Beta(\alpha = 3, \beta = 2)$. What is the Maximum A Posteriori (MAP) estimate for θ?

Solution

Let N_U be the number of 'Up' outcomes and N_D be the number of 'Down' outcomes. From the data, $N_U = 4$ and $N_D = 1$. Total flips $N = 5$.

Part 1: Maximum Likelihood Estimation (MLE)

- The likelihood of observing this specific sequence of data given θ is:

$$P(D|\theta) = \theta \cdot (1 - \theta) \cdot \theta \cdot \theta \cdot \theta = \theta^4(1 - \theta)^1$$

- The MLE finds the value of θ that maximizes this likelihood function. For a binomial process, this is simply the observed proportion of successes.

- $\hat{\theta}_{MLE} = \frac{N_U}{N_U+N_D} = \frac{4}{4+1} = \frac{4}{5} = 0.8$.

Part 2: Maximum A Posteriori (MAP) Estimation

- The MAP estimate aims to maximize the posterior probability $P(\theta|D)$, which is proportional to the likelihood times the prior: $P(\theta|D) \propto P(D|\theta)P(\theta)$.
- We are given a prior $P(\theta) \sim Beta(\alpha = 3, \beta = 2)$. The probability density function for a Beta distribution is proportional to $\theta^{\alpha-1}(1-\theta)^{\beta-1}$.
- Therefore, the posterior is:

$$P(\theta|D) \propto [\theta^{N_U}(1-\theta)^{N_D}] \times [\theta^{\alpha-1}(1-\theta)^{\beta-1}]$$
$$P(\theta|D) \propto \theta^{N_U+\alpha-1}(1-\theta)^{N_D+\beta-1}$$

- This shows that the posterior distribution is also a Beta distribution, specifically $Beta(N_U + \alpha, N_D + \beta)$. This is because the Beta distribution is the conjugate prior for the Bernoulli/Binomial likelihood.
- The mode (the peak) of a Beta(α', β') distribution gives the MAP estimate:

$$\hat{\theta}_{MAP} = \frac{\alpha' - 1}{\alpha' + \beta' - 2}$$

- Our posterior parameters are $\alpha' = N_U + \alpha = 4 + 3 = 7$ and $\beta' = N_D + \beta = 1 + 2 = 3$.
- $$\hat{\theta}_{MAP} = \frac{7-1}{7+3-2} = \frac{6}{8} = 0.75$$

Comparison: The MLE is 0.8, purely based on the data. The MAP estimate is 0.75, which is "pulled" away from the MLE towards the mean of the prior distribution (which was centered around $(\alpha - 1)/(\alpha + \beta - 2) = 2/3 \approx 0.67$).

Question 17 (Numerical)

Problem: Given the following dataset, use the Naïve Bayes classifier to predict whether to play tennis on a new day with the conditions: 'Outlook=Sunny', 'Temp=Cool', 'Humidity=High', 'Wind=Strong'. Use Laplace (add-1) smoothing.

Outlook	Temp	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Solution

Let X = (Outlook=Sunny, Temp=Cool, Humidity=High, Wind=Strong). We want to find the class (Yes or No) that maximizes $P(\text{Class}|X) \propto P(X|\text{Class})P(\text{Class})$.

Step 1: Calculate Prior Probabilities P(Class)

- Total instances: 14
- Play=Yes: 9 instances. $P(\text{Yes}) = 9/14$
- Play=No: 5 instances. $P(\text{No}) = 5/14$

Step 2: Calculate Likelihoods for Class = Yes ($N_{Yes} = 9$) (Using Laplace smoothing: $P(\text{feature} = v|\text{class}) = \frac{\text{count}(v,\text{class})+1}{\text{count(class)}+k}$, where k is the number of values for the feature)

- $P(\text{Outlook=Sunny}|\text{Yes}) = \frac{2+1}{9+3} = \frac{3}{12}$ (k=3 for Outlook)
- $P(\text{Temp=Cool}|\text{Yes}) = \frac{3+1}{9+3} = \frac{4}{12}$ (k=3 for Temp)
- $P(\text{Humidity=High}|\text{Yes}) = \frac{3+1}{9+2} = \frac{4}{11}$ (k=2 for Humidity)
- $P(\text{Wind=Strong}|\text{Yes}) = \frac{3+1}{9+2} = \frac{4}{11}$ (k=2 for Wind)

Step 3: Calculate Likelihoods for Class = No ($N_{No} = 5$)

- $P(\text{Outlook=Sunny}|\text{No}) = \frac{3+1}{5+3} = \frac{4}{8}$
- $P(\text{Temp=Cool}|\text{No}) = \frac{1+1}{5+3} = \frac{2}{8}$
- $P(\text{Humidity=High}|\text{No}) = \frac{4+1}{5+2} = \frac{5}{7}$
- $P(\text{Wind=Strong}|\text{No}) = \frac{3+1}{5+2} = \frac{4}{7}$

Step 4: Calculate Posterior Proportional Scores

- **For Play=Yes:** $P(\text{Yes}|X) \propto P(\text{Yes}) \cdot P(\text{Sunny}|\text{Yes}) \cdot P(\text{Cool}|\text{Yes}) \cdot P(\text{High}|\text{Yes}) \cdot P(\text{Strong}|\text{Yes})$ $\propto \frac{9}{14} \cdot \frac{3}{12} \cdot \frac{4}{12} \cdot \frac{4}{11} \cdot \frac{4}{11} \approx 0.0053$
- **For Play=No:** $P(\text{No}|X) \propto P(\text{No}) \cdot P(\text{Sunny}|\text{No}) \cdot P(\text{Cool}|\text{No}) \cdot P(\text{High}|\text{No}) \cdot P(\text{Strong}|\text{No})$ $\propto \frac{5}{14} \cdot \frac{4}{8} \cdot \frac{2}{8} \cdot \frac{5}{7} \cdot \frac{4}{7} \approx 0.0182$

Step 5: Make the Prediction Since $0.0182 > 0.0053$, the model predicts that ‘Play Tennis‘ = **No**.

Question 18 (Scenario)

Problem: The “naïve” assumption in the Naïve Bayes classifier is that all features are conditionally independent given the class. Provide a real-world example where this assumption is clearly violated, and explain why the classifier might still perform surprisingly well.

Solution

Real-World Example of Violated Assumption: Spam Detection A common application of Naïve Bayes is text classification, such as detecting spam emails. The features are the presence or absence of words in an email (a "bag-of-words" model).

- **The Violation:** The assumption of conditional independence means that the probability of seeing the word "Viagra" is independent of the probability of seeing the word "buy", given that the email is spam. This is clearly false. Certain words are highly correlated and tend to appear together. The phrase "buy now" is far more likely to appear than the two words independently. The presence of "Hong" makes the presence of "Kong" much more likely.

Why It Can Still Perform Well:

1. **Classification vs. Probability Estimation:** The goal of the classifier is not to produce accurate probability estimates, but simply to make the correct classification. It only needs to determine which class has the *higher* posterior probability. The independence assumption might be wrong, causing the calculated probabilities to be systematically skewed (e.g., pushed towards 0 or 1), but as long as the correct class ends up with the highest score, the classification is correct.

2. **Error Cancellation:** Even if dependencies exist, their effects might "cancel each other out" during the classification process. Some word dependencies might increase the score for the "spam" class, while other dependencies might decrease it, and the overall effect on the final decision boundary might be minimal.

3. **Dominance of Evidence:** In many cases, the evidence for a class is overwhelming. A spam email might contain dozens of words ("free", "Viagra", "money", "offer", "click") that are each strongly indicative of spam. Even if their probabilities are multiplied together incorrectly due to the independence assumption, the sheer volume of evidence will still push the final score for "spam" far above the score for "not spam". The decision doesn't require a perfectly calibrated probability, just a clear winner.

Essentially, Naïve Bayes often works well because its decision boundary can be reasonable even if its underlying probabilistic assumptions are flawed.

Question 19 (Conceptual/Applied)

Problem: Differentiate between a generative and a discriminative classifier. Classify Naïve Bayes and Logistic Regression into these categories and justify your classification.

Solution

Differentiation: The core difference lies in what probability distribution they model.

- **Generative Classifiers:**
  - **What they model:** They model the joint probability distribution $P(X, Y)$, where X are the features and Y is the class label. They learn how the data for each class is "generated".

- **How they classify:** They use Bayes' rule to calculate the posterior probability $P(Y|X)$ from the joint probability. The formula is:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

  They learn a model for the likelihood $P(X|Y)$ and the prior $P(Y)$.
- **Analogy:** A generative model learns what a "cat" looks like and what a "dog" looks like. To classify a new image, it checks whether it looks more like the "cat" model or the "dog" model.

- **Discriminative Classifiers:**
  - **What they model:** They directly model the conditional probability distribution $P(Y|X)$. They are not interested in how the data was generated, only in learning the boundary between classes.
  - **How they classify:** They learn a direct mapping from the features X to the class label Y.
  - **Analogy:** A discriminative model doesn't learn what cats or dogs look like. It only learns the specific features that are different between them (e.g., whisker length, snout shape). It learns to "discriminate" between the two classes.

Classification and Justification:

- **Naïve Bayes is a Generative Classifier.**
  - **Justification:** Naïve Bayes explicitly models the components needed for the joint distribution. It calculates the class priors $P(Y)$ and the class-conditional likelihoods of the features $P(X|Y)$ (under the naïve independence assumption). It then uses these two parts with Bayes' rule to make a prediction. It learns a model of how the features are generated for each class.

- **Logistic Regression is a Discriminative Classifier.**
  - **Justification:** Logistic Regression directly models $P(Y|X)$ using the sigmoid (logistic) function:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\ldots)}}$$

    It finds the parameters (β) that best fit this direct relationship. It never attempts to model $P(X)$ or $P(X|Y)$. It only learns the decision boundary that separates the classes.

Question 20 (Numerical)

Problem: Consider a Bayesian Network with the structure '(Rain) -¿ (Wet Grass) ¡- (Sprinkler)'. The conditional probability tables (CPTs) are given below. What is the probability that the grass is wet, given that it is raining and the sprinkler is on? i.e., calculate $P(W = T|R = T, S = T)$.

P(Rain)

P(R=T)	P(R=F)
0.2	0.8

P(Sprinkler)

P(S=T)	P(S=F)
0.1	0.9

P(Wet Grass — Rain, Sprinkler)

Rain	Sprinkler	P(W=T)
F	F	0.0
F	T	0.9
T	F	0.9
T	T	0.99

Solution

This problem is simpler than it appears and is designed to test understanding of how to read and use a Bayesian Network's Conditional Probability Tables (CPTs).

Step 1: Identify the required probability. We need to find $P(\text{Wet Grass} = \text{True}|\text{Rain} = \text{True}, \text{Spri}$

Step 2: Understand the network structure. The variable ‘Wet Grass‘ is directly conditioned on its parents, ‘Rain‘ and ‘Sprinkler‘. This means that if we know the state of both parents, the probability of ‘Wet Grass‘ is given directly by its CPT and is independent of any other information.

Step 3: Look up the value in the CPT. We go to the CPT for ‘P(Wet Grass — Rain, Sprinkler)‘. We find the row that corresponds to the condition where ‘Rain = T‘ and ‘Sprinkler = T‘.

Rain	Sprinkler	P(W=T)
...	...	...
T	T	**0.99**

Step 4: State the result. The CPT directly gives us the answer. The probability that the grass is wet, given that it is raining and the sprinkler is on, is **0.99**. The prior probabilities for Rain and Sprinkler are not needed for this specific query.

5 Ensemble Learning

Question 21 (Numerical/Scenario)

Problem: You are applying AdaBoost with decision stumps (1-level decision trees) to the following 1D dataset. Perform the first two iterations of the algorithm. For each iteration, show the chosen stump, its error, its weight (α_t), and the new sample weights.

x	1	2	3	4	5	6
y	+1	+1	-1	-1	+1	+1

Solution

Iteration 1 (t=1):

- **Initialize Weights:** All N=6 samples start with equal weight. $w_i^{(1)} = 1/6$ for all i. Weights: [0.167, 0.167, 0.167, 0.167, 0.167, 0.167]
- **Find Best Stump:** We test all possible stumps (splits).
  - Split at $x < 3.5$: Predicts +1 for {1,2,3}, -1 for {4,5,6}. Misclassifies {3,5,6}. Error = 3 * (1/6) = 0.5.
  - Split at $x < 2.5$: Predicts +1 for {1,2}, -1 for {3,4,5,6}. Misclassifies {5,6}. Error = 2 * (1/6) = 0.333.

  - Split at $x < 4.5$: Predicts +1 for {1,2,3,4}, -1 for {5,6}. Misclassifies {3,4,5,6}. Error = 4 * (1/6) = 0.667.

    Let's choose the stump: **if $x < 2.5$, predict +1, else predict -1**.

    * Predictions: [+1, +1, -1, -1, -1, -1]
    * True Labels: [+1, +1, -1, -1, +1, +1]
    * Misclassifies: points 5 and 6.
    * Error $\epsilon_1 = w_5 + w_6 = 1/6 + 1/6 = 1/3 \approx 0.333$.

- **Calculate Stump Weight (α_1):**

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \frac{1}{2}\ln\left(\frac{1-1/3}{1/3}\right) = \frac{1}{2}\ln(2) \approx 0.347$$

- **Update Sample Weights:**

$$w_i^{(t+1)} = \frac{w_i^{(t)}\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  Where $y_i h_t(x_i)$ is +1 for correctly classified points and -1 for misclassified points.

  - For correctly classified points {1,2,3,4}: $w_{new} = w_{old} \cdot e^{-0.347} \approx 0.167 \cdot 0.707 = 0.118$
  - For misclassified points {5,6}: $w_{new} = w_{old} \cdot e^{0.347} \approx 0.167 \cdot 1.414 = 0.236$

- Unnormalized weights: [0.118, 0.118, 0.118, 0.118, 0.236, 0.236]
- Normalization factor $Z_1 = 4(0.118) + 2(0.236) = 0.472 + 0.472 = 0.944$.
- **New weights** $w^{(2)}$: [0.125, 0.125, 0.125, 0.125, 0.25, 0.25]

Iteration 2 (t=2):

- **Current Weights:** [0.125, 0.125, 0.125, 0.125, 0.25, 0.25]
- **Find Best Stump:** Now we find the best stump using the new weights. The points {5,6} that were misclassified now have double the weight.

  - Let's try the stump: **if $x > 4.5$, predict +1, else predict -1**.
  - Predictions: [-1, -1, -1, -1, +1, +1]
  - True Labels: [+1, +1, -1, -1, +1, +1]
  - Misclassifies: points 1 and 2.
  - Error $\epsilon_2 = w_1^{(2)} + w_2^{(2)} = 0.125 + 0.125 = 0.25$. (This is likely the best stump as it correctly classifies the high-weight points).

- **Calculate Stump Weight (α_2):**

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.25}{0.25}\right) = \frac{1}{2}\ln(3) \approx 0.549$$

- **Update Sample Weights:**

  - Misclassified {1,2}: $w_{new} = 0.125 \cdot e^{0.549} \approx 0.125 \cdot 1.73 = 0.216$

  - Correct {3,4,5,6}: $w_{3,4} = 0.125 \cdot e^{-0.549} \approx 0.125 \cdot 0.578 = 0.072$ $w_{5,6} = 0.25 \cdot e^{-0.549} \approx 0.25 \cdot 0.578 = 0.144$

- Unnormalized weights: [0.216, 0.216, 0.072, 0.072, 0.144, 0.144]
- Normalization factor $Z_2 = 2(0.216) + 2(0.072) + 2(0.144) = 0.432 + 0.144 + 0.288 = 0.864$.
- **New weights** $w^{(3)}$: [0.25, 0.25, 0.083, 0.083, 0.167, 0.167]

The final classifier is a weighted combination of these two (and subsequent) stumps.

Question 22 (Scenario)

Problem: Explain the key differences between Bagging (as used in Random Forest) and Boosting (as used in AdaBoost). Focus on how they handle bias and variance and when you would prefer one over the other.

Solution

Key Differences:

Aspect	**Bagging (e.g., Random Forest)**	**Boosting (e.g., AdaBoost)**
Model Training	Models are trained in **parallel**. Each model is trained independently on a different bootstrap sample of the data.	Models are trained **sequentially**. Each new model is trained to correct the errors made by the previous models.
Sample Weighting	All training samples have an **equal** likelihood of being selected in a bootstrap sample.	Sample weights are **adjusted**. Misclassified samples from the previous iteration are given higher weights to force the next model to focus on them.
Model Weighting	All models have an **equal vote** in the final prediction (majority vote or average).	Models are weighted based on their performance. Better-performing models have a **higher vote** in the final prediction.
Primary Goal	To **reduce variance**.	To **reduce bias**.

Handling of Bias and Variance:

- **Bagging / Random Forest:** This technique is most effective with low-bias, high-variance base models, such as fully grown decision trees. By training many deep trees on different subsets of data and averaging their predictions, the variance is reduced significantly. The individual trees are prone to overfitting (high variance), but the ensemble as a whole generalizes well. The bias remains roughly the same as that of a single tree.
- **Boosting / AdaBoost:** This technique is most effective with high-bias, low-variance base models (weak learners), such as decision stumps. Each sequential model focuses on the errors of the previous one, gradually reducing the overall error and bias of the ensemble. By combining many weak learners, Boosting can create a single strong learner with low bias. However, it can be more prone to overfitting if the number of trees is too high, as it will start fitting the noise in the data.

When to Prefer One Over the Other:

- **Prefer Bagging/Random Forest when:**
  - The main concern is **overfitting** (high variance).
  - You want a model that is robust, stable, and easy to train in parallel.
  - The dataset might be noisy, as averaging helps to cancel out the noise.
- **Prefer Boosting when:**
  - The main concern is **underfitting** (high bias) and you want the highest possible predictive accuracy.
  - You have cleaner data, as boosting can overfit noisy data by trying too hard to classify every point correctly.
  - You have more computational budget for sequential training.

Question 23 (Conceptual/Applied)

Problem: Random Forest uses two key mechanisms of randomness: 1) training each tree on a bootstrap sample of the data, and 2) considering only a random subset of features at each split. Explain the purpose of each of these mechanisms.

Solution

Both mechanisms are designed to **decorrelate** the individual trees in the forest, which is the essential ingredient for a successful bagging-style ensemble.

1. Bootstrap Sampling (Row Sampling):

- **Mechanism:** Each tree in the forest is trained on a different random sample of the training data, drawn with replacement. This is called a bootstrap sample. On average, each bootstrap sample contains about 63.2% of the original training data.
- **Purpose:** This is the "Bagging" (Bootstrap Aggregating) part of the algorithm. If all trees were trained on the exact same dataset, they would all be very similar (or identical, if the algorithm is deterministic). This would be like asking the same person the same question multiple times – you'd get the same, potentially wrong, answer. By training each tree on a slightly different dataset, the trees will learn slightly different patterns and make different errors. When their predictions are averaged, these errors tend to cancel out, which is what reduces the overall variance of the model.

2. Random Feature Subspace (Column Sampling):

- **Mechanism:** At each node in a tree, when deciding on the best split, the algorithm does not consider all available features. Instead, it selects a random subset of features (e.g., $\sqrt{p}$ where p is the total number of features) and only considers these for the split. A new random subset is chosen for every single split in every tree.
- **Purpose:** This is the "Random" part that distinguishes Random Forest from standard Bagging of decision trees. Suppose there is one very strong predictor feature in the dataset. In standard Bagging, most of the trees would likely choose this strong feature as the root split. This would make all the trees in the forest structurally very similar and thus highly correlated. Their predictions would be very similar, and the benefit of

averaging would be diminished. By restricting the choice of features at each split, the algorithm forces other, weaker predictors to be chosen in some trees. This creates more diversity among the trees, further decorrelating them and leading to a greater reduction in variance when their results are aggregated.

In summary, bootstrap sampling creates tree diversity at the data level, while random feature subspaces create diversity at the model structure level. Both are crucial for reducing variance.

Question 24 (Numerical)

Problem: You have an ensemble of three binary classifiers (predicting Class A or Class B). For a new instance, their individual outputs are:

- Classifier 1: Predicts Class A, with $P(A) = 0.7, P(B) = 0.3$
- Classifier 2: Predicts Class A, with $P(A) = 0.8, P(B) = 0.2$
- Classifier 3: Predicts Class B, with $P(A) = 0.4, P(B) = 0.6$

What is the final prediction using a) Hard Voting (majority vote) and b) Soft Voting?

Solution

a) Hard Voting (Majority Vote):

- In hard voting, we only look at the final class prediction of each model.
- Classifier 1 predicts: **Class A**
- Classifier 2 predicts: **Class A**
- Classifier 3 predicts: **Class B**
- There are 2 votes for Class A and 1 vote for Class B.
- The majority class is Class A.
- The final prediction is **Class A**.

b) Soft Voting:

- In soft voting, we average the predicted probabilities from each classifier for each class and then choose the class with the highest average probability.
- **Average Probability for Class A:**
$$P_{avg}(A) = \frac{P_1(A) + P_2(A) + P_3(A)}{3} = \frac{0.7 + 0.8 + 0.4}{3} = \frac{1.9}{3} \approx 0.633$$
- **Average Probability for Class B:**
$$P_{avg}(B) = \frac{P_1(B) + P_2(B) + P_3(B)}{3} = \frac{0.3 + 0.2 + 0.6}{3} = \frac{1.1}{3} \approx 0.367$$
- We compare the average probabilities: $0.633 > 0.367$.
- The class with the highest average probability is Class A.
- The final prediction is **Class A**.

In this case, both methods agree. However, soft voting is often preferred as it accounts for the confidence of each classifier's prediction.

Question 25 (Scenario)

Problem: Describe the core intuition behind how Gradient Boosting works. How does it differ from AdaBoost in the way it corrects the mistakes of previous learners?

Solution

Core Intuition of Gradient Boosting: Gradient Boosting builds an ensemble of models sequentially, much like AdaBoost. However, its core idea is to treat the training process as an optimization problem where we want to minimize a loss function (e.g., Mean Squared Error for regression, Log Loss for classification).

The intuition is as follows:

1. Start with a very simple initial model, like a model that just predicts the mean (for regression) or the log-odds (for classification) of the target variable for all samples.
2. For each iteration:
   - Calculate the **residuals** (or "pseudo-residuals") for each sample. The residual is the difference between the true value and the current prediction of the ensemble: $residual_i = y_i - F_{m-1}(x_i)$. These residuals represent the "errors" that the current ensemble is making.
   - Train a new weak learner (e.g., a small decision tree) to **predict these residuals**, not the original target variable. The goal of this new tree is to learn the part of the error that the current ensemble can't explain.
   - Add this new learner to the ensemble, scaled by a learning rate. The ensemble's prediction is updated by adding a small step in the direction that the new tree suggests to correct the error. $F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$, where ν is the learning rate and $h_m(x)$ is the new tree.
3. Repeat this process for a specified number of iterations. Each new tree incrementally improves the overall ensemble by correcting the remaining errors.

The name "Gradient" comes from the fact that the residuals are the negative gradient of the loss function (for MSE, this is exact; for other loss functions, it's an approximation). So, each new tree is essentially fitting the negative gradient of the loss, which is the core idea of gradient descent optimization.

Difference from AdaBoost's Error Correction:

- **AdaBoost:** Corrects mistakes by **re-weighting the samples**. It identifies the samples that the previous learner got wrong and increases their weights. The next learner is then trained on this re-weighted dataset, forcing it to focus on the difficult, misclassified examples. The mechanism is changing the data distribution for the next learner.
- **Gradient Boosting:** Corrects mistakes by **fitting the residuals**. It does not change the weights of the samples. Instead, it defines the target for the next learner to be the error (residual) of the current ensemble. The mechanism is changing the learning objective for the next learner.

6 Unsupervised Learning

Question 26 (Numerical)

Problem: Given the 2D points A(2,10), B(2,5), C(8,4), D(5,8), E(7,5), F(6,4), G(1,2), H(4,9), perform two full iterations of the K-means clustering algorithm with K=2. The initial centroids are $C_1 = (2, 10)$ (point A) and $C_2 = (8, 4)$ (point C).

Solution

Initial Centroids: $C_1 = (2, 10)$, $C_2 = (8, 4)$. Points: A(2,10), B(2,5), C(8,4), D(5,8), E(7,5), F(6,4), G(1,2), H(4,9).

Iteration 1:

- **Assignment Step:** Assign each point to the nearest centroid (using squared Euclidean distance for simplicity).
  - A(2,10): $d(A, C_1)^2 = 0$, $d(A, C_2)^2 = (8-2)^2 + (4-10)^2 = 36+36 = 72$. Assign to **Cluster 1**.
  - B(2,5): $d(B, C_1)^2 = (2-2)^2 + (10-5)^2 = 25$, $d(B, C_2)^2 = (8-2)^2 + (4-5)^2 = 36+1 = 37$. Assign to **Cluster 1**.
  - C(8,4): $d(C, C_1)^2 = 72$, $d(C, C_2)^2 = 0$. Assign to **Cluster 2**.
  - D(5,8): $d(D, C_1)^2 = (2-5)^2+(10-8)^2 = 9+4 = 13$, $d(D, C_2)^2 = (8-5)^2+(4-8)^2 = 9+16 = 25$. Assign to **Cluster 1**.
  - E(7,5): $d(E, C_1)^2 = (2-7)^2+(10-5)^2 = 25+25 = 50$, $d(E, C_2)^2 = (8-7)^2+(4-5)^2 = 1+1 = 2$. Assign to **Cluster 2**.
  - F(6,4): $d(F, C_1)^2 = (2-6)^2+(10-4)^2 = 16+36 = 52$, $d(F, C_2)^2 = (8-6)^2+(4-4)^2 = 4+0 = 4$. Assign to **Cluster 2**.
  - G(1,2): $d(G, C_1)^2 = (2-1)^2+(10-2)^2 = 1+64 = 65$, $d(G, C_2)^2 = (8-1)^2+(4-2)^2 = 49+4 = 53$. Assign to **Cluster 2**.
  - H(4,9): $d(H, C_1)^2 = (2-4)^2+(10-9)^2 = 4+1 = 5$, $d(H, C_2)^2 = (8-4)^2+(4-9)^2 = 16+25 = 41$. Assign to **Cluster 1**.
- Cluster 1 = {A(2,10), B(2,5), D(5,8), H(4,9)}
- Cluster 2 = {C(8,4), E(7,5), F(6,4), G(1,2)}
- **Update Step:** Recalculate centroids as the mean of the points in each cluster.
  - New C_1: $x = (2+2+5+4)/4 = 13/4 = 3.25$. $y = (10+5+8+9)/4 = 32/4 = 8$. $C_1 = (3.25, 8)$.
  - New C_2: $x = (8+7+6+1)/4 = 22/4 = 5.5$. $y = (4+5+4+2)/4 = 15/4 = 3.75$. $C_2 = (5.5, 3.75)$.

Iteration 2:

- **Assignment Step:** Assign points to the new centroids $C_1 = (3.25, 8)$ and $C_2 = (5.5, 3.75)$.
  - A(2,10): $d(A, C_1)^2 \approx (1.25)^2 + (2)^2 = 5.56$. Assign to **Cluster 1**.
  - B(2,5): $d(B, C_1)^2 \approx (1.25)^2 + (-3)^2 = 10.56$. Assign to **Cluster 1**.
  - C(8,4): $d(C, C_2)^2 \approx (2.5)^2 + (0.25)^2 = 6.31$. Assign to **Cluster 2**.

  - D(5,8): $d(D, C_1)^2 \approx (1.75)^2 + (0)^2 = 3.06$. Assign to **Cluster 1**.
  - E(7,5): $d(E, C_2)^2 \approx (1.5)^2 + (1.25)^2 = 3.81$. Assign to **Cluster 2**.
  - F(6,4): $d(F, C_2)^2 \approx (0.5)^2 + (0.25)^2 = 0.31$. Assign to **Cluster 2**.
  - G(1,2): $d(G, C_2)^2 \approx (-4.5)^2 + (-1.75)^2 = 23.31$. Assign to **Cluster 2**.
  - H(4,9): $d(H, C_1)^2 \approx (0.75)^2 + (1)^2 = 1.56$. Assign to **Cluster 1**.

- The final cluster assignments are:
  - Cluster 1 = {A(2,10), B(2,5), D(5,8), H(4,9)}
  - Cluster 2 = {C(8,4), E(7,5), F(6,4), G(1,2)}

- The cluster assignments did not change from the end of Iteration 1 to the end of Iteration 2.

- **Update Step:** Since the assignments did not change, the centroids will not change either.
  - New $C_1'' = (3.25, 8)$
  - New $C_2'' = (5.5, 3.75)$

- The algorithm has converged after the second iteration.

Question 27 (Scenario)

Problem: You are using K-means for market segmentation. A major challenge is that the final clusters are highly dependent on the initial choice of centroids. Describe how the K-means++ initialization algorithm addresses this "sensitivity to initialization" problem.

Solution

The standard K-means algorithm often initializes by picking K random data points as the initial centroids. If these points are chosen poorly (e.g., all close together, or all in one dense region), the algorithm can converge to a suboptimal clustering (a local minimum of the within-cluster sum of squares).

K-means++ Initialization Algorithm: K-means++ is a "smart" initialization technique that aims to place the initial centroids far apart from each other, leading to better and more consistent final clusters. The process is as follows:

1. **Choose the First Centroid:** Select one data point uniformly at random from the dataset and set it as the first centroid, c_1.

2. **Choose Subsequent Centroids Iteratively:** For each subsequent centroid $k = 2, ..., K$:
   - **Calculate Distances:** For every data point x_i in the dataset, calculate its distance to the *nearest* centroid that has already been chosen. Let this distance be $D(x_i)$.
   - **Weighted Probabilistic Selection:** Select the next centroid, c_k, from the data points with a probability proportional to its squared distance, $D(x_i)^2$. A point that is far away from all existing centroids will have a large $D(x_i)^2$ and thus a high probability of being chosen as the next centroid.

3. **Proceed with Standard K-means:** Once all K centroids have been selected using this method, proceed with the standard K-means algorithm (assignment and update steps) until convergence.

How it Solves the Problem: By selecting new centroids with a probability proportional to their distance from existing centroids, K-means++ ensures that the initial centroids are spread out across the data. It avoids the pitfall of picking all initial centroids from within the same cluster. This better initialization makes K-means:

- **More likely to find the optimal (or a near-optimal) clustering.**
- **Converge faster** than with a random initialization, as it starts from a more sensible configuration.

It adds a small amount of computational overhead to the initialization step, but this is almost always outweighed by the benefits of better and faster convergence.

Question 28 (Conceptual/Applied)

Problem: Explain the difference between the hard clustering of K-means and the soft clustering of Gaussian Mixture Models (GMMs). Describe a real-world scenario where the soft clustering approach would be more beneficial.

Solution

Difference Between Hard and Soft Clustering:

- **K-means (Hard Clustering):**
  - In K-means, the assignment of data points to clusters is “hard” or absolute.
  - Each data point belongs to exactly **one** cluster – the one with the nearest centroid.
  - The output is a set of distinct, non-overlapping clusters. The decision boundary between clusters is sharp.
- **Gaussian Mixture Models (GMMs) (Soft Clustering):**
  - GMM assumes that the data points are generated from a mixture of several Gaussian distributions, where each Gaussian corresponds to a cluster.
  - It performs “soft” or probabilistic assignment. Each data point has a **probability of belonging to each cluster**.
  - For a given point, the output would be a set of probabilities that sum to 1, e.g., Point X has a 70% probability of belonging to Cluster 1, a 25% probability of belonging to Cluster 2, and a 5% probability of belonging to Cluster 3.

Scenario Where Soft Clustering is More Beneficial: Scenario: Customer Segmentation for Targeted Marketing Imagine a company analyzing its customer base to create segments for targeted marketing campaigns. The features might include ‘age‘, ‘income‘, ‘spending$_frequency‘, and‘average_transaction_value‘$.

Why K-means might be limiting: K-means would assign each customer to exactly one segment, e.g., “High-Value Spender,” “Budget-Conscious,” or “Infrequent Buyer.” However, a customer could realistically exhibit traits of multiple segments. For example, a customer might have a high income (like a “High-Value” customer) but be very price-sensitive and only buy during sales events (like a “Budget-Conscious” customer). A hard assignment would lose this nuance.

Benefit of GMM (Soft Clustering): A GMM would provide a probabilistic assignment. This same customer might be assigned:

- 60% probability of belonging to "Budget-Conscious"
- 35% probability of belonging to "High-Value Spender"
- 5% probability of belonging to "Infrequent Buyer"

This is far more insightful for marketing. The company can now understand that this customer is primarily motivated by discounts but has high purchasing power. They could be targeted with exclusive, high-value items that are on sale, a strategy that would be missed if they were only labeled as "Budget-Conscious." This allows for more nuanced, personalized, and effective marketing strategies.

Other good examples include document clustering (a document can be about both "politics" and "economics") or genetic analysis (an individual can have ancestry from multiple populations).

Question 29 (Numerical)

Problem: You are using the Expectation-Maximization (EM) algorithm for a GMM with two clusters (K=2) on a 1D dataset: $\{1, 2, 8, 9, 10\}$. The initial parameters for the two Gaussian components are:

- Cluster 1: $\mu_1 = 1.5, \sigma_1 = 1, \pi_1 = 0.5$ (mixing coefficient)
- Cluster 2: $\mu_2 = 8.5, \sigma_2 = 1, \pi_2 = 0.5$ (mixing coefficient)

Perform the E-step of the first iteration. That is, calculate the responsibilities (probabilities) of each data point belonging to each cluster.

Solution

The E-step involves calculating the responsibility that cluster k takes for data point i, denoted by $\gamma(z_{ik})$. The formula is:

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \sigma_j)}$$

Where $\mathcal{N}(x|\mu, \sigma)$ is the probability density of the Gaussian distribution:

$$\mathcal{N}(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Since the term $\frac{1}{\sqrt{2\pi}\sigma}$ will appear in both the numerator and denominator for all points (as $\sigma_1 = \sigma_2 = 1$), we can work with the unnormalized probabilities $P'_k(x_i) = \pi_k e^{-\frac{(x_i-\mu_k)^2}{2}}$ for simplicity.

Calculations for each point:

- **For x=1:**
  - $P'_1(1) = 0.5 \cdot e^{-\frac{(1-1.5)^2}{2}} = 0.5 \cdot e^{-0.125} \approx 0.5 \cdot 0.882 = 0.441$
  - $P'_2(1) = 0.5 \cdot e^{-\frac{(1-8.5)^2}{2}} = 0.5 \cdot e^{-28.125} \approx 0$

- Sum = 0.441. Responsibilities: $\gamma(z_{1,1}) = 0.441/0.441 \approx \mathbf{1.0}$, $\gamma(z_{1,2}) \approx \mathbf{0.0}$

- **For x=2:**
  - $P'_1(2) = 0.5 \cdot e^{-\frac{(2-1.5)^2}{2}} = 0.5 \cdot e^{-0.125} \approx 0.441$
  - $P'_2(2) = 0.5 \cdot e^{-\frac{(2-8.5)^2}{2}} = 0.5 \cdot e^{-21.125} \approx 0$
  - Sum = 0.441. Responsibilities: $\gamma(z_{2,1}) \approx \mathbf{1.0}$, $\gamma(z_{2,2}) \approx \mathbf{0.0}$
- **For x=8:**
  - $P'_1(8) = 0.5 \cdot e^{-\frac{(8-1.5)^2}{2}} = 0.5 \cdot e^{-21.125} \approx 0$
  - $P'_2(8) = 0.5 \cdot e^{-\frac{(8-8.5)^2}{2}} = 0.5 \cdot e^{-0.125} \approx 0.441$
  - Sum = 0.441. Responsibilities: $\gamma(z_{3,1}) \approx \mathbf{0.0}$, $\gamma(z_{3,2}) \approx \mathbf{1.0}$
- **For x=9:**
  - $P'_1(9) = 0.5 \cdot e^{-\frac{(9-1.5)^2}{2}} = 0.5 \cdot e^{-28.125} \approx 0$
  - $P'_2(9) = 0.5 \cdot e^{-\frac{(9-8.5)^2}{2}} = 0.5 \cdot e^{-0.125} \approx 0.441$
  - Sum = 0.441. Responsibilities: $\gamma(z_{4,1}) \approx \mathbf{0.0}$, $\gamma(z_{4,2}) \approx \mathbf{1.0}$
- **For x=10:**
  - $P'_1(10) = 0.5 \cdot e^{-\frac{(10-1.5)^2}{2}} = 0.5 \cdot e^{-36.125} \approx 0$
  - $P'_2(10) = 0.5 \cdot e^{-\frac{(10-8.5)^2}{2}} = 0.5 \cdot e^{-1.125} \approx 0.5 \cdot 0.325 = 0.1625$
  - Sum = 0.1625. Responsibilities: $\gamma(z_{5,1}) \approx \mathbf{0.0}$, $\gamma(z_{5,2}) \approx \mathbf{1.0}$

The subsequent M-step would use these responsibilities to update the parameters μ_k, σ_k, π_k.

Question 30 (Scenario)

Problem: You are a data scientist at a bank tasked with building an anomaly detection system for credit card transactions. How could you use a density-based clustering algorithm like DBSCAN to identify potentially fraudulent transactions?

Solution

The core idea is to use a clustering algorithm to define what "normal" transaction behavior looks like and then identify any transactions that do not conform to this normal behavior as anomalies (potential fraud). DBSCAN is particularly well-suited for this task.

Methodology using DBSCAN:

1. **Feature Engineering:** First, engineer relevant features from the raw transaction data. These could include:
   - 'Transaction Amount'
   - 'Time of Day' (converted to a cyclical feature)
   - 'Frequency' of transactions in the last hour/day for the user

- ‘Location‘ based features (e.g., distance from user’s home location)
- ‘Type‘ of merchant

These features should be appropriately scaled.

2. **Applying DBSCAN:** DBSCAN (Density-Based Spatial Clustering of Applications with Noise) works by grouping together points that are closely packed together, marking as outliers points that lie alone in low-density regions.

   - It defines clusters as dense regions of points in the feature space.
   - It has two key parameters: ‘eps‘ (epsilon), the maximum distance between two samples for one to be considered as in the neighborhood of the other, and ‘min$_s$*amples*‘, *thenumberofsamplesinaneighborhoodforapointtobeconsideredasacorepoir*

3. **Identifying Anomalies:**

   - Run the DBSCAN algorithm on the dataset of transaction features.
   - The algorithm will categorize each transaction into one of three types:
     (a) **Core point:** A point with at least ‘min$_s$*amples*‘*pointswithindistance*‘*eps*‘.*Thesearet*

     (a) **Border point:** A point that is reachable from a core point but is not a core point itself. These are the edges of a cluster.
     (b) **Noise point (Outlier):** A point that is neither a core point nor a border point.
   - The **noise points** identified by DBSCAN are our candidates for fraudulent transactions. These are the transactions that do not fit into any of the dense clusters of “normal” behavior. For example, a transaction with an unusually large amount, at an odd time of day, from a location far from the user’s typical area, would likely end up in a sparse region of the feature space and be flagged as noise by DBSCAN.

Why DBSCAN is a Good Choice for This:

- **No Need to Specify K:** Unlike K-means, you don’t need to specify the number of “normal behavior” clusters in advance. DBSCAN discovers them automatically based on density.

- **Arbitrary Cluster Shapes:** Fraudulent behavior might not form neat spherical clusters. DBSCAN can find arbitrarily shaped clusters, making it more flexible in defining what is “normal”.

- **Built-in Noise Detection:** Its primary strength is its explicit mechanism for identifying noise points, which is exactly what is needed for anomaly detection.

7 Model Evaluation and Comparison

Question 31 (Numerical)

Problem: A binary classifier for disease detection is tested on a set of 200 patients. There are 50 patients who actually have the disease (Positive class) and 150 who do not

(Negative class). The model's confusion matrix is shown below. Calculate the Accuracy, Precision, Recall, and F1-score.

Confusion Matrix

		Predicted Class	
		Positive	Negative
Actual Class	Positive	40 (TP)	10 (FN)
	Negative	20 (FP)	130 (TN)

Solution

From the confusion matrix, we have:

- True Positives (TP) = 40
- False Negatives (FN) = 10
- False Positives (FP) = 20
- True Negatives (TN) = 130
- Total Population = TP + FN + FP + TN = 40 + 10 + 20 + 130 = 200

1. Accuracy: (Overall, how often is the classifier correct?)

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{40 + 130}{200} = \frac{170}{200} = 0.85$$

2. Precision: (When it predicts positive, how often is it correct?)

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{40}{40 + 20} = \frac{40}{60} \approx 0.667$$

3. Recall (Sensitivity): (Of all the actual positives, how many did the classifier find?)

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{40}{40 + 10} = \frac{40}{50} = 0.80$$

4. F1-score: (The harmonic mean of Precision and Recall)

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.667 \cdot 0.80}{0.667 + 0.80} = \frac{1.0672}{1.467} \approx 0.727$$

Question 32 (Scenario)

Problem: You are building a model to predict fraudulent credit card transactions. The dataset is highly imbalanced: only 0.1% of transactions are fraudulent. Why is accuracy a poor metric for evaluating your model in this scenario? What metric(s) would be more appropriate?

Solution

Why Accuracy is a Poor Metric: In a highly imbalanced dataset, a model can achieve very high accuracy by simply predicting the majority class every time. This is called the "accuracy paradox."

Example:

- Dataset: 1,000,000 transactions.
- Fraudulent (Positive): 1,000 transactions (0.1%)
- Legitimate (Negative): 999,000 transactions (99.9%)

Consider a trivial "dummy" classifier that predicts **"legitimate"** for every single transaction.

- True Negatives (TN): 999,000 (correctly identifies all legitimate transactions)
- False Negatives (FN): 1,000 (misses every single fraudulent transaction)
- True Positives (TP): 0
- False Positives (FP): 0

The accuracy of this useless model would be:

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{0 + 999,000}{1,000,000} = 99.9\%$$

The model has 99.9% accuracy, but it is completely useless because its entire purpose – to detect fraud – has failed. It has zero predictive power for the positive class. This is why accuracy is highly misleading for imbalanced problems.

More Appropriate Metrics: Metrics that focus on the performance of the positive (minority) class are crucial here.

1. **Precision and Recall:**
   - **Recall (Sensitivity):** This is often the most important metric. It answers: "Of all the fraudulent transactions that occurred, what percentage did we catch?" We want to maximize this to catch as much fraud as possible.
   - **Precision:** This is also important. It answers: "When our model flags a transaction as fraud, what percentage of the time is it actually fraud?" We want this to be reasonably high to avoid bothering customers with too many false alarms (false positives).
2. **F1-score:** This is the harmonic mean of precision and recall, providing a single score that balances both concerns. It's useful when you care about both catching fraud and not having too many false alarms.
3. **Area Under the ROC Curve (AUC-ROC):** The ROC curve plots the True Positive Rate (Recall) against the False Positive Rate at various classification thresholds. The AUC represents the likelihood that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance. It gives a good aggregate measure of performance across all possible thresholds.

4. **Precision-Recall Curve (PRC) and Area Under the PRC (AUC-PR):** For highly imbalanced datasets, the PRC is often more informative than the ROC curve. It plots Precision vs. Recall. A good model will maintain high precision as recall increases. The area under this curve is a very good summary metric for imbalanced classification.

Question 33 (Conceptual/Applied)

Problem: Explain the Bias-Variance Trade-off. How does the complexity of a model (e.g., the depth of a decision tree or the value of 'k' in k-NN) relate to this trade-off?

Solution

The Bias-Variance Trade-off is a fundamental concept in supervised learning that describes the relationship between a model's complexity, its ability to fit the training data, and its ability to generalize to new, unseen data.

Definitions:

- **Bias:** Bias is the error from erroneous assumptions in the learning algorithm. High bias means the model is too simple and fails to capture the true underlying patterns in the data. This leads to **underfitting**. A high-bias model has high error on both the training and test sets.
- **Variance:** Variance is the error from sensitivity to small fluctuations in the training set. High variance means the model is too complex and captures the noise in the training data, not just the signal. This leads to **overfitting**. A high-variance model has very low error on the training set but high error on the test set.

The Trade-off: The total expected error of a model can be decomposed into three parts: Error = Bias^2 + Variance + Irreducible Error. The irreducible error is the noise inherent in the problem itself, which cannot be reduced by any model.

- As you increase the complexity of a model, its **bias decreases** (it can fit the training data better).
- However, as you increase complexity, its **variance increases** (it becomes more sensitive to the specific training data and less general).

The goal is to find a sweet spot in model complexity that minimizes the total error by balancing bias and variance.

Relation to Model Complexity:

- **Decision Tree Depth:**
  - **Low Complexity (Shallow Tree):** A decision tree with a very small depth (e.g., a decision stump) is a simple model. It will have **high bias** (it can't capture complex relationships) and **low variance** (it won't change much with different training data). It is likely to underfit.

- **High Complexity (Deep Tree):** A very deep, unpruned decision tree is a complex model. It will have **low bias** (it can perfectly memorize the training data) and **high variance** (it is highly sensitive to the training data and will not generalize well). It is likely to overfit.

- **Value of 'k' in k-NN:**
  - **High Complexity (Small 'k', e.g., k=1):** A 1-NN model is very complex. Its decision boundary is highly irregular and fits every training point perfectly. It has **low bias** but **high variance**. It is very sensitive to noise and outliers.
  - **Low Complexity (Large 'k', e.g., k=N):** If 'k' is equal to the total number of data points (N), the model is very simple. It will always predict the majority class of the entire dataset, regardless of the input. It has **high bias** but **zero variance**.

In both cases, tuning the hyperparameter (depth or 'k') is an exercise in navigating the bias-variance trade-off to find the best generalization performance.

Question 34 (Numerical)

Problem: A logistic regression model outputs the following probabilities for the positive class on 5 test instances. The true labels are [1, 0, 1, 0, 1].

True Label	Predicted Probability (for class 1)
1	0.9
0	0.7
1	0.6
0	0.3
1	0.2

Calculate the True Positive Rate (TPR) and False Positive Rate (FPR) for classification thresholds of $T > 0.8$ and $T > 0.5$.

Solution

First, let's establish our counts. Total Positives (P) = 3. Total Negatives (N) = 2. TPR = TP / P FPR = FP / N

Case 1: Threshold $T > 0.8$

- Predictions are made if probability ¿ 0.8.
- Predicted Positives: The instance with probability 0.9.
- Predicted Negatives: All other instances.
- Let's check the predictions:
  - Prob 0.9 (True=1): Predicted=1. This is a **True Positive (TP)**.
  - Prob 0.7 (True=0): Predicted=0. This is a **True Negative (TN)**.
  - Prob 0.6 (True=1): Predicted=0. This is a **False Negative (FN)**.

  - Prob 0.3 (True=0): Predicted=0. This is a **True Negative (TN)**.
  - Prob 0.2 (True=1): Predicted=0. This is a **False Negative (FN)**.
- Counts: TP=1, FP=0, TN=2, FN=2.
- TPR = TP / P = 1 / 3 ≈ 0.33
- FPR = FP / N = 0 / 2 = 0.0

Case 2: Threshold $T > 0.5$

- Predictions are made if probability ¿ 0.5.
- Predicted Positives: Instances with probabilities 0.9, 0.7, 0.6.
- Predicted Negatives: Instances with probabilities 0.3, 0.2.
- Let's check the predictions:
  - Prob 0.9 (True=1): Predicted=1. This is a **TP**.
  - Prob 0.7 (True=0): Predicted=1. This is a **False Positive (FP)**.
  - Prob 0.6 (True=1): Predicted=1. This is a **TP**.
  - Prob 0.3 (True=0): Predicted=0. This is a **TN**.
  - Prob 0.2 (True=1): Predicted=0. This is a **FN**.
- Counts: TP=2, FP=1, TN=1, FN=1.
- TPR = TP / P = 2 / 3 ≈ 0.67
- FPR = FP / N = 1 / 2 = 0.5

Summary of ROC points:

- For $T > 0.8$: (FPR=0.0, TPR=0.33)
- For $T > 0.5$: (FPR=0.5, TPR=0.67)

Question 35 (Scenario)

Problem: You need to compare the performance of three different algorithms (e.g., Logistic Regression, SVM, Random Forest) on a dataset of 5,000 samples. Why is a single train/test split (e.g., 80/20) insufficient for a robust comparison? Explain how K-fold cross-validation works and why it is superior.

Solution

Insufficiency of a Single Train/Test Split: Using a single, fixed train/test split to compare models is not robust due to its sensitivity to how the data is partitioned. This is known as **sampling bias**.

- **Luck of the Draw:** By random chance, the test set might contain particularly "easy" or "hard" examples. One algorithm might perform well on this specific test set simply due to luck, not because it is fundamentally a better model. If we were to re-shuffle the data and create a new split, the relative performance of the models could change completely.
- **Inefficient Use of Data:** With an 80/20 split, we are only ever evaluating the model on 20% of the data (1000 samples). The model's performance estimate is based on this single, small set. The remaining 80% of the data is used for training but never for evaluation.

A conclusion drawn from a single split (e.g., "Random Forest is the best model") might not be reliable and may not hold true in production.

How K-Fold Cross-Validation Works: K-fold cross-validation provides a more robust and reliable estimate of a model's performance by using all the data for both training and validation. The process is as follows:

1. **Partition:** The dataset is randomly shuffled and partitioned into K equal-sized folds (or subsets). A common choice for K is 5 or 10. Let's assume K=5.
2. **Iterate:** The process iterates K times. In each iteration:
   - A different fold is held out as the **validation set**.
   - The remaining K-1 folds are combined and used as the **training set**.
   - The model is trained on the training set and evaluated on the validation set. The performance score (e.g., accuracy, F1-score) for that fold is recorded.
3. **Aggregate:** After K iterations, we will have K performance scores. The final performance estimate for the model is the average of these K scores. The standard deviation of the scores can also be calculated to understand the stability of the model's performance.

Why it is Superior:

- **Reduces Sampling Bias:** Since every data point gets to be in a validation set exactly once, the performance estimate is less dependent on the specific way the data was split. The final averaged score is a more stable and reliable estimate of the model's true performance on unseen data.
- **More Efficient Use of Data:** Every data point is used for both training (in K-1 folds) and validation (in 1 fold). This is particularly important for smaller datasets where holding out a large test set is not feasible.
- **Provides a Measure of Variance:** By looking at the standard deviation of the K scores, we can get a sense of how sensitive the model's performance is to different training sets. A large standard deviation might indicate an unstable model.

To compare the three algorithms, you would perform K-fold cross-validation for each one and then compare their average performance scores. This provides a much more trustworthy basis for model selection.

8 Emerging Requirements in ML

Question 36 (Scenario)

Problem: A bank uses an ML model to approve or deny loan applications. The model was trained on historical data. It is discovered that the model denies loans to applicants from a specific minority group at a rate 30% higher than for the majority group, even for applicants with identical credit scores and income levels. What type of bias might be present, and what are two steps you could take to mitigate it?

Solution

Type of Bias Present: This scenario describes **algorithmic bias**, which likely stems from biases present in the historical training data. The specific types could be:

1. **Historical Bias / Societal Bias:** The historical loan data used to train the model likely reflects past discriminatory lending practices. Even if sensitive attributes like race were removed, other features (like zip code, which can be a proxy for race) might have allowed the model to learn and perpetuate these historical biases.
2. **Measurement Bias:** The features used might not mean the same thing for different groups. For example, 'years at current job' might be a less effective proxy for stability in a demographic group that experiences higher job mobility for systemic reasons.
3. **Representation Bias:** The minority group might be underrepresented in the training data, especially in the subset of approved loans, leading the model to be less accurate for this group.

Two Mitigation Steps:

1. **Pre-processing: Re-sampling or Re-weighting the Data**
   - **Technique:** This involves modifying the training data before feeding it to the model.
     - **Re-sampling:** We can oversample the underrepresented group (e.g., by creating synthetic samples using techniques like SMOTE - Synthetic Minority Over-sampling Technique) or undersample the overrepresented group. The goal is to create a more balanced dataset with respect to both the outcome (approved/denied) and the demographic groups.
     - **Re-weighting:** Assign higher weights to samples from the underrepresented group during model training. This forces the model to pay more attention to correctly classifying these instances, thereby reducing bias against them.

- **Goal:** To ensure the model learns from a dataset that does not marginalize the minority group, leading to fairer outcomes.

2. **Post-processing: Adjusting Classification Thresholds**
   - **Technique:** After the model is trained, we can apply different classification thresholds for different demographic groups to achieve a fairness goal. For example, the model outputs a probability score for loan approval. We might set a threshold of 0.6 for the majority group but a lower threshold of 0.5 for the minority group.
   - **Goal:** To enforce a specific fairness constraint, such as **Equal Opportunity** (ensuring the True Positive Rate is the same for both groups) or **Demographic Parity** (ensuring the overall approval rate is the same for both groups). This step directly adjusts the model's decisions to counteract the learned bias, without needing to retrain the model.

Question 37 (Scenario)

Problem: A hospital deploys a "black box" deep learning model to help doctors diagnose cancer from X-ray images. The model is 98% accurate, but doctors are hesitant to trust its predictions. Why is interpretability/explainability crucial in this high-stakes scenario? Name and briefly describe one technique that could be used to explain a specific prediction.

Solution

Why Interpretability is Crucial: In high-stakes environments like healthcare, a correct prediction is not enough. The reasoning behind the prediction is equally, if not more, important.

1. **Trust and Accountability:** Doctors are ultimately responsible for a patient's diagnosis and treatment. They cannot blindly trust a model's output without understanding its reasoning. If the model makes a mistake, the doctor needs to be able to identify it. Explainability builds trust and establishes a framework for accountability.
2. **Debugging and Identifying Flaws:** An explainable model allows developers and doctors to understand *why* it makes certain predictions. If the model is making correct predictions for the wrong reasons (e.g., focusing on a watermark or an artifact on the X-ray machine instead of the actual tumor), this can be identified and corrected. This is crucial for ensuring the model is robust and not relying on spurious correlations.
3. **Scientific Discovery and Learning:** By highlighting the features in an image that are most indicative of cancer, the model can potentially teach doctors to spot subtle patterns they might otherwise miss, leading to improved human diagnostic skills.
4. **Patient Communication and Recourse:** Doctors need to be able to explain the diagnosis to the patient. Citing a "black box" is unacceptable. If a patient is denied a treatment based on an algorithm's output, they have a right to understand the basis for that decision.

Example Technique: SHAP (SHapley Additive exPlanations)

- **What it is:** SHAP is a game theory-based approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory.
- **How it Works (Intuition):** For a specific prediction (e.g., classifying one X-ray as "cancerous"), SHAP treats the features (or super-pixels in the image) as "players" in a game, where the "payout" is the model's prediction. It calculates the contribution of each "player" to the final prediction.
- **Application to X-ray Image:** When applied to the X-ray image, a SHAP explanation would produce a **heatmap** overlaid on the original image.
  - Regions colored in **red** would indicate pixels or areas that pushed the model's prediction towards "cancerous."
  - Regions colored in **blue** would indicate pixels or areas that pushed the prediction towards "healthy."

  This heatmap would visually show the doctor exactly which parts of the X-ray the model considered most important in making its diagnosis, allowing the doctor to verify if the model is focusing on the correct pathological area.

Question 38 (Conceptual/Applied)

Problem: What is an adversarial attack on an ML model? Provide a specific example in the context of image classification and explain why robustness against such attacks is critical for deploying models in applications like self-driving cars.

Solution

What is an Adversarial Attack? An adversarial attack is a technique that involves making small, often imperceptible, perturbations to a model's input with the intent of causing the model to make an incorrect prediction. The goal is to exploit the model's learned patterns and sensitivities in a way that is not intuitive to humans. The modified input is called an "adversarial example."

Example in Image Classification:

- **Original Input:** A high-confidence image of a "Panda" is fed into a state-of-the-art image classifier, which correctly classifies it as a Panda with 99% confidence.
- **The Attack:** The attacker computes a specific "noise" pattern. This pattern is not random; it is calculated by looking at the gradient of the model's output with respect to the input pixels. This gradient tells the attacker how to change each pixel value slightly to maximally increase the model's prediction error.
- **Adversarial Example:** The attacker adds this carefully crafted, low-magnitude noise pattern to the original Panda image. To a human observer, the new image looks identical to the original Panda image.

- **Model's Output:** When this slightly perturbed image is fed to the same classifier, the model now misclassifies it as a "Gibbon" with high confidence.

This demonstrates how a change invisible to the human eye can completely fool a powerful machine learning model.

Why Robustness is Critical for Self-Driving Cars: Self-driving cars rely heavily on computer vision models to interpret the real world. A lack of robustness to adversarial attacks in this context could have catastrophic consequences.

Scenario: A self-driving car's perception system uses a model to recognize traffic signs.

- An attacker could create a small, inconspicuous sticker with an adversarial pattern and place it on a "Stop" sign.
- To a human driver, it still looks like a regular stop sign.
- However, the adversarial sticker is designed to make the car's model misclassify the "Stop" sign as a "Speed Limit: 60 mph" sign.
- The car would fail to stop at the intersection, potentially causing a serious accident.

This extends to other critical tasks:

- **Pedestrian Detection:** A pattern on a pedestrian's t-shirt could make them "invisible" to the car's detection system.
- **Lane Detection:** Small alterations to road markings could cause the car to swerve into another lane.

Therefore, ensuring that models are **robust** – meaning their performance does not degrade significantly in the face of such adversarial perturbations – is a critical safety and security requirement before they can be deployed in mission-critical systems like autonomous vehicles.

Question 39 (Scenario)

Problem: Your team has trained a state-of-the-art transformer-based model for natural language understanding. The model is highly accurate but has billions of parameters, making both training and real-time inference very slow and expensive. Discuss two different strategies you could employ to improve the model's scalability and efficiency for deployment.

Solution

Deploying massive models like large transformers presents significant challenges in terms of latency, computational cost, and memory footprint. Here are two key strategies to improve efficiency and scalability:

Strategy 1: Model Pruning and Quantization This strategy focuses on reducing the size and computational requirements of the existing trained model without significantly impacting its accuracy.

- **Pruning:**
  - **What it is:** Neural networks, especially large ones, are often highly over-parameterized. Pruning is the process of identifying and removing redundant or unimportant connections (weights) or even entire neurons/attention heads from the network.
  - **How it works:** After training, a sensitivity analysis is performed to find weights that have very small magnitudes (close to zero) and contribute little to the model's output. These weights are set to zero, effectively removing the connection. This results in a sparse weight matrix, which can be stored and processed much more efficiently. The model is often "fine-tuned" briefly after pruning to recover any lost accuracy.
- **Quantization:**
  - **What it is:** This is the process of reducing the numerical precision of the model's weights and activations. Models are typically trained using 32-bit floating-point numbers (FP32). Quantization converts these numbers to a lower-precision format, such as 16-bit floats (FP16) or even 8-bit integers (INT8).
  - **How it works:** By using fewer bits to represent each number, the model size is drastically reduced (e.g., INT8 quantization can reduce model size by 4x). Furthermore, computations with lower-precision numbers are significantly faster on modern hardware (like GPUs and TPUs) that have specialized cores for these operations. This leads to both lower memory usage and faster inference speed.

Strategy 2: Knowledge Distillation This strategy involves training a new, much smaller and faster model to mimic the behavior of the large, powerful one.

- **What it is:** Knowledge distillation is a process where a large, complex model (the "teacher") transfers its knowledge to a smaller, more efficient model (the "student").
- **How it works:**
  1. First, you have your large, pre-trained "teacher" model.
  2. Then, you design a much smaller "student" model architecture (e.g., a transformer with fewer layers and heads, or even a different architecture like an LSTM).
  3. You train the student model not just on the true labels of the training data, but on the *probabilistic outputs* (the "soft labels" or logits) produced by the teacher model.
  4. The loss function for the student is a combination of matching the true labels and matching the teacher's soft labels. By trying to mimic the teacher's probability distribution over all classes, the student learns the more nuanced patterns and "dark knowledge" that the teacher has captured, which is often more informative than just learning from the hard, true labels.
- **Outcome:** The result is a compact student model that can achieve performance very close to the large teacher model but with a fraction of the parameters, making it much faster and cheaper to deploy for real-time inference.

Question 40 (Conceptual/Applied)

Problem: Explain the difference between "fairness through unawareness" and more advanced fairness metrics like "demographic parity." Why is the former often an insufficient and naïve approach to achieving fairness?

Solution

Fairness Through Unawareness (Naïve Approach):

- **Definition:** This approach to fairness suggests that to prevent a model from being biased with respect to a sensitive attribute (e.g., race, gender), one should simply remove that attribute from the training data. The model is "unaware" of the sensitive feature, and the hope is that it therefore cannot be biased based on it.
- **Example:** To build a fair loan application model, a bank would remove the "race" column from their dataset before training.

Demographic Parity (Advanced Fairness Metric):

- **Definition:** This is a group fairness metric that requires the model's predictions to be independent of the sensitive attribute. It states that the probability of receiving a positive outcome should be the same for all demographic groups.
- **Mathematical Formulation:** A model satisfies demographic parity if $P(\hat{Y} = 1|A = a) = P(\hat{Y} = 1|A = b)$ for all groups a and b, where $\hat{Y} = 1$ is the positive outcome (e.g., loan approved) and A is the sensitive attribute.
- **Example:** If a loan model satisfies demographic parity, the percentage of male applicants who are approved will be the same as the percentage of female applicants who are approved.

Why "Fairness Through Unawareness" is Insufficient and Naïve: The "unawareness" approach fails because it ignores the problem of **redundant encodings** or **proxy variables**. Sensitive attributes are often highly correlated with other, non-sensitive attributes in the dataset.

1. **Proxy Variables:** Even if you remove the 'race' column, other features can act as strong proxies for race. For example, in many societies, 'zip code', 'income level', 'school district', and even 'first name' can be highly correlated with race due to systemic societal factors.
2. **Model's Behavior:** The machine learning model is designed to find patterns. It will learn the correlation between these proxy variables and the historical outcomes. As a result, it can effectively re-create the bias of the sensitive attribute without ever seeing it directly. It learns to discriminate based on zip code, which in turn leads to discrimination based on race.
3. **False Sense of Security:** This approach gives a false sense of having addressed the fairness issue. A company might believe their model is fair because they removed sensitive data, while in reality, it continues to perpetuate and even amplify existing societal biases.

In contrast, advanced metrics like demographic parity directly measure and constrain the *outcomes* of the model, regardless of how the model arrived at its decision. They acknowledge that bias can be learned implicitly and focus on ensuring the impact of the model is equitable across groups, which is a much more robust and meaningful way to address algorithmic fairness.