WA 2 Solutions

Om Khadka, U51801771
Collabs: N/A

Question 1: Which Points Belong in the Test Set? (25 points)

Suppose you have a learning algorithm that you want to test the performance of. The labels of your data are boolean, and you have n positive examples as well as n negative examples in your dataset. In order to test the performance of your model, you are going to compare the accuracy of your model against the accuracy of a *majority classifier* (i.e. a model that predicts the majority class contained within the training data). You will run a *leave-one-out* cross-validaton experiment (i.e. if you have x samples in your dataset, you will train your model on $x - 1$ samples and test on the remaining "left out" sample. You will repeat this process until each sample gets to be the "left out" test point, training the model from scratch each time). You expect the majority classifier to score about 50% every time, but it scores 0%. Why?

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples of piazza for guidance).

Assume the following variables:

- Let $P = \{x \in D \mid \text{label}(x) = +1\}$, $|P| = n$
- Let $N = \{x \in D \mid \text{label}(x) = -1\}$, $|N| = n$
- $D = P \cup N$, $P \cap N = \emptyset$

D is essentially the dataset with n positive examples $(+1)$ and n negative examples (-1) such that $|D| = 2n$.
The LOOCV process intuitively does the following:

- For each fold, remove 1 example e
- Train a majority classifier on the remaining $2n - 1$ examples
- Predict the class of e

The majority classifier predicts the class with more examples in the training set
Note: Since the total amount of classes in this validation is $2n - 1$, ties are impossible since the dataset is odd.

Case 1: $e \in P$ (The left-out example is positive)

Assertion	**Explanation**
Training set: $\|P \setminus \{e\}\| = n - 1$, $\|N\| = n$	1 positive class is now removed.
$n > n - 1 \rightarrow$ majority class = negatives.	Since there're more negatives in training.
Prediction for $e \rightarrow$ negative.	Majority classifier rule.
Actual label $\rightarrow$ positive.	This is incorrect.

Case 2: $e \in N$ (The left-out example is negative)

Assertion	**Explanation**
Training set: $\|P\| = n$, $\|N \setminus \{e\}\| = n - 1$	1 negative class is now removed.
$n > n - 1 \rightarrow$ majority class = positive.	Since there're more positives in training.
Prediction for $e \rightarrow$ positive.	Majority classifier rule.
Actual label $\rightarrow$ negative.	This is incorrect.

Looking at this result, it becomes clear that the reason why there's a 0% accuracy in the LOOCV is being the training set's majority class is always the opposite of the left-out example's true class. This is due to the initial perfect balance of the dataset (since we initially have both n amount of positive, negative classes).

Assertion	**Explanation**
Both cases, prediction is wrong.	
$\forall e \in D,$ majority classifier is wrong on e.	Holds for all examples.
$\therefore$, Accuracy $= 0/2n = 0\%$.	No correct predictions.

Question 2: Combining Multiple Models into an Ensemble (25 points)

Let us say that you have K separate classifiers, each trained on the same data. You combine them together into a single model by letting them vote: given a test point, predict that test point using all K classifiers, and then choose the most-frequently predicted class as your prediction. Such a model is called a *majority voting ensemble*. Suppose that each classifier has error ϵ, and that the errors made by each classifier are independent of the others'. Derive a formula for the error of the ensemble as a function of K and ϵ.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples of piazza for guidance).

Let there be K independent binary classifiers, each with error probablity ϵ. Let the true label for a given test point be fixed (so class 1 without loss of generality). We can define the random variable X as the number of classifiers that predict incorrectly. Then, $X \sim \text{Binomial}(K, \epsilon)$, so:

$$P(X = m) = \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m}, m = 0, 1, \ldots, K.$$

The predicted class will be the one receiving more than half of the votes (majority voting).

Assertion	**Explanation**
Let $X_i = \mathbb{1}\{\text{classifier } i \text{ is wrong}\}$	Define indicator variables.
$X_i \sim \text{Bernoulli}(\epsilon)$ for $i = 1, \ldots, K$	Each classifier has error ϵ.
$X = \sum_{i=1}^{K} X_i$	Total wrong classifiers.
$X \sim \text{Binomial}(K, \epsilon)$	Sum of i.d. Bernoulli variables.

Case 1: K is odd (Majority threshold is clear).

Assertion	Explanation
$K = 2t + 1$.	Odd K parameterization.
Majority means $\geq t + 1$ wrong.	More than half when K odd.
$$E(K,\epsilon) = \sum_{m=t+1}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m}$$.	Sum over all wrong-majority cases.
Substitute $t = \frac{K-1}{2}$.	Express in terms of K.
$$E(K,\epsilon) = \sum_{m=(K+1)/2}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m}$$.	Final form for odd K.

Case 2: K is even (Tie-breaking is needed).

Assertion	Explanation
$K = 2t$.	Even K parameterization.
Majority means $> t$ wrong.	Strict majority is required.
If $X = t$, then tie $\to$ random guess (error with prob. $1/2$).	Standard tie-breaking assumption.
Error if $X > t$ or ($X = t$ and guess is wrong).	There're 2 error-producing scenarios.
$$E(K,\epsilon) = \sum_{m=t+1}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m} + \frac{1}{2}\binom{K}{t} \epsilon^t (1-\epsilon)^t$$.	Sum over strict majority plus half of tie cases.
Substitute $t = K/2$.	Express in terms of K.
$$E(K,\epsilon) = \sum_{m=K/2+1}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m} + \frac{1}{2}\binom{K}{K/2} \epsilon^{K/2} (1-\epsilon)^{K/2}$$.	Final form for even K.

Thus, the error rate of the majority voting ensemble is the following:

$$E(K, \epsilon) = \begin{cases} \sum_{m=(K+1)/2}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m}, & K \text{ is odd,} \\ \\ \sum_{m=K/2+1}^{K} \binom{K}{m} \epsilon^m (1-\epsilon)^{K-m} + \frac{1}{2}\binom{K}{K/2} \epsilon^{K/2}(1-\epsilon)^{K/2} & K \text{ is even.} \end{cases}$$

This should account for independent classifier errors and the majority voting mechanism. Also note that $\epsilon < 0.5$, $E(K, \epsilon) \to 0$ as $K \to \infty$; Basically the ensemble improves with more classifiers.

Question 3: Linear Activations (25 points)

Suppose you had a neural network where every unit is equipped with a linear activation function, i.e. the output of a unit is some constant c times the weighted sum of its inputs:

1. Assume that the network has one hidden layer. For a given assignment of parameters, derive equations for the output of the units in the output layer as a function of the units in the input layer without any explicit mention of the output of the hidden layer. Show that there is a network with no hidden units that computes the same function.
2. Repeat part 1. but this time for a network with an arbitrary number of hidden layers. Conclude that for a neural network to learn any kind of nonlinear relationships, there must be at least a single unit with a nonlinear activation function.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples of piazza for guidance).

Consider a neural network where every unit has linear activation $g(z) = c \times z$ for some constant $c \neq 0$.
Let input be $\mathbf{x} \in R^d$, hidden layers have arbitary widths, and the output be $y \in R^m$.

For the case of One Hidden Layer

Assertion	**Explanation**
Hidden is $h = c(W_1\mathbf{x} + b_1)$.	Applying linear activation.
Output is $y = c(W_2 h + b_2)$.	Doing a second linear transformation.
$y = c(W_2[c(W_1\mathbf{x} + b_1)] + b_2)$.	Composition.
$y = c^2 W_2 W_1 \mathbf{x} + c^2 W_2 b_1 + c b_2$.	Expanidng terms.
Let $W' = c^2 W_2 W_1, b' = c^2 W_2 b_1 + c b_2$.	New parameters.
Then, $y = W'\mathbf{x} + b'$.	Single-layer linear model.
$\therefore$ The network with 1 hidden layers $\equiv$ to no hidden layer.	It is the same function class.

For the case of an Arbitary Number of Hidden Layers

Assertion	Explanation
Given a network of hidden layers, L, of depth k, $\exists W, b$ such that $y = Wx + b$.	This is a claim we're making, will be **proven via induction.**
Base case: $L = 1$ hidden layer	Previously proven as $y = W'\mathbf{x} + b'$ from the previous question (single linear transform)
Inductive step: Assume true for $L = k$ layers.	The inductive hypothesis.
For $L = k+1$: First k layers are $= h = W_k\mathbf{x} + b_k$.	By hypothesis.
Add layer $k+1$ such that $y = c(W_{k+1}h + b_{k+1})$.	Linear activation.
$y = c(W_{k+1}(W_k\mathbf{x} + b_k) + b_{k+1})$.	Composition.
$y = cW_{k+1}W_k\mathbf{x} + cW_{k+1}b_k + cb_{k+1}$.	Expanding terms.
This is essentially $y = W'\mathbf{x} + b'$.	And is also still linear.
$\therefore$ Is true for all $L \geq 0$.	By induction.

We can also solve for nonlinearity in order for everything to work out in the end too:

Assertion	Explanation
Any composition of linear maps is linear.	Just a fact of this mathematics.
$\therefore$ Linear networks can only learn from linear $f(\mathbf{x})$.	Showing limited expressivity.
To learn a nonlinear relationship, it'll need ≥ 1 nonlinear unity.	Showing the necessary condition.
There exists other functions for this case (sigmoid, relu).	The specifics of these function aren't important right now, the main point is that there exist ways to enable nonlinear maps.

Therefore, this proves how a neural network with only linear activation functions, regardless of depth, is equivalent to a single-layer linear model. This THUS means that AT LEAST 1 nonlinear activation function is necessary for learning nonlinear relationships.

Question 4: Datasets with Weights (25 points)

Consider a dataset in which each data point $\left(x^{(i)}, y_{gt}^{(i)}\right)$ is associated with some weight $r^{(i)} > 0$. If we want to use a mean squared error for our loss function (like we want to do in temporal difference learning), our objective now becomes:

$$L(\vec{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} r^{(i)}\left(y_{gt}^{(i)} - f_{\vec{\theta}}(x^{(i)})\right)^2$$

For now, lets simplify $f_{\vec{\theta}}$ to be a linear model (which in an earlier homework you showed that any completely-linear neural network could be reduced to this) $f_{\vec{\theta}}(x) = \vec{\theta}^T\phi(x)$. Plugging this in:

$$L(\vec{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} r^{(i)}\left(y_{gt}^{(i)} - \vec{\theta}^T\phi(x^{(i)})\right)^2$$

Derive an expression for the optimum $\vec{\theta}^*$ that minimizes this loss function.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples of piazza for guidance).

This is essentially a weighted least squares problem. To solve this we have to do the following:

- Write the loss in matrix/vector notation
- Take the gradient with respect to $\overrightarrow{\theta}$
- Set the gradient to 0 and solve for $\overrightarrow{\theta}^*$
- Make sure its a minimum

Before solving this though, I will clarify on some variables that'll be used for the solution:

- $$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N} r^{(i)}(y_{gt}^{(i)} - \theta^T\phi(x^{(i)}))^2$$
  the weighted mean squared error loss function.
- $$\Phi \in \mathbb{R}^{N\times d} \text{ with } \Phi_i = \phi(x^{(i)})^T$$
  the feature matrix
- $$y = [y_{gt}^{(i)}, \dots, y_{gt}^{(N)}]^T$$
  the target vector
- $$R = \text{diag}(r^{(1)}, \dots, r^{(N)})$$
  the weight matrix

Assertion	**Explanation**
$L(\theta) = \frac{1}{2N}(y - \Phi\theta)^T R(y - \Phi\theta)$.	This is the matrix form of the weighted MSE.
$L(\theta) = \frac{1}{2N}[y^T Ry - 2y^T R\Phi\theta + \theta^T \Phi^T R\Phi\theta]$.	Expanding via quadratic expansion.
$\nabla_\theta L(\theta) = \frac{1}{2n}[-2\Phi^T Ry + 2\Phi^T R\Phi\theta]$.	Performing gradient calculation.
Set $\nabla_\theta L(\theta) = 0$ to find critical points.	This is the 1st-order condition, it is necessary for the optimum.
$-\Phi^T Ry + \Phi^T R\Phi\theta = 0$	Multiply both sides by N to eliminate $\frac{1}{N}$.
$\Phi^T R\Phi\theta = \Phi^T Ry$.	Normal equations.
Since $R \succ 0$ and Φ has full column rank,	Positive weights and linearly independent features.
$\Phi^T R\Phi$ is invertible.	This is the full-rank condition; Guarantees a unique solution.
$\theta^* = (\Phi^T R\Phi)^{-1}\Phi^T Ry$.	This is the optimal solution.
Hessian: $\nabla^2_\theta L(\theta) = \frac{1}{N}\Phi^T R\Phi$	Second derivative test.
Since R is diagonal with $r^{(i)} > 0$, and Φ full rank,	All weights positive, design matrix full rank.
$\Phi^T R\Phi \succ 0$ (positive definite)	Quadratic form is positive definite.
$\therefore$ Critical point is a global minimum.	Convex optimization guarantee.

In conclusion, the optimal parameter for the weighted linear regression is the following:

$$\theta^* = (\Phi^T R\Phi)^{-1}\Phi^T Ry$$

where R is the diagonal weight matrix, Φ is the feature matrix, and y is the target vector.

Extra Credit: Decision Trees with Missing Values (25 points)

Standard decision trees are not able to handle examples where one (or more) of the attributes contain an unknown value. Any unknown or unfilled entry in an example is called a "missing value", and our decision tree will fail if presented with such an example:

1. First, we need a way to classify any example that contains missing value(s). Suppose an example $\vec{x}$ which has a missing value for attribute A, and that the decision tree test for A at a node that $\vec{x}$ reaches. One way to handle this missing value is to pretend that $\vec{x}$ has *all possible* values of A, and to weight each value according to the frequency that the values appear in the dataset the node was constructed from. This classification algorithm should follow all branches at any node for which a value is missing and should multiply weights along each path. Design a classification algorithm for decision trees that has this behavior.
2. Now, modify the information gain calculation so that when constructing a node from dataset D, the examples with missing values for any of the remaining attributes are given "as-if" values according to the frequencies of those values in D.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples of piazza for guidance).

This problem can be solved with the probabilistic split method. Since the desicion tree has missing values, we treat missing values as having probablity distribution over possible values, then use fractional examples instead of just throwing away that data.

Let a decision tree be constructed from dataset D. Each node will store the following:

- The attribute A to test
- The frequency distribution $P_A(v)$ of each value $v \in \text{Values}(A)$ from the training data at that node.
- For leaves specifically, the class distribution.

The classification algorithm can be described as follows:

Assertion	**Explanation**
Input is example $\vec{x}$, tree T.	The example may have missing values.
Ouput is the predicted class $\hat{y}$.	Weighted combination.
Init. with leaves = $\emptyset$.	Store as (leaf, weight) pairs.
traverse($node, w$).	Do a recusive traversal with weight w.
If the node is a leaf, then leaves.add(($node, w$)).	We've reached a leaf with weight w.
Else, let A = node's attribute.	Defining a decision node.
If $\vec{x}[A]$ is not missing, $v = \vec{x}[A]$, child = node.child[v], then traverse(child, w).	On a normal case, we'll following the corresponding branches of same weight.
Else, if missing $\vec{x}[A]$. For each $v \in$ Values(A): $w_v = w \times P_A(v)$ child = node.child[v] traverse(child, w_v)	If the value is missing, then for all possible values, we get the weight by value frequency, go the corresponding branch, then do a recursive call.
Start of with traverse(root, 1.0)	The init. call.
Output (prediction): $\hat{y} = \arg\max_c \sum_{(\text{leaf},w)\in\text{leaves}} w \times P_{\text{leaf}}(c)$.	The weighted vote.

For the modified information gain calculations, we can suppose at node contruction with dataset D, let $D_{\text{complete}} \subset D$ be examples with known attribute A.

Assertion	Explanation
Let $D_{\text{miss}} = D \backslash D_{\text{complete}}$.	All examples missing A.
Compute $(P_A(v) = \frac{\lvert D^v_{\text{complete}} \rvert}{D_{\text{complete}}})$.	All value frequencies from the complete data.
For each $x \in D_{\text{miss}}$ For each $v \in \text{Values}(A)$.	To handle missing examples, and then to split up into all values.
Create a fractional example (x, y) with weight $w = P_A(v)$.	Weight by frequency.
Let $D' = D_{\text{complete}} \cup \{\text{fractional } D_{\text{miss}}\}$.	Make an augmented dataset.
Original entropy is $H(D) = -\sum_c P(c) \log P(c)$.	Using weighted counts.
Weight split is $H(D\|\|A) = \sum_v P(v) H(D_v)$.	Conditional entropy.
$(P(v) = \frac{\sum_{x \in D'} \mathbb{I}[x.A=v] \cdot w_x}{D'})$.	The weighted probability.
$H(D_v) = -\sum_c P(c\|\|D_v) \log P(c\|\|D_v)$.	Entropy of split.
$P(c\|\|D_v) = \frac{\sum_{x \in D_v} \mathbb{I}[x.y=c] \times w_x}{\sum_{x \in D_v} w_x}$.	The weighted class distribution.
Info gain is $IG(A) = H(D) - H(D\|\|A)$.	The standard formula with weights.

For a tree construction with missing values, the following algorithm could work:

Assertion	**Explanation**
Suppose a procedure, build_tree(D, attribute).	Defining recursive construction.
IF stopping critierion is the following Return leaf with $P(c)$.	The base case.
FOR EACH attribute $A \in$ attributes.	This is the candidate splits.
$A^* = \arg\max_A IG(A)$.	The best attribute.
Create node testing A^*.	
Store $P_{A^*}(v)$ from D.	For the classification phase.
FOR EACH $v \in$ Values(A^*).	Build the children.
$D_v = \{x \in D : x.A^* = v\} \cup \{$fractional $x \in D_{\text{miss}}$ with weight $P_{A^*}(v)\}$.	The weighted split.
child = build_tree(D_v, attributes $\{A^*\}$).	The recursive call.
Return node	

In total, this series of algorithm should handle missing values consistently in both classification and training. This is because:

- The probablistic splits are based off of value frequencies during classififcation.
- The algorithm employs fractional examples based on value distributions during training.
- And weight propagation is maintained throughout both phases

Thus leading to no thrown away data.

Some questions for my answer:

- How I've written out my algorithm kinda feels messy since it's within my 2-column format. Please do let me know if you want me to actually write out the algorithm and then explain it, or if my explanation was sufficent.
- Should I also prove how the weight propagation actually preserves probability (like such that the sum of weights = 1)
- Is my IG function right? I dunno since I had to also consider the impurity reduction.
- Did my explantion sufficently show how this is better than just single imputation?
- If there're any issues with the tree construction part of my answer, please let me know.