Machine Learning Methods

Topic 2: Decision Stumps, VC Dimension & Decision Trees

Yonghao Lee

January 2026

Contents

1 K-Nearest Neighbors (KNN)

KNN is a non-parametric, lazy learning algorithm used for both classification and regression. "Lazy" means it does not learn a model during training; it simply memorizes the data and performs computation only during inference.

1.1 How KNN Works

1. **Choose K:** Decide on a hyperparameter K (number of neighbors).
2. **Calculate Distance:** Given a query point x_q, compute the distance (usually Euclidean) to all training points x_i:
$$d(x_q, x_i) = \sqrt{\sum_{j=1}^{d} (x_{q,j} - x_{i,j})^2}$$
3. **Find Neighbors:** Identify the K points with the smallest distance.
4. **Vote (Classification):** Assign the label that appears most frequently among the neighbors.
5. **Average (Regression):** Assign the mean value of the neighbors' targets.

Study Note

The Trade-off of K:

- **Small K (e.g., $K = 1$):** Low Bias, High Variance. The model captures local noise and outliers. The decision boundary is jagged.
- **Large K (e.g., $K = N$):** High Bias, Low Variance. The model ignores local structure and predicts the global majority class. The boundary is overly smooth.

1.2 The Curse of Dimensionality

KNN works well in low dimensions but struggles in high dimensions (e.g., images with thousands of pixels).

- **Space becomes sparse:** As dimensions increase, the volume of the space grows exponentially. Data points become incredibly far apart.
- **Distance loses meaning:** In very high dimensions, the distance to the *nearest* neighbor approaches the distance to the *farthest* neighbor. "Nearest" essentially becomes random.

2 Clustering (K-Means)

2.1 Motivation

KNN is Supervised (requires labels). **Clustering** is Unsupervised. We use it to find structure in unlabeled data, such as customer segmentation or image compression.

2.2 Objective Function

We want to minimize the **Inertia** (Within-Cluster Sum of Squares).

Definition 2.1: K-Means Objective

$$J = \sum_{k=1}^{K} \sum_{x_i \in S_k} ||x_i - \mu_k||^2$$

where μ_k is the centroid of cluster k.

2.3 Lloyd's Algorithm

Minimizing J is NP-Hard, so we use an iterative approximation:

1. **Initialization:** Pick K random points as starting centroids.
2. **Assignment:** Assign every point x_i to the closest centroid.
3. **Update:** Move each centroid to the mean of the points assigned to it.
4. **Repeat:** Steps 2-3 until centroids stop moving (convergence).

3 Parametric Methods: Decision Stumps

3.1 The Shift to Parametric Models

In KNN, we kept all the data. In Parametric methods, we summarize data into a fixed set of parameters $\boldsymbol{\theta}$.

$$\text{Data} \xrightarrow{\text{Learning}} \boldsymbol{\theta} \xrightarrow{\text{Inference}} \text{Prediction}$$

3.2 Decision Stump Definition

A decision stump is the simplest possible parametric classifier: a tree with a **single split**. It is defined by:

- A specific feature j.
- A threshold θ.
- A direction (which side is class 1).

3.3 Training a Decision Stump

Since the threshold θ is continuous, it seems we have infinite choices. However, we only need to test thresholds **between** data points.

Example 3.1: Finding the Optimal Threshold

Dataset: 4 Healthy (H), 4 Sick (S) patients, sorted by temperature.

Temp (x):	36.1	36.5	36.9	37.2	37.8	38.1	38.5	39.0
Label (y):	H	H	H	H	S	S	S	S

Step 1: Identify Candidate Splits. Possible splits are midpoints between sorted values: $36.3, 36.7, \ldots, 37.5, \ldots$

Step 2: Evaluate Errors.

- Split at 37.0 (between 36.9 and 37.2):
  - Left (< 37.0): {H, H, H} $\rightarrow$ Predict H (0 errors).
  - Right (≥ 37.0): {H, S, S, S, S} $\rightarrow$ Predict S (1 error: the H at 37.2).
  - Total Accuracy: 7/8.
- Split at 37.5 (between 37.2 and 37.8):
  - Left (< 37.5): {H, H, H, H} $\rightarrow$ Predict H (0 errors).
  - Right (≥ 37.5): {S, S, S, S} $\rightarrow$ Predict S (0 errors).
  - Total Accuracy: **8/8**.

Step 3: Maximum Margin. Any threshold in $(37.2, 37.8)$ gives 0 error. We pick the midpoint to maximize robustness:

$$\theta^* = \frac{37.2 + 37.8}{2} = 37.5$$

4 VC Dimension & Shattering

4.1 Why do we need this?

Decision stumps have an infinite number of possible thresholds. Does this mean they are infinitely complex and will overfit? **No.** VC Dimension gives us a rigorous way to measure the "capacity" or "richness" of a model class, independent of the number of parameters.

4.2 Concept: Shattering

A model class $\mathcal{H}$ **shatters** a dataset D if $\mathcal{H}$ is capable of assigning **any possible labeling** to the points in D. If $|D| = d$, there are 2^d possible label combinations (00, 01, 10, 11, etc.). The model must be able to reproduce **all** of them.

4.3 VC Dimension Definition

The VC Dimension is the size of the **largest** set of points that can be shattered.

$$\text{VCdim} = d \iff \exists \text{ a set of size } d \text{ that is shattered, AND NO set of size } d+1 \text{ is shattered.}$$

Theorem 4.1: VC Dimension of Decision Stumps

For 1D Decision Stumps, VCdim = 1.

4.3.1 Proof Breakdown

Step 1: Can we shatter 1 point? Yes.

- Point x_1.
- If we want $y = 0$, put threshold $\theta > x_1$.
- If we want $y = 1$, put threshold $\theta \leq x_1$.
- We can produce all $2^1 = 2$ labelings.

Step 2: Can we shatter 2 points? No. Let's look at the "XOR" problem with 2 points, $x_1 < x_2$.

y_1	y_2	Required Rule	Possible with Stump?
0	0	$\theta > x_2$	Yes
1	1	$\theta \leq x_1$	Yes
0	1	$x_1 < \theta \leq x_2$	Yes
1	**0**	**Impossible**	**NO**

Why is (1, 0) impossible? A decision stump is a single cut. Everything to the right is 1 (or 0). If $x_1 = 1$ and $x_2 = 0$, we need the function to go High $\rightarrow$ Low. But standard decision stumps are defined as step functions that go Low $\rightarrow$ High (or vice versa). Even if we allow direction flipping, we cannot assign "1" to the left, "0" to the right, and then "1" again later. A single cut cannot isolate the middle.

Since we fail for $d = 2$, the VC dimension is 1.

5 Inductive Bias

Definition 5.1: Inductive Bias

The set of assumptions a learner makes to predict outputs for unseen inputs. Without inductive bias, a learner cannot generalize—it can only memorize.

Inductive Bias of Decision Trees:

1. **Axis-Aligned Boundaries:** Splits are always perpendicular to axes (e.g., $x_1 > 5$).
   - *Consequence:* Trees struggle to learn diagonal boundaries (like $y > x$). To approximate a diagonal line, a tree must create a "staircase" pattern, which requires many splits.
2. **Prefer Short Trees:** We assume the true concept is simple. We stop splitting when "purity" is high to avoid overfitting.
3. **Hierarchical Structure:** We assume that features interact in a specific order (e.g., Feature A is the most important, then Feature B).

6 Decision Trees: Splitting Criteria

6.1 The "Surprise" of Uncertainty

How do we mathematically choose the best split? We need a metric for "messiness" or "impurity."

Imagine I tell you an event occurred with probability p.

- If $p = 1$ (Certainty), you are **not surprised**. Surprise $= 0$.
- If $p = 0.01$ (Rare), you are **very surprised**.

Mathematically, this is modeled as Surprise $= -\log_2(p)$.

6.2 Entropy

Entropy is simply the **average surprise** of a distribution.

Definition 6.1: Entropy

$$H(S) = -\sum_c p_c \log_2(p_c)$$

- **Pure Node ($p = [1, 0]$):** Entropy $= -1 \log 1 - 0 \log 0 = 0$. (Minimum uncertainty).
- **Impure Node ($p = [0.5, 0.5]$):** Entropy $= -0.5(-1) - 0.5(-1) = 1$. (Maximum uncertainty).

6.3 Connection to Maximum Likelihood

Why do we minimize Entropy? It's not just arbitrary. Minimizing Entropy is mathematically equivalent to maximizing the **Log-Likelihood** of the data.

$$\text{Maximize Likelihood} \iff \text{Minimize Negative Log-Likelihood} \iff \text{Minimize Entropy}$$

Example 6.1: Likelihood Example

Leaf Node with 10 samples: 8 Positive, 2 Negative. Model predicts $P(+) = 0.8, P(-) = 0.2$. Likelihood of seeing this data:

$$L = (0.8)^8 \times (0.2)^2$$

Log-Likelihood:

$$\log(L) = 8\log(0.8) + 2\log(0.2)$$

Notice that this formula looks exactly like Entropy (weighted sum of logs)! By maximizing this sum, we find the probabilities that best fit the data.