CS 3780 Notes

Jinzhou Wu

September 14, 2025

Contents

1 Supervised Learning and KNN

Date: Aug 28, 2025

1.1 Supervised Learning

In supervised learning, we are given a dataset:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where $x_i \in \mathcal{X}$ is a feature vector and $y_i \in \mathcal{Y}$ is its label. The goal is to learn a hypothesis function:

$$h : \mathcal{X} \to \mathcal{Y}$$

that approximates the unknown target function $f : \mathcal{X} \to \mathcal{Y}$.

Key Concepts

- **Instance:** A single feature vector $\mathbf{x} \in \mathcal{X}$.
- **Instance Space** $\mathcal{X}$: The set of all possible feature vectors.
- **Label** y: The output to be predicted.
- **Label Space** $\mathcal{Y}$: The set of all possible labels.

Types of Supervised Learning

- **Binary Classification:** $\mathcal{Y} = \{-1, +1\}$
- **Multi-class Classification:** $\mathcal{Y} = \{1, 2, \ldots, k\}$
- **Regression:** $\mathcal{Y} \subseteq \mathbb{R}$
- **Structured Output:** $\mathcal{Y} =$ Object (e.g., protein structures)

1.2 K-Nearest Neighbors (KNN)

KNN is a non-parametric learning algorithm that predicts the label of a new instance $\mathbf{x}'$ using the labels of the k closest points in the training dataset according to a similarity (or distance) measure.

Algorithm

KNN Algorithm:

1. **Input:** Training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, similarity function K, number of neighbors k.
2. For a test point $\mathbf{x}'$, compute $K(\mathbf{x}_i, \mathbf{x}')$ for all i.
3. Find the k nearest neighbors:
$$\text{knn}(\mathbf{x}') = \{i \mid \mathbf{x}_i \text{ among } k \text{ closest to } \mathbf{x}'\}.$$
4. Predict the label:
$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \text{knn}(\mathbf{x}')} \mathbf{1}(y_i = y).$$

1.3 Weighted KNN

Weighted KNN assigns higher weights to closer neighbors using the similarity function K.

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \text{knn}(\mathbf{x}')} K(\mathbf{x}_i, \mathbf{x}') \cdot \mathbf{1}(y_i = y)$$

Weighted KNN for Regression

For regression problems, the prediction is a weighted average:

$$h(\mathbf{x}') = \frac{\sum_{i \in \text{knn}(\mathbf{x}')} y_i \cdot K(\mathbf{x}_i, \mathbf{x}')}{\sum_{i \in \text{knn}(\mathbf{x}')} K(\mathbf{x}_i, \mathbf{x}')}$$

1.4 Similarity Measures

Different similarity or distance measures can be used depending on the problem:

- **Gaussian Kernel:**
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right)$$
- **Laplace Kernel:**
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|\right)$$
- **Cosine Similarity:**
$$K(\mathbf{x}, \mathbf{x}') = \cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}$$

1.5 Types of Attributes

- **Categorical:** e.g., EyeColor $\in$ {brown, blue, green}
- **Boolean:** e.g., Alive $\in$ {True, False}
- **Numeric:** e.g., Age, Height
- **Structured:** e.g., sentences, protein sequences

1.6 Properties of KNN

- Simple, intuitive, and non-parametric.
- Requires a meaningful similarity measure.
- Memory-intensive: stores the entire training dataset.
- Computationally expensive for large datasets.
- Suffers from the **curse of dimensionality**.
- KNN is more like a *memorization* method rather than true generalization.

2 Inductive Learning and Decision Trees

Date: Sep 2, 2025

2.1 Inductive Learning

Inductive learning is the process of learning a general rule or hypothesis from specific observed examples. Given a training dataset:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

we aim to find a hypothesis h such that $h(x) \approx y$ for unseen examples.

Key Ideas

- Training data provides labeled examples of inputs and outputs.
- The goal is to infer a hypothesis h consistent with as many training examples as possible.
- If multiple hypotheses are consistent, we aim to choose one that generalizes well.

2.2 Version Space

Definition (Version Space)**.** The **version space** is the set of all hypotheses in the hypothesis space H that are consistent with the observed training examples:

$$VS = \{h \in H \mid \forall (x_i, y_i) \in S,\ h(x_i) = y_i\}.$$

Using Version Space for Learning

- Start with the set of all hypotheses.
- Remove any hypothesis inconsistent with any training example.
- The remaining hypotheses form the version space.

2.3 List-Then-Eliminate Algorithm

Algorithm

Algorithm: *List-Then-Eliminate*

1. Initialize $VS \leftarrow H$ (all hypotheses).
2. For each training example (x_i, y_i):
   - Remove all $h \in VS$ such that $h(x_i) \neq y_i$.
3. Return the remaining hypotheses in VS.

Takeaway: Tracking the entire version space can be expensive in both time and memory. Instead, we often directly construct a consistent hypothesis, e.g., using decision trees.

2.4 Decision Trees

Definition (Decision Tree)**.** A decision tree represents a function $h : \mathcal{X} \to \mathcal{Y}$ as a tree structure:

- **Internal nodes:** Test a single feature (e.g., "Color").
- **Branches:** Possible values of that feature (e.g., "Red" or "Green").
- **Leaf nodes:** Assign a label based on the path from root to leaf.

Using a Decision Tree

To classify a new example:

1. Start at the root.
2. At each internal node, test the corresponding feature.
3. Follow the branch matching the feature value.
4. Stop at a leaf and return its label.

2.5 Top-Down Induction of Decision Trees (IDT)

Algorithm

Algorithm: *IDT(S, Features)*

1. If all examples in S have the same label, return a leaf node with that label.
2. If no features remain, return a leaf node with the majority label in S.
3. Otherwise:
   - Choose the best feature A to split on.
   - Partition S into subsets $\{S_v\}$ by the values of A.
   - For each value v of A:
     - Recursively call *IDT(S_v, Features* $\setminus\{A\}$*)*.

2.6 Choosing the Best Split

Error-Based Split Criterion: To choose the best attribute A to split on:

$$\mathrm{Err}(S) = \min(\#\text{positive},\ \#\text{negative})$$

$$\mathrm{Err}(S \mid A) = \sum_v \mathrm{Err}(S_v)$$

Select A that maximizes:

$$\Delta\mathrm{Err} = \mathrm{Err}(S) - \mathrm{Err}(S \mid A).$$

2.7 Properties of Decision Trees

- Easy to interpret and visualize.
- Can represent complex decision boundaries.
- Handles both categorical and numerical features.
- Prone to overfitting if the tree grows too deep.
- Typically combined with pruning techniques for better generalization.
- Basis for more advanced models like Random Forests and Gradient Boosted Trees.

3 Prediction and Overfitting

Date: Sep 4, 2025

Learning as Prediction

Goal: Given a training dataset $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ drawn **i.i.d.** from an unknown distribution $\mathcal{D}$, learn a hypothesis h such that $h(x) \approx y$ for unseen data.

World as a Distribution

Features (e.g., Farm, Color, Size, Firmness) and labels (e.g., Tasty) are random variables. The underlying distribution $\mathcal{D}$ defines:

- **Joint Distribution:** $P(X, Y)$
- **Marginal Distribution:** $P(X)$
-

3.1 D

istribution: $P(Y|X)$

Sample vs. Prediction Error

Definition (Sample (Empirical) Error)**.**

$$\hat{L}_S(h) = \frac{1}{|S|} \sum_{(x_i, y_i) \in S} \mathbf{1}\big[h(x_i) \neq y_i\big]$$

Definition (Prediction (True) Error)**.**

$$L_{\mathcal{D}}(h) = \mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y]$$

Goal: Minimize *true prediction error*, not just training error.

Overfitting

Overfitting occurs when a hypothesis h achieves **low training error** but **high test error** because it learns noise and idiosyncrasies of the training set instead of general patterns.

Example: Take an i.i.d. training set $S = \{(x_1, f(x_1)), \ldots\}$ and return h_s such that

$$h_s(x) = \begin{cases} f(x_i) & \text{if } x = x_i \text{ for some } i \\ \text{flip a coin} & \text{otherwise} \end{cases}$$

- h_s has zero training error but predicts randomly on unseen data.

Key Characteristics

- Fits the training set too closely, including random noise.
- Poor generalization to unseen data.
- Common when the hypothesis space is very flexible (e.g., deep trees, high-degree polynomials).

Overfitting in Decision Trees

- Fully grown decision trees can perfectly memorize the training set.
- This leads to zero empirical error but poor generalization.
- Needs mechanisms like early stopping or pruning to avoid overfitting.

Mitigating Overfitting in Decision Trees

Strategies

- **Limit Model Complexity:** Restrict tree depth or number of nodes.
- **Early Stopping:** Stop splitting when:
  - Error reduction after splitting is small.
  - Too few examples remain in a node.
- **Pruning:** Grow the full tree, then prune back:
  - Replace a subtree with a leaf if it does not significantly increase prediction error.
  - E.g., reduced-error pruning.

Inductive Bias

Definition (Inductive Bias)**.** An **inductive bias** is a set of assumptions a learning algorithm uses to predict unseen data. Without bias, learning from finite samples would be impossible.

Inductive Bias in Decision Trees

- Standard IDT assumes:
  - The simplest consistent tree is preferred.
  - Features are chosen based on information gain or error reduction.
- If tree depth is restricted, bias increases but variance decreases.

Bias-Variance Tradeoff

Definition (Prediction Error Decomposition)**.**

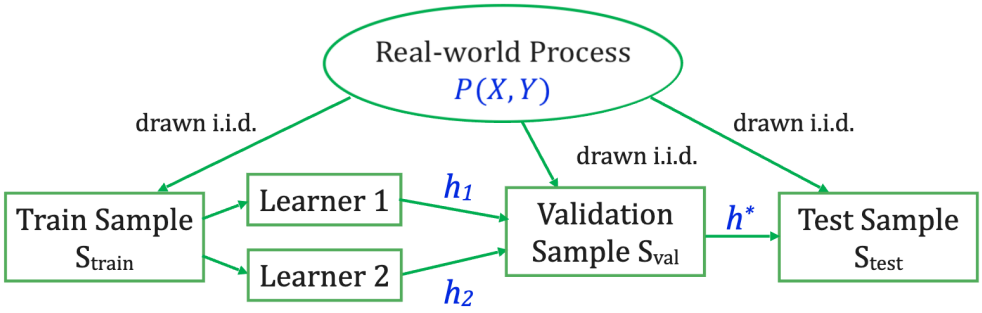
$$\text{Expected Error} = \underbrace{\text{Bias}^2}_{\text{error from assumptions}} + \underbrace{\text{Variance}}_{\text{error from data fluctuations}} + \underbrace{\sigma^2}_{\text{irreducible noise}}$$

- **High bias:** Model too simple → underfits.
- **High variance:** Model too complex → overfits.
- Goal: Balance bias and variance for best generalization.

4 Model Selection and Assessment

Date: Sep 9, 2025

4.1 Validation Sample



- **Training:** Run learning algorithm l times (e.g. different parameters).
- **Validation Error:** Errors $err_{S_{val}}(h_i)$ are estimates of $err_p(h_i)$ for each h_i.
- **Selection:** Use h^* with min $err_{S_{val}}(\hat{h}_i)$ for prediction on test examples.

Two Nested Learning Algorithms

- **Primary Learning Algorithm on** S_{train}
  - For each variant $A_1 \ldots A_l$ of learning algorithm, $h_i = A_i(S_{train})$
  - Example: Decision Tree (DT) that stops at i nodes.
- **Secondary Learning Algorithm on** S_{val}
  - Hypothesis space: $H' = \{h_1, \ldots, h_l\}$
  - Learning Algorithm: $h^* = \arg\min_{h \in H'} [err_{S_{val}}(h)]$

Typical ML Experiment

- Collect data $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)\}$
- Split randomly into S_{train}, S_{val}, S_{test}
- REPEAT
  - Train on S_{train}
  - Validate on S_{val}
- UNTIL we think we have a good rule h.
- Test h on S_{test} to evaluate its accuracy/error.

4.2 Cross-Validation

k-fold Cross-Validation:

- **Given:**
  - Training examples S
  - Learning algorithm A_p with parameter p (model architectures or hyperparameters)
- **Compute:**
  - Randomly partition S into k equally sized subsets $S_1, \ldots, S_k$
  - For each value of p:
    - For i from 1 to k:
      - Train A_p on $S \setminus S_i$ and get h_i
      - Apply h_i to S_i and compute $err_{S_i}(h_i)$
    - Compute cross-validation error:

$$err_{CV}(A_p) = \frac{1}{k} \sum_i err_{S_i}(h_i)$$

- **Selection:**
  - Pick parameter p^* that minimizes $err_{CV}(A_p)$
  - Train $A_{p^*}(S)$ on full sample S to get final h

4.3 Generalization Error of Hypothesis

- **Given**
  - Samples S_{train} and S_{test} of labeled instances
  - Learning Algorithm A
- **Setup**
  - Train learning algorithm A on S_{train}, result is h
  - Apply h to S_{test} and compare predictions against true labels
- **Test**
  - Error on test sample $err_{S_{\text{test}}}(h)$ is estimate of true error $err_p(h)$
  - Compute confidence interval

5 Linear Classifiers

Date: Sep 11, 2025

5.1 Vectors and Hyperplanes

Euclidean Embeddings Data are represented as d dimensional vectors in $\mathbb{R}^d$. The choice of representation is part of "inductive bias".

Euclidean Norm The Euclidean norm of a vector $x \in \mathbb{R}^d$ is defined as:

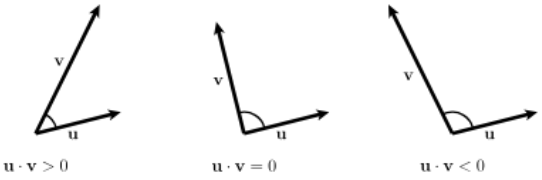
$$||\vec{x}|| = \sqrt{\sum_{i=1}^{d} x_i^2}$$

Dot Product The dot product of two vectors $\vec{x}, \vec{y} \in \mathbb{R}^d$ is defined as:

$$\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{d} x_i y_i$$

Alternative notation: $\vec{x} \cdot \vec{y}$ or $\vec{x}^T \vec{y}$.

Angle & Projection $\vec{w} \cdot \vec{x} = ||\vec{w}||||\vec{x}|| \cos(\theta)$, where θ is the angle between $\vec{w}$ and $\vec{x}$, and $\frac{\vec{w} \cdot \vec{x}}{||\vec{w}||}$ is the signed length of the projection of $\vec{x}$ onto $\vec{w}$.



Hyperplanes In d-dimensional space, a **hyperplane** is defined by a vector $\vec{w} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$:

$$\left\{ \vec{x} \in \mathbb{R}^d \mid \vec{w} \cdot \vec{x} + b = 0 \right\}$$

Geometric interpretation:

- The hyperplane is orthogonal to $\vec{w}$.
- The distance to the origin (along $\vec{w}$) is $-\frac{b}{||\vec{w}||}$.
- All points on the hyperplane satisfy $\vec{w} \cdot \vec{x} = -b$.

5.2 Linear Classifiers

For a vector $\vec{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, the hypothesis $h_{\vec{w},b} : \mathbb{R}^d \to \{-1, +1\}$ is called a d dimensional linear classifier and defined as

$$h_{\vec{w},b}(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b) = \begin{cases} +1 & \vec{w} \cdot \vec{x} + b > 0 \\ -1 & \vec{w} \cdot \vec{x} + b \leq 0 \end{cases}$$

Also called linear predictor or halfspace.

Decision Boundaries in Different Dimensions

- **One dimension:** $h_{w,b}(x) = \text{sign}(wx + b)$
  - Decision boundary: a point (1d hyperplane) at $x = -\frac{b}{w}$
- **Two dimensions:** $h_{\vec{w},b}(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b)$
  - Decision boundary: a line (2d hyperplane) defined by $\vec{w} \cdot \vec{x} + b = 0$
- d **dimensions:** $\text{sign}(\vec{w} \cdot \vec{x} + b)$
  - Decision boundary: a hyperplane $\vec{w} \cdot \vec{x} + b = 0$

Homogenous Linear Classifiers A classifier is **homogenous** if $b = 0$ (otherwise non-homogenous).

Fact: Any d dimensional learning problem for linear classifiers has a **homogenous** form in $d + 1$ dimensions.

Non-homogenous:	**Homogenous:**
$HS^d = \{h_{\vec{w},b} \mid \vec{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$	$HS^{d+1}_{homog} = \{h_{\vec{w},b} \mid \vec{w} \in \mathbb{R}^{d+1}\}$
• $\vec{x}$	• $\vec{x}' = (\vec{x}, 1)$
• $\vec{w}, b$	• $\vec{w}' = (\vec{w}, b)$
• $\vec{w} \cdot \vec{x} + b$	• $\vec{w}' \cdot \vec{x}' = \vec{w} \cdot \vec{x} + b$

Without loss of generality, we will now focus on **homogenous linear classifiers** with $||w_i|| = 1$.

Consistent Linear Classifiers

- Data set of labelled instances $\mathcal{S} = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_m, y_m)\}$.
- Linear classifier $h_{\vec{w}}$ is **consistent** with $\mathcal{S}$ if for all $(\vec{x}_i, y_i) \in \mathcal{S}$:
  - $\vec{w} \cdot \vec{x}_i > 0$ if $y_i = 1$
  - $\vec{w} \cdot \vec{x}_i \leq 0$ if $y_i = -1$
- Data set $\mathcal{S}$ is **linearly separable** (zero training error) if there is a linear classifier $h_{\vec{w}}$ that is consistent with it.

Margin A data set $\mathcal{S}$ is linearly separable with a **(geometric) margin** γ if:

- There is a linear classifier $h_{\vec{w}}$ that is consistent with $\mathcal{S}$.
- The distance of any instance in $\mathcal{S}$ to the decision boundary of $h_{\vec{w}}$ is at least γ.

Mathematical definition: There is $\vec{w}$ such that $||\vec{w}|| = 1$ and for all data points $(\vec{x}_i, y_i) \in \mathcal{S}$:

$$\begin{cases} \vec{w} \cdot \vec{x}_i \geq \gamma & \text{if } y_i = 1 \\ \vec{w} \cdot \vec{x}_i \leq -\gamma & \text{if } y_i = -1 \end{cases} \iff y_i(\vec{w} \cdot \vec{x}_i) \geq \gamma$$

Larger margin $\implies$ Easier to find consistent linear classifier.

5.3 Perceptron Algorithm

Algorithm

The Perceptron Algorithm (homogeneous & batch)

- **Input:** training data $\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)\}$
- **Initialize** $\vec{w}^{(0)} = (0, \ldots, 0)$ and $t = 0$
- **While** there is $i \in [m]$ such that $y_i(\vec{w}^{(t)} \cdot \vec{x}_i) \leq 0$:
  - $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_i \vec{x}_i$
  - $t \leftarrow t + 1$
- **End While**
- **Output** $\vec{w}^{(t)}$

Good Practice: Initialize $\vec{w}^{(0)}$ randomly. Shuffle $\mathcal{S}$ and check data one-by-one for update condition. Iterate until no updates are needed or maximum iterations are reached.