



# Naive Bayes: Weather Prediction Example

- **Problem**

- Predict if kids play outside using past weather observations

- **ML formulation**

- Supervised learning
- Predictor vars:
  - outlook = {sunny, overcast, rainy}
  - temperature = {hot, mild, cold}
  - humidity = {high, normal}
  - windy = {true, false}
- Response var:
  - play = {yes, no}
- Training set:
  - Samples for predictors and response from observations
  - Possible noise in data (e.g., kids have different preferences, some are sick illness, some have homework)

Outlook	Temp	Humidity	Windy	Play
Overcast	Cold	Normal	True	Yes
Overcast	Hot	High	False	Yes
Overcast	Hot	Normal	False	Yes
Overcast	Mild	High	True	Yes
Rainy	Cold	Normal	False	Yes
Rainy	Cold	Normal	True	No
Rainy	Mild	High	False	Yes
Rainy	Mild	High	True	No
Rainy	Mild	Normal	False	Yes
Sunny	Cold	Normal	False	Yes
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Mild	Normal	True	Yes

# Naive Bayes: Weather Prediction Example

---

- Use Bayes' rule to decide class  $H_j$ :

$$\Pr(H_j|\underline{E}) = \frac{\Pr(\underline{E}|H_j) \Pr(H_j)}{\Pr(\underline{E})}$$

where:

- $H_j$ : event to predict
  - E.g., play = yes
- $\underline{E}$ : event with feature values
  - E.g., outlook=sunny, temperature=high, humidity=high, windy=yes

# Naive Bayes: Weather Prediction Example

- The **model** is:

$$\Pr(H_j|\underline{E}) = \frac{\Pr(\underline{E}|H_j) \Pr(H_j)}{\Pr(\underline{E})}$$

where:

- $\Pr(H_j)$ : **prior probability** (probability of the outcome before evidence  $\underline{E}$ )
  - E.g.,  $\Pr(\text{play} = \text{yes})$
  - Computed from training set as:

$$\Pr(H_j) = \sum_{k=1}^N \Pr(H_j \wedge \underline{E}_k)$$

- $\Pr(\underline{E})$ : **probability of the evidence**
  - Computed from training set
  - Not needed as it is common to the probability of each class
- $\Pr(\underline{E}|H_j)$ : **conditional probability**
  - Computed as independent probabilities (the “naive” assumption):

$$\begin{aligned}\Pr(\underline{E}|H_j) &= \Pr(E_1 = e_1, E_2 = e_2, \dots, E_n = e_n|H_j) \\ &\approx \Pr(E_1 = e_1|H_j) \cdot \dots \cdot \Pr(E_n = e_n|H_j)\end{aligned}$$

- Interpretation of Bayes theorem**

- The prior is modulated through the conditional probability and the probability of the evidence

# 1-Rule

---

- Aka “tree stump”, i.e., a decision tree with a single node
- **Algorithm**
  - Pick a single feature (e.g., outlook):
    - Most discriminant
    - Based on expert opinion
  - Choose the most frequent output for the feature's current value
- **Weather problem example:**
  - Pick outlook as single feature
  - Know predictor vars, e.g., outlook = sunny
  - Compute probability of play = yes given outlook = sunny using training set:
$$\Pr(\text{play} = \text{yes} | \text{outlook} = \text{sunny})$$
  - Output the predicted variable

# Naive Bayes: Why Independence Assumption

- **Independence assumption** factors joint probability into marginal probabilities:

$$\begin{aligned}\Pr(\underline{E}|H_j) &= \Pr(E_1 = e_1, E_2 = e_2, \dots, E_n = e_n|H_j) \\ &\approx \Pr(E_1 = e_1|H_j) \cdot \dots \cdot \Pr(E_n = e_n|H_j)\end{aligned}$$

- **Pros:**
  - Simplifies probability computation
  - Aids generalization due to underlying model and so fewer samples needed
- **Cons:**
  - Assumption may not be true

# Estimating Probabilities: MLE

- **Maximum Likelihood Estimate** (MLE) estimates event probability by counting occurrences among all possible events:

$$\Pr(\underline{E} = \underline{e}') = \frac{\#I(\underline{E} = \underline{e}')}{K}$$

- For Naive Bayes, we need to estimate probability of each feature:

$$\Pr(E_i = e') = \frac{\#I(E_i = e')}{\sum_{k=1}^K \#I(E_i = e_k)}$$

- **Problem**

- Value  $e'$  of feature  $E_i$  not in training set with output class  $H_j$
- Estimated probability  $\Pr(E_i = e' | H_j) = 0$
- Plugging  $\Pr(E_i = e' | H_j) = 0$  into

$$\Pr(H_j | \underline{E}) \approx \Pr(E_1 = e_1 | H_j) \cdot \dots \cdot \Pr(E_n = e_n | H_j)$$

SCIENCE YIELDS  $\Pr(H_j | \underline{E}) = 0 \rightarrow$  impossible to decide output  
ACADEMY

# Estimating Probabilities: Laplace Estimator

- Use Laplace estimator for events not in training set instead of MLE
- **Maximum likelihood estimate** (MLE)


$$\Pr(E_i = e') = \frac{\#I(E_i = e')}{\sum_{k=1}^K \#I(E_i = e_k)}$$

- **Laplace estimator**
  - Adds 1 to each count and  $V$  (number of feature values) to denominator:

$$\Pr(E_i = e') = \frac{1 + \#I(E_i = e')}{\sum_{j=1}^V (1 + \#I(E_i = e'_j))} = \frac{1 + \#I(E_i = e')}{V + \sum_{j=1}^V \#I(E_i = e'_j)}$$

- Blends prior of equiprobable feature values with MLE estimates



- 
- ***Decision Trees***
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression

# Decision Tree

---

- **Characteristics**
  - Supervised learning
  - Classification and regression
  - Non-parametric (i.e., no functional form)
- **Model**
  - Decision rules organized in a tree
- **Training**
  - Infer decision rules from data
  - Worst case complexity:  $O(2^n)$  with number of variables
  - Greedy divide-and-conquer improves average complexity
- **Evaluation**
  - Evaluate model from root to leaves
  - Prediction cost:  $O(\log(n))$  with number of training points

# Typical Decision Trees

- **Each node** tests a feature against a constant
  - Split input space with hyperplanes parallel to axes
- **Each feature**:
  - Tested once
  - Labeled with  $(x^{(j)}, <, =, >, t)$
  - Checks feature  $x^{(j)}$  against threshold  $t$
  - Result determines branch to follow
- **Leaves** represent decisions:
  - Class labels (e.g., classification)
    - Predict class or probability
  - Regression function
- Deeper tree, more complex decision rules, fitter model
  - No re-converging paths: it's a tree!
  - Trees are non-linear models using variable interaction in OR-AND form:

$$y_i = (x_1 \geq x'_1) \wedge (x_2 \geq x'_2) \dots \vee \dots$$

# Decision Trees: Pros

---

- Both for **regression and classification**
- **Simple to understand and interpret**
  - White box model: explainable decisions
  - Visualizable
  - Can be created by hand
- Requires **little data preparation**
  - Invariant under feature scaling
    - No data normalization
    - No dummy variables
  - Handles numerical and categorical data
  - Robust to irrelevant features (implicit feature selection)
  - Missing values are treated:
    - As their own value; or
    - Assigned the most frequent value (i.e., imputation)
- **Scalable**
  - Performs well with large datasets

# Decision Trees: Cons

---

- **Learning a decision tree** is NP-complete
  - Much worse than complexity of OLS
  - Algorithms use heuristics (e.g., greedy algorithms)
- **Risk of overfitting**
  - Solutions:
    - Pruning
    - Minimum samples at a leaf node
    - Max depth of trees
- **Some training sets are hard to learn**
  - E.g., XORs, parity
- **Unstable**
  - Small data variations greatly influence the tree
  - Solutions:
    - Ensemble learning / randomization

# Handling of Missing Values

---

- Missing values are treated:
  - As their own value; or
  - Assigned the most frequent value (i.e., imputation)

# Learning Decision Trees: Intuition

---

- **Several algorithms**
  - ID3
  - C4.5
  - CART (Classification And Regression Trees)
- Typically, the problem has a **recursive formulation**
  - Consider the current “leaf”
  - Find the variable/split (“node”) that best separates outcomes into two groups (“leaves”)
    - “Best” = samples with same labels grouped together
  - Continue splitting until:
    - Groups are small enough
    - Maximum depth is reached
    - Sufficiently “pure”

# Splitting at One Node: Problem Formulation

- Consider the  $m$ -th node of a decision tree
  - Given  $p_i = (\underline{x}_i, y_i)$  where  $i = 1, \dots, N_m$ , with training vectors  $\underline{x}_i$  and labels  $y_i$
  - Candidate splits are  $\theta = (j, t_m)$  with feature  $j$  and threshold  $t_m$
  - Each split partitions data into subsets:

$$Q_L(j, t_m) = \{p_i = (\underline{x}_i, y_i) : x_j \leq t_m\}$$

$$Q_R(j, t_m) = \{p_i \notin Q_L\}$$

- Impurity of a split is defined as:

$$H(j, t_m) = \frac{n_L}{N_m} H(Q_L) + \frac{n_R}{N_m} H(Q_R)$$

- Find split  $(j, t_m)^*$  minimizing  $H(j, t_m)$
- **Recurse** on  $Q_L(j, t_m)^*$  and  $Q_R(j, t_m)^*$



# Measures of Node Impurity for Classification

---

- **Measures of node impurity:**
  - Based on probabilities of choosing objects of different classes  $k = 1, \dots, K$  in the  $m$ -th node, i.e.,  $p_k$
  - Smaller impurity values are better
  - Less impurity means smaller probability of misclassification
- **Examples**
  1. Misclassification error
  2. Gini impurity index
  3. Information gain

# Probability of Classification in a Node

- Assume  $m$ -th node has  $N_m$  objects  $x_i$  in  $K$  classes
- Compute class probability in  $m$ -th node as:

$$\hat{f}_{m,k} \triangleq \Pr(\text{pick object of class } k \text{ in } m\text{-th node}) = \frac{1}{N_m} \sum_{x_i \in \text{node}} I(c(x_i) = k)$$

where  $y_i = c(x_i)$  is the correct class for element  $x_i$

- E.g., if there are  $N_m = 10$  objects belonging to  $K = 3$  classes
  - 3 red, 6 blue, 1 green
  - The probability of classification  $\hat{f}_{m,k}$  are:
    - Red = 3/10
    - Blue = 6/10
    - Green = 1/10

# Misclassification Error: Definition

- You have several class probabilities  $p_i$  need a single probability
  - Consider worst case: most common class  $k'$  in node
- **Misclassification error**

$$H_M(p) \triangleq 1 - \max_k p_k$$

- **Binary classifier**
  - Best case (perfect purity)
    - Only one class in node
    - $H_M(p) = 0$
  - Worst case (perfect impurity)
    - 50-50 split between classes in node
    - $H_M(p) = 0.5$
- **Multi-class classifier** with  $K$  classes
  - Misclassification error has upper bound:  $1/K$

# Gini Impurity Index: Definition

- **Gini index**  $H_G(p)$  is probability of picking an element randomly and classifying it incorrectly
  - By using the law of total probability:

$$H_G(p) = \Pr(\text{pick elem of } k\text{-th class}) \cdot \Pr(\text{misclassification} \mid \text{elem of } k \text{ class})$$

$$= \sum_{k=1}^K p_k \cdot (1 - p_k)$$

- **Binary classifier**
  - $H_G(p)$  is between 0 (perfect purity) and 0.5 (perfect impurity)
- **Multi-class classifier** with  $K$  classes
  - $H_G(p)$  has upper bound:  $1 - K \frac{1}{K}^2 = 1 - \frac{1}{K}$

# Information Gain: Definition

---

- Aka cross-entropy (remember entropy is  $-p \log p$ )
- **Information gain**

$$H_{IG} = - \sum_{k=1}^K p_k \cdot \log_2(p_k)$$

- **Binary classifier**
  - $H_{IG}$  varies between 0 (perfect purity) and 1 (perfect impurity)
- **Multi-class classifier** with  $K$  classes
  - $H_{IG}$  has upper bound:  $\log K$

# Measures of Impurity: Examples

- Consider the case of 16 elements in a node , belonging to 2 classes
- If all elements are of the same class:
  - Misclassification error = 0
  - Gini index =  $1 - (1 - 0) = 0$
  - Information gain =  $-(1 \cdot \log_2(1) - 0 \cdot \log_2(0)) = 0$
- If one element is of one class:
  - Misclassification error =  $1 - \max(\frac{1}{16}, \frac{1}{15}) = \frac{1}{16}$
  - Gini index =  $1 - ((\frac{1}{16})^2 + (\frac{1}{15})^2) = 0.12$
  - Information gain =  $-(\frac{1}{16} \log_2(\frac{1}{16}) + \frac{15}{16} \log_2(\frac{15}{16})) = 0.34$
- If elements are split in the two classes equally:
  - Misclassification error =  $\frac{8}{16} = 0.5$
  - Gini index =  $1 - (\frac{8}{16}^2 + \frac{8}{16}^2) = 0.5$
  - Information gain =  $-(\frac{8}{16} \log_2(\frac{8}{16}) + \frac{8}{16} \log_2(\frac{8}{16})) = 1$

# Measures of Impurity for Regression

---

- For continuous variables with  $N_m$  observations at a node, minimize mean squared error:

$$c_m = \frac{1}{N_m} \sum_{i \in N_m} y_i \quad (\text{average class})$$

$$H = \frac{1}{N_m} \sum_{i \in N_m} (y_i - c_m)^2 \quad (\text{variance})$$

# Tips for Using Trees

---

- **Decision trees overfit** with **many features**
  - Get right ratio of training samples to features
- **Solutions**
  - Use dimensionality reduction (PCA, feature selection) to remove non-discriminative features
  - Use maximum tree depth to prevent overfitting
  - Control minimum number of examples at a leaf node / split
    - Small number → overfitting
    - Large number → no learning
- **Balance dataset before training** to avoid **bias**
  - Normalize sum of sample weights for each class



# Feature Selection with Trees

---

- **Intuition**
  - Top tree features predict more samples
- **Importance of a variable**
  - Feature's control over samples estimates importance
  - Feature's depth as a decision node assesses importance
- **Trees are unstable**
  - Reduce estimation variance by averaging variable depth over multiple randomized trees (random forest)

# Embeddings with Trees

---

- **Intuition**
  - Learning a tree is like a non-parametric density estimation
  - Neighboring data points likely lie within the same leaf
- **Embedding: unsupervised data transformation**
  - Tree encodes data by indices of leaves a data point belongs to
  - Index encoded one-hot for high-dimensional, sparse, binary coding
- Use **number of trees and max depth per tree** to control space size

- Decision Trees
- ***Random Forests***
- Linear Models
- Perceptron
- Logistic Regression

# From Decision Trees to Random Forests

- Decision trees are **high capacity models**:
  - Low bias
  - High variance
- Idea: apply ensemble methods to trees → “random forests”
- **Bagging**
  - Reduces variance in “unstable” non-linear models
    - Learning trees is unstable
  - Best with complex models (e.g., fully grown trees)
    - Boosting works best with weak models (e.g., shallow decision trees, aka tree stumps)
  - Works for regression and classification
  - Customizable for trees
    - Different types of randomization in trees
- **Bias-variance trade-off** in random forests
  - Forest bias could increase compared to a single non-random tree
  - Forest variance reduced by averaging, usually compensating for bias

# Randomization in Trees

---

- **Bagging** (bootstrap aggregate) / perturb-and-combine techniques designed for trees
  1. Training samples (with replacement)
  2. Picking features (random subspaces)
  3. Decision split thresholds
  4. All the above
- **Random forests**
  - Each tree built from samples drawn with replacement (bootstrap sample)
  - Split picked as best among random subset of features
- **Extremely randomized trees** (aka “Extra-Trees”)
  - Thresholds randomized
  - More randomness than random forests
  - Trade off more bias for variance
- **Combine random forests**
  - Majority vote on class
  - Averaging prediction probability
  - Averaging prediction

# Random Forests: Pros and Cons

---

- Pros and cons are the same as ensemble learning
- **Pros**
  - Increased accuracy
- **Cons**
  - Lower training and evaluation speed
  - Loss of interpretability
  - Overfitting (cross-validation is needed)

- Decision Trees
- Random Forests
- ***Linear Models***
- Perceptron
- Logistic Regression

# Linear Regression Model

- **Data set**

- $(\underline{x}_1, y_1), \dots, (\underline{x}_N, y_N)$
- $N$  examples
- $P$  features,  $\underline{x}_i \in \mathbb{R}^P$

- **Linear regression model**

$$h(\underline{x}) = \sum_{i=1}^P w_i x_i = \underline{\mathbf{w}}^T \underline{\mathbf{x}} \in \mathbb{R}$$

- Add **bias term**  $w_0$  to model with  $x_0 = 1$  to data

$$h(\underline{\mathbf{x}}) = w_0 + \sum_{i=1}^P w_i x_i = \sum_{i=0}^P w_i x_i = \underline{\mathbf{w}}^T \underline{\mathbf{x}}$$



# Linear Regression: In-sample Error

- For regression, use **squared error for in-sample error**:

$$E_{in}(h) = \frac{1}{N} \sum_{i=1}^N (h(\underline{\mathbf{x}}_i) - f(\underline{\mathbf{x}}_i))^2$$

- Squared error for **linear regression**

$$E_{in}(h) = E_{in}(\underline{\mathbf{w}}) = \frac{1}{N} \sum_{i=1}^N (\underline{\mathbf{w}}^T \underline{\mathbf{x}}_i - y_i)^2$$

- Squared error **in vector form**

$$E_{in}(h) = \frac{1}{N} \|\underline{\mathbf{X}}\underline{\mathbf{w}} - \underline{\mathbf{y}}\|^2 = \frac{1}{N} (\underline{\mathbf{X}}\underline{\mathbf{w}} - \underline{\mathbf{y}})^T (\underline{\mathbf{X}}\underline{\mathbf{w}} - \underline{\mathbf{y}})$$

where:

- $\underline{\mathbf{X}}$  is the matrix with examples  $\underline{\mathbf{x}}_i^T$  as rows (“design matrix”)
- $\underline{\mathbf{X}}$  is a tall matrix with few parameters ( $P$ ) and many examples ( $N$ )
- $\underline{\mathbf{y}}$  is the column vector with all outputs (target vector)

# Linear Regression: Find Optimal Model

- You want to minimize  $E_{in}(\underline{\mathbf{w}}) = (\underline{\mathbf{X}}\underline{\mathbf{w}} - \underline{\mathbf{y}})^T(\underline{\mathbf{X}}\underline{\mathbf{w}} - \underline{\mathbf{y}})$  with respect to  $\underline{\mathbf{w}}$

$$\nabla E_{in}(\underline{\mathbf{w}}^*) = \underline{\mathbf{0}}$$

$$\frac{2}{N} \underline{\mathbf{X}}^T (\underline{\mathbf{X}}\underline{\mathbf{w}}^* - \underline{\mathbf{y}}) = \underline{\mathbf{0}}$$

$$\underline{\mathbf{X}}^T \underline{\mathbf{X}}\underline{\mathbf{w}}^* = \underline{\mathbf{X}}^T \underline{\mathbf{y}}$$

- If the square matrix  $\underline{\mathbf{X}}^T \underline{\mathbf{X}}$  is invertible:

$$\underline{\mathbf{w}}^* = (\underline{\mathbf{X}}^T \underline{\mathbf{X}})^{-1} \underline{\mathbf{X}}^T \underline{\mathbf{y}} = \underline{\mathbf{X}}^\dagger \underline{\mathbf{y}}$$

- The matrix  $\underline{\mathbf{X}}^\dagger \triangleq (\underline{\mathbf{X}}^T \underline{\mathbf{X}})^{-1} \underline{\mathbf{X}}^T$  is called **pseudo-inverse**
  - It generalizes the inverse for non-square matrices, in fact:
    - $\underline{\mathbf{X}}^\dagger \underline{\mathbf{X}} = \underline{\mathbf{I}}$
    - If  $\underline{\mathbf{X}}$  is square and invertible, then  $\underline{\mathbf{X}}^\dagger = \underline{\mathbf{X}}^{-1}$

# Complexity of One-Step Learning

---

- Learning with the pseudo-inverse is **one-step learning**
  - Contrasts with iterative methods, e.g., gradient descent
- Inverting a square matrix of size  $P$  is related to the number of parameters, not examples  $N$ 
  - Complexity of one-step learning is  $O(P^3)$

# Linear Models Are Linear in What?

- A **model is linear** when the signal  $s = \sum_{i=0}^P w_i x_i = \underline{\mathbf{w}}^T \underline{\mathbf{x}}$  is linear **with variables**
  - Unknown variables: weights  $w_i$
  - Inputs  $x_i$  are fixed
- Applying a **non-linear transform to inputs**  $z_i = \Phi(x_i)$  keeps the model linear, e.g.,
  - Positive/negative part (e.g.,  $z_i = \text{RELU}(x_i), \text{RELU}(-x_i)$ )
  - Waterfall (conditioning model to different feature ranges)
  - Thresholding (e.g.,  $z_i = \min(x_i, T)$ )
  - Indicator variables (e.g.,  $z_i = I(x_i > 0)$ )
  - Winsorizing (replace outliers with a large constant value)
- Applying a **non-linear transform to weights**  $z_i = \Phi(w_i)$  makes the model non-linear

# Non-Linear Transformations in Linear Models

- **Transform variables**

- Use  $\Phi : \mathcal{X} \rightarrow \mathcal{Z}$
- Transform each point  $\underline{x}_n \in \mathcal{X} = \mathbb{R}^d$  into a point in feature space  $\underline{z}_n = \Phi(\underline{x}_n) \in \mathcal{Z} = \mathbb{R}^{\tilde{d}}$  with  $d \neq \tilde{d}$

- **Learn**

- Learn linear model in  $\mathcal{Z}$ , obtaining  $\underline{\tilde{w}}$  for separating hyperplane

- **Predict**

- Evaluate model on new point in  $\mathcal{Z}$ :

$$y = \text{sign}(\underline{\tilde{w}}^T \Phi(\underline{x})) \text{ or } y = \underline{\tilde{w}}^T \Phi(\underline{x})$$

- Compute **decision boundary**

- In  $\mathcal{X}$  if  $\Phi$  is invertible; or
- By classifying any  $\underline{x} \in \mathcal{X}$  in  $\mathcal{Z}$

- Decision Trees
- Random Forests
- Linear Models
- *Perceptron*
- Logistic Regression

# Example of Classification Problems

---

- **Binary classification problem**

- $y \in \{0, 1\}$ 
  - Typically assign 1 to what you want to detect
- Email: spam, not\_spam
- Online transaction: fraudulent, valid
- Tumor: malignant, benign

- **Multi-class classification problem**

- $y \in \{0, 1, 2, \dots, K\}$
- Email tagging: work, family, friends
- Medical diagnosis: not\_ill, cold, flu
- Weather: sunny, rainy, cloudy, snow

# Linear Regression for Classification

---

- Use **linear regression for classification**
  - Transform outputs into  $\{+1, -1\} \in \mathbb{R}$
  - Learn  $\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n \approx y_n = \pm 1$
  - Use  $\text{sign}(\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n)$  as model (perceptron)
- **Not optimal**: outliers influence fit due to square distance metric
  - Use weights from linear regression to initialize a learning algorithm for classification (e.g., PLA)



# Perceptron Learning Algorithm (PLA)

- First machine learning algorithm discovered
- **Algorithm**
  - Training set  $\mathcal{D} = \{(\underline{\mathbf{x}}_1, y_1), \dots, (\underline{\mathbf{x}}_n, y_n)\}$
  - Initialize weights  $\underline{\mathbf{w}}$ 
    - Random values
    - Use linear regression for classification as seed
  - Pick a misclassified point  $\text{sign}(\underline{\mathbf{w}}^T \underline{\mathbf{x}}_i) \neq y_i$  from training set  $\mathcal{D}$
  - Update weights:  $\underline{\mathbf{w}}(t+1) = \underline{\mathbf{w}}(t) + y_i \underline{\mathbf{x}}_i$ 
    - Like stochastic gradient descent
  - Iterate until no misclassified points
- Algorithm **converges** (slowly) for linearly separable data
- **Pocket version of PLA**
  - Idea
    - Continuously update solution
    - Keep best solution “in the back pocket”
  - Have a solution if stopping after max iterations

# Non-Linear Transformations for Classifications

---

- Classification problems have **varying degrees of non-linear boundaries**:
  1. Non-linearly separable data
    - E.g., + in center, - in corners in a 2-feature scatter plot
  2. Mostly linear classes with few outliers
  3. Higher-order decision boundary
    - E.g., quadratic
  4. Non-linear data-feature relationship
    - E.g., variable threshold

- Decision Trees
- Random Forests
- Linear Models
- Perceptron
- ***Logistic Regression***

# Logistic Regression Is a Probabilistic Classifier

- Logistic regression learns:
  - The probability of each class  $\Pr(y|\underline{x})$  given input  $\underline{x}$
  - Instead of predicting class  $y$  directly
- **Parametric approach**: assume  $\Pr(y|\underline{x}; \underline{w})$  has a known functional form

$$\Pr(y = 1|\underline{x}; \underline{w}) = \text{logit}(\underline{w}^T \underline{x})$$

- **Optimize parameters**  $\underline{w}$  using maximum likelihood

$$\underline{w}^* = \text{argmax}_{\underline{w}} \Pr(\mathcal{D}; \underline{w})$$

- **Predict** by outputting class with highest probability

$$h_{\underline{w}}(\underline{x}) = \begin{cases} +1 & \Pr(y = 1|\underline{x}; \underline{w}) \geq 0.5 \\ -1 & \Pr(y = 1|\underline{x}; \underline{w}) < 0.5 \end{cases}$$

# Logistic Regression: Example

---

- Assume  $y$  is:
  - Event “patient had heart attack”
  - Function of params  $\underline{x}$  (E.g., age, gender, diet)
- Learn  $\Pr(y|\underline{x})$
- In data set  $\mathcal{D}$ :
  - No samples of  $\Pr(y|\underline{x})$ , i.e., probability that someone with  $\underline{x}$  had a heart attack
  - Have realizations: “patient with  $\underline{x}_1$  had a heart attack”, “patient with  $\underline{x}_2$  didn’t”,
- Find best parameters  $\underline{w}$  for logistic regression model to explain data  $\mathcal{D}$