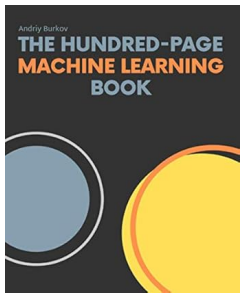
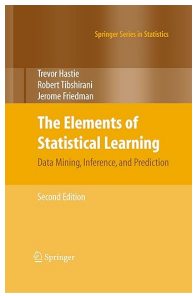
Machine Learning Models

Instructor: Dr. GP Saggese - gsaggese@umd.edu

References:

- Burkov: *"Machine Learning Engineering"* (2020)
- Hastie et al.: *"The Elements of Statistical Learning"* (2nd ed, 2009)





- ***Models***
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

- Models
  - ***Naive Bayes Model***
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Naive Bayes Model

- **Problem**: predict classes $H_1, ..., H_n$ using evidence $\underline{\boldsymbol{E}}$:
  - Use Bayes' rule of conditional probability to decide output class $H_1, ...H_n$ given evidence $\underline{\boldsymbol{E}}$:

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

    where $\underline{\boldsymbol{E}}$ is the vector of features
  - Training data estimates joint probability $\Pr(H_i, \underline{\boldsymbol{E}})$
- **Assumptions:**
  1. Features are independent
  2. Features are equally important (or at least all relevant)
- **Naive Bayes model:**
  - Called "naive" due to the simplifying assumption of independence, even if not true
  - Works surprisingly well

Naive Bayes: Weather Prediction Example

- **Problem**
  - Predict if kids play outside using past weather observations
- **Supervised learning problem**
  - Predictor vars:
    - `outlook = {sunny, overcast, rainy}`
    - `temperature = {hot, mild, cold}`
    - `humidity = {high, normal}`
    - `windy = {true, false}`
  - Response var:
    - `play = {yes, no}`

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

- **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

Naive Bayes: Weather Prediction Example

- Use Bayes' rule to decide class H_j:

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

where:

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

Naive Bayes: Weather Prediction Example

- The **model** is:

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

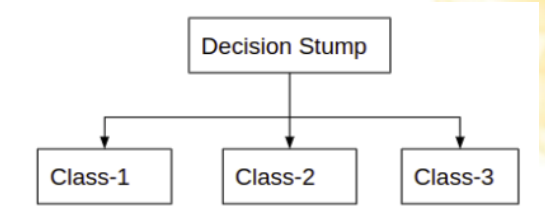
  where:
  - $\Pr(H_j)$: **prior probability** (probability of the outcome before evidence $\underline{\boldsymbol{E}}$)
    - E.g., $\Pr(\text{play} = \text{yes})$
    - Computed from training set as: $\Pr(H_j) = \sum_{k=1}^{N} \Pr(H_j \wedge \underline{\boldsymbol{E}}_k)$
  - $\Pr(\underline{\boldsymbol{E}})$: **probability of the evidence**
    - Computed from training set
    - Not needed as it is common to the probability of each class
  - $\Pr(\underline{\boldsymbol{E}}|H_j)$: **conditional probability**
    - Computed as independent probabilities (the "naive" assumption):

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

- **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; or
    - 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 conditional probability using training set:

$$\Pr(\texttt{play = yes, no} \mid \texttt{outlook = sunny})$$

  - Output the predicted variable

Naive Bayes: Why Independence Assumption

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

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

- **Pros**:
  - Simplifies probability computation
  - Aids generalization due to few 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:

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

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

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

- **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

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

  - Yields $\Pr(H_j|\underline{\boldsymbol{E}}) = 0 \implies$ impossible to decide output

Estimating Probabilities: Laplace Estimator

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

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

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

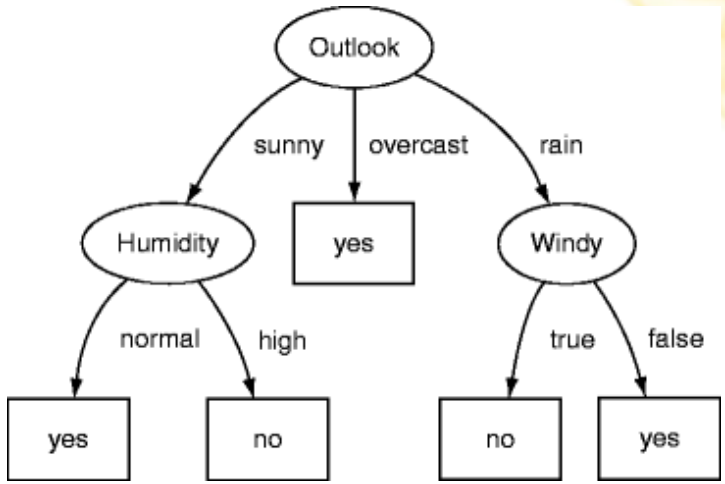
$$\Pr(E_i = e') = \frac{1 + \#I(E_i = e')}{\sum_{j=1}^{V}(1 + \#I(E_i = e'))} = \frac{1 + \#I(E_i = e')}{V + \sum_{j=1}^{V} \#I(E_i = e')}$$

  - Blends prior of equiprobable feature values with MLE estimates

- Models
  - Naive Bayes Model
  - ***Decision Trees***
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

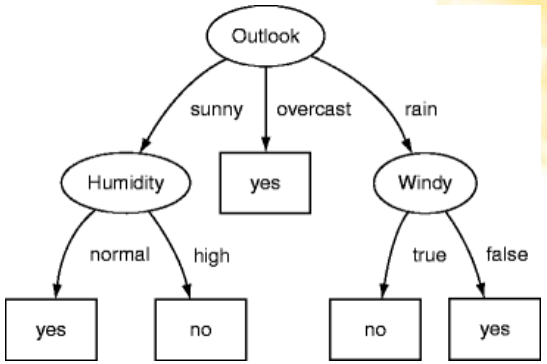
Decision Tree

- **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
  - Greedy divide-and-conquer
  - Worst case complexity: $O(2^n)$ with number of variables
- **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
  - Checks feature $x^{(j)}$ against threshold t
    - E.g., $x^{(j)}\{<, =, >\}t$
  - Result determines branch to follow
- **Leaves** represent decisions:
  - Class labels (e.g., classification)
    - Predict class or probability
  - Regression function



- Trees are non-linear models using variable interaction in `OR-AND` form:

$$y_i = (x_1 \geq x_1') \wedge (x_2 \geq x_2')...) \vee (...)$$

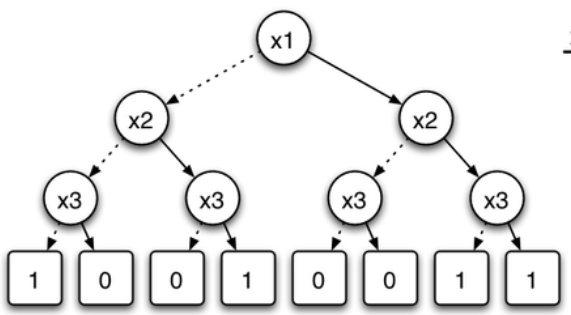
  - No re-converging paths: it's a tree!
  - Deeper tree $\implies$ more complex decision rules $\implies$ fitter model

Decision Trees: Pros

- Works for both **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 optimal 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

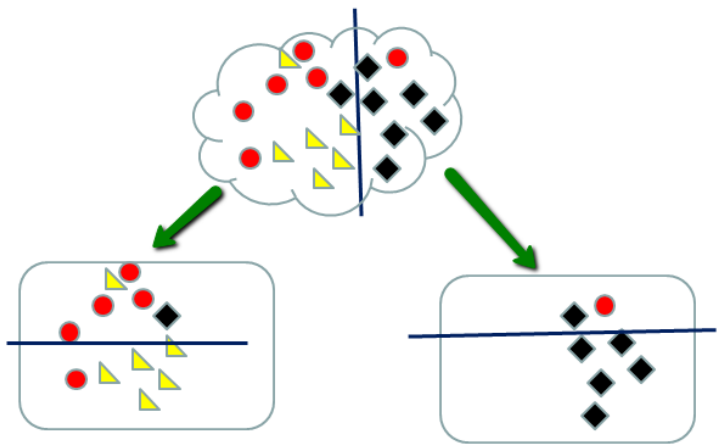


x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- **Unstable**
  - Small data variations greatly influence the tree
  - Solutions:
    - Ensemble learning / randomization

Learning Decision Trees: Intuition

- **Several algorithms**
  - ID3
  - C4.5 (proprietary)
  - 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 $p_i = (\underline{\boldsymbol{x}}_i, y_i)$ where $i = 1, ..., N_m$, with training vectors $\underline{\boldsymbol{x}}_i$ and labels y_i
  - Candidate splits are $\theta = (j, t_m)$ with feature j and threshold t_m
  - Each split partitions data into subsets:

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

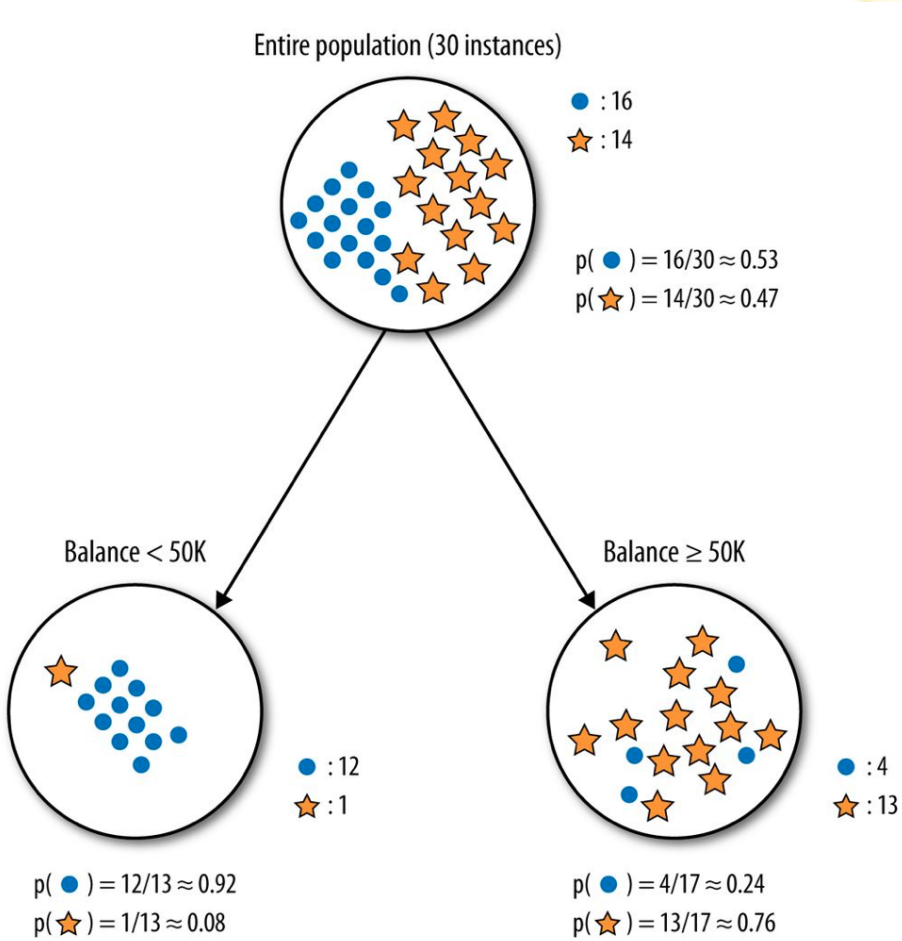
  - Impurity of a split is defined as:

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

  - 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 $k = 1, ..., K$ 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 and 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:

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

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

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 $\rightarrow$ overfitting
    - Large number $\rightarrow$ 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

- Models
  - Naive Bayes Model
  - Decision Trees
  - ***Random Forests***
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

From Decision Trees to Random Forests

- Decision trees are **high capacity models**:
  - Low bias
  - High variance
- Idea: apply ensemble methods to trees $\rightarrow$ "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 increase

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)

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - ***Linear Models***
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Linear Regression Model

- **Data set**
  - $(\underline{\boldsymbol{x}}_1, y_1), ..., (\underline{\boldsymbol{x}}_N, y_N)$
  - N examples
  - P features, $\underline{\boldsymbol{x}}_i \in \mathbb{R}^P$
- **Linear regression model**

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

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

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

Linear Regression: In-sample Error

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

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

- Squared error for **linear regression**

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

- Squared error **in vector form**

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

where:

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

Linear Regression: Find Optimal Model

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

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

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

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

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

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 $s = \sum_{i=0}^{P} w_i x_i = \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}$ 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., $z_i = RELU(x_i), RELU(-x_i)$)
  - 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 $\underline{\boldsymbol{x}}_n \in \mathcal{X} = \mathbb{R}^d$ into a point in feature space $\underline{\boldsymbol{z}}_n = \Phi(\underline{\boldsymbol{x}}_n) \in \mathcal{Z} = \mathbb{R}^{\tilde{d}}$ with $d \neq \tilde{d}$
- **Learn**
  - Learn linear model in $\mathcal{Z}$, obtaining $\underline{\tilde{\boldsymbol{w}}}$ for separating hyperplane
- **Predict**
  - Evaluate model on new point in $\mathcal{Z}$:

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

- Compute **decision boundary**
  - In $\mathcal{X}$ if Φ is invertible; or
  - By classifying any $\underline{\boldsymbol{x}} \in \mathcal{X}$ in $\mathcal{Z}$

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - ***Perceptron***
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

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**
  - $y \in \{0, 1, 2, ..., K\}$
  - 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 $\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n \approx y_n = \pm 1$
  - Use sign($\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n$) 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 $\mathcal{D} = \{(\underline{\boldsymbol{x}}_1, y_1), ..., (\underline{\boldsymbol{x}}_n, y_n)\}$
  - Initialize weights $\underline{\boldsymbol{w}}$
    - Random values
    - Use linear regression for classification as seed
  - Pick a misclassified point $\text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i) \neq y_i$ from training set $\mathcal{D}$
  - Update weights: $\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) + y_i \underline{\boldsymbol{x}}_i$
    - 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

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - ***Logistic Regression***
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Logistic Regression Is a Probabilistic Classifier

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

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

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

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

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

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

Logistic Regression: Example

- Assume y is:
  - The event y_i *"patient with characteristics* $\underline{\boldsymbol{x}}$ *had heart attack"*
  - Function of parameters $\underline{\boldsymbol{x}}$ (e.g., age, gender, diet)
- In data set $\mathcal{D}$:
  - No samples of $\Pr(y|\underline{\boldsymbol{x}})$
  - Have realizations:
    - *"Patient with* $\underline{\boldsymbol{x}}_1$ *had a heart attack"*
    - *"Patient with* $\underline{\boldsymbol{x}}_2$ *didn't"*
    - ...
- Learn $\Pr(y|\underline{\boldsymbol{x}})$
  - Find best parameters $\underline{\boldsymbol{w}}$ for logistic regression model to explain data $\mathcal{D}$

Logistic Function

- Aka "sigmoid"
- **Logistic function** logit(s) is defined as

$$\text{logit}(s) \triangleq \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

  - Varies in [0, 1]
  - Crosses the origin at 0.5
  - Asymptotes at 0 and 1
  - It is a soft version of sign()

Logistic Regression vs Linear Classifier

- **Functional form is similar**
  - **Logistic regression**: $h(\underline{\boldsymbol{w}}) = \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}})$
  - **Linear classifier** (perceptron): $h(\underline{\boldsymbol{w}}) = \text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}})$
- **Difference** in probabilistic interpretation and fitting method
  - Logistic regression** lacks samples of probability function to interpolate
    - Uses realizations of random variable
    - Seeks model parameters maximizing data likelihood
  - **Linear classification** assumes class value is linear function of inputs

Error for Probabilistic Binary Classifiers

- For probabilistic binary classification, use **log-probability error** as point-wise error

$$e(h(\underline{\boldsymbol{x}}), y) \triangleq -\log(\Pr(y = h(\underline{\boldsymbol{x}})|\underline{\boldsymbol{x}}; \underline{\boldsymbol{w}}))$$

  - Negate for positive errors: $\log([0, 1]) \in [-\infty, 0)$
- Log probability generalizes 0-1 error
  - Case $y = 1$
    - Output $h(\underline{\boldsymbol{x}})$ close to 1 $\implies$ probability 1 $\implies$ $\log(1) = 0$ $\implies$ $e(\cdot) = 0$
    - Output close to 0 $\implies$ $e() = -\log(0) \to +\infty$
  - Similar behavior for $y = 0$

One-Liner Error for Probabilistic Binary Classifiers

- **Point-wise error** for example $(\underline{\boldsymbol{x}}, y)$ for probabilistic binary classifiers is defined as:

$$
\begin{aligned}
e(h(\underline{\boldsymbol{x}}), y) &\triangleq -\log(\Pr(h(\underline{\boldsymbol{x}}) = y|\underline{\boldsymbol{x}})) \\
&= \begin{cases} -\log(\Pr(y=1|\underline{\boldsymbol{x}})) & y=1 \\ -\log(\Pr(y=0|\underline{\boldsymbol{x}})) & y=0 \end{cases} \\
&= \begin{cases} -\log(\Pr(y=1|\underline{\boldsymbol{x}})) & y=1 \\ -\log(1-\Pr(y=1|\underline{\boldsymbol{x}})) & y=0 \end{cases}
\end{aligned}
$$

- Any function of a binary variable:

$$
y = \begin{cases} a & x=1 \\ b & x=0 \end{cases}
$$

  can be written as one-liner: $y = x \cdot a + (1-x) \cdot b$

- Point-wise error can be written independently of $\Pr(y=1|\underline{\boldsymbol{x}})$:

$$
e(h(\underline{\boldsymbol{x}}), y) = -y \log(\Pr(y=1|\underline{\boldsymbol{x}}) - (1-y)\log(1-\Pr(y=1|\underline{\boldsymbol{x}}))
$$

One-Liner Error for Logistic Regression

- The point-wise error for a binary classifier is:

$$e(h(\underline{\boldsymbol{x}}), y) = -y \log(\Pr(y = 1|\underline{\boldsymbol{x}}) - (1 - y) \log(1 - \Pr(y = 1|\underline{\boldsymbol{x}}))$$

- Simplify further **with logit function**:

$$\begin{aligned} e(h(\underline{\boldsymbol{x}}), y) &\triangleq -\log \Pr(h(\underline{\boldsymbol{x}}) = y) \\ &\quad \ldots \text{ a bit of math manipulation } \ldots \\ &= -\log \text{logit}(y \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}) \qquad \text{since } \text{logit}(s) = \frac{1}{1 + e^{-s}} \\ &= \log(1 + \exp(-y \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}})) \end{aligned}$$

- Point-wise error for logistic regression equals **cross-entropy error**

Cross-Entropy Error

- **Point-wise error for logistic regression** has expression:

$$e(h(\underline{\boldsymbol{x}}), y) = \log(1 + \exp(-y \cdot \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}))$$

  - It is called **cross-entropy error**
  - Note: no $-$ before $\log(\cdot)$ but before $y \cdot \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}$
- Cross-entropy error **generalizes 0-1 error**
  - If $\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}$ agrees with y in sign and $|\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}|$ is large $\implies$ error goes to 0
  - If they disagree in sign $\implies$ error goes towards ∞
- Define **in-sample error on training set** as average of point-wise errors:

$$E_{in} \triangleq \frac{1}{N} \sum_n e(h(\underline{\boldsymbol{x}}_n), y_n)$$

Fitting Logistic Regression

- A plausible error measure of a hypothesis is based on **likelihood of data**

$$\Pr(\mathcal{D}|h = f)$$

  - *"How likely is the data $\mathcal{D}$ under the assumption that $h = f$?"*
  - *"How likely is that the data $\mathcal{D}$ was generated by h?"*

- **Maximize likelihood** $\mathcal{D}$ generated from logistic regression

$$\Pr(y = 1|\underline{\boldsymbol{x}}; \underline{\boldsymbol{w}})$$

- It can be proved that this is equivalent to **minimizing in-sample error** on training set **using cross-entropy error**

Fitting Logistic Regression (Optional)

- Find $\underline{\boldsymbol{w}}$ that maximizes likelihood for data set $\mathcal{D} = \{(\underline{\boldsymbol{x}}_1, y_1), ..., (\underline{\boldsymbol{x}}_N, y_N)\}$ generated by model $h(\underline{\boldsymbol{x}})$:

  $$\Pr(D|\underline{\boldsymbol{w}}) = \Pr(y_1 = h(\underline{\boldsymbol{x}}_1) \wedge ... \wedge y_N = h(\underline{\boldsymbol{x}}_N)) = \Pr(y_1 = y_1' \wedge ... \wedge y_N = y_N')$$

- Model form:

$$y' = h(\underline{\boldsymbol{x}}) = \begin{cases} +1 & \text{if } \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}) > 0.5 \\ -1 & \text{otherwise} \end{cases}$$

- Assuming independence among training examples:

$$\Pr(D|\underline{\boldsymbol{w}}) = \prod_{i=1}^{N} \Pr(y_i = y_i'|\underline{\boldsymbol{x}}_i)$$

- Fold y_n in expression:
  - When $y_n = 1$, $\Pr(y_n = y_n') = \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$
  - When $y_n = -1$, $\Pr(y_n = y_n') = \text{logit}(-\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$
- Thus, $\Pr(y_n = y_n') = \text{logit}(y_n \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$

Fitting Logistic Regression (Optional)

- Given:

$$\Pr(D|\underline{\boldsymbol{w}}) = \prod_{i=1}^{N} \text{logit}(y_n \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$$

- Re-write optimization as minimizing sum of point-wise errors $E_{in} = \sum e(h(\underline{\boldsymbol{x}}_n), y_n)$
  - Maximize log(...) with respect to $\underline{\boldsymbol{w}}$ since log argument > 0 and log() is monotone
- Equivalently, minimize:

$$\begin{aligned} -\frac{1}{N}\log(...) &= -\frac{1}{N}\log(\prod(...)) = -\frac{1}{N}\sum(\log(...)) \\ &= \frac{1}{N}\sum \log(\frac{1}{\text{logit}(y_n \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)}) \\ &= \frac{1}{N}\sum \log(1+\exp(-y_n \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)) = \frac{1}{N}\sum e(h(\underline{\boldsymbol{x}}_n), y_n) = E_{in}(\underline{\boldsymbol{w}}) \end{aligned}$$

Gradient Descent for Logistic Regression

- Gradient descent requires two inputs:
  - Gradient of the cost function $\frac{\partial E}{w_j}$ for all j
  - Cost function $E_{in}(\underline{\boldsymbol{w}})$
- The cost function is:

$$E_{in}(\underline{\boldsymbol{w}}) = \frac{1}{N} \sum_i e(h(\underline{\boldsymbol{x}}_i; \underline{\boldsymbol{w}}), y_i)$$

- The cost function for logistic regression:

$$E_{in}(\underline{\boldsymbol{w}}) = \frac{1}{N} \sum_i \log(1 + \exp(-y_i \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i))$$

- Thus gradient descent converges to global minimum
  - It can be shown that $E_{in}(\underline{\boldsymbol{w}})$ is convex in $\underline{\boldsymbol{w}}$
  - In fact sum of exponentials and flipped exponentials is convex and log is monotone

One-Vs-All Multi-Class Classification

- Aka "one-vs-rest" classifier
- **Problem**: you have n classes $c_1, \ldots, c_n$ to distinguish given $\underline{\boldsymbol{x}}$
- **Learn**
  - Create n binary classification problems where we classify c_i vs c_{-i} (everything but i)
  - Learn n classifiers with optimal $\underline{\boldsymbol{w}}_i$, each estimating $\Pr(y = i|\underline{\boldsymbol{x}}; \underline{\boldsymbol{w}}_i)$
- **Predict**
  - Evaluate the n classifiers
  - Pick the class $y = i$ with the maximum $\Pr(y = i|\underline{\boldsymbol{x}})$

Cost Function for Multi-Class Classification (Opt)

- The cost function for logistic regression is:

$$E_{in}(\underline{\boldsymbol{w}}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \Pr(y=1|\underline{\boldsymbol{x}}_i) + (1-y_i)\log(1-\Pr(y=1|\underline{\boldsymbol{x}}_i)))$$

- Encode expected outputs $\underline{\boldsymbol{y}}_i$ one-hot
  - j-th element $\underline{\boldsymbol{y}}_i|_j$ is 1 if correct class is j-th
  - E.g., for $k = 4$ 1000
- Using $\underline{\boldsymbol{h}}(\underline{\boldsymbol{x}})$ as model outputs and $\Pr(y=1|\underline{\boldsymbol{x}}) = p(\underline{\boldsymbol{x}})$:

$$E_{in}(\underline{\boldsymbol{w}}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\underline{\boldsymbol{y}}_i \log(\underline{\boldsymbol{p}}(\underline{\boldsymbol{x}}_i)) + (1-\underline{\boldsymbol{y}}_i)\log(1-\underline{\boldsymbol{p}}(\underline{\boldsymbol{x}}_i))\right)|_k$$

- Innermost summation considers error on each class/digit
  - E.g., for $k = 4$ 1000 vs 0100
  - Equal digits don't contribute to error
  - Different digits give positive contribution

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - ***LDA, QDA***
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Basic Idea of Parametric Models

- Assume a model generates data
  - Known functional form
  - Parametrized with unknown parameters to estimate
- **Pros**
  - Utilize data structure
  - Easy to fit: few parameters
  - Accurate predictions if assumptions correct
- **Cons**
  - Strong data assumptions
  - Low accuracy if assumptions incorrect

Linear and Quadratic Discriminant Analysis

- Aka LDA and QDA
- **Parametric models**
  - Assume each class generating process is multivariate Gaussian
  - Classifiers with linear and quadratic decision surface
- **Pros**
  - Closed-form solutions easy to compute (sample mean and covariance)
  - Inherently multiclass
  - No hyperparams to tune
- **Cons**
  - Strong assumptions about the data

LDA / QDA: Model Form

- Both LDA and QDA assume class generating process $f_k(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu_k}}, \underline{\underline{\boldsymbol{\Sigma_k}}})$ is **multivariate Gaussian**
- **Linear discriminant analysis (LDA)** model:

$$f_k(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu_k}}, \underline{\underline{\boldsymbol{\Sigma_k}}}) \sim \mathcal{N}(\underline{\boldsymbol{\mu_k}}, \underline{\underline{\boldsymbol{\Sigma}}}))$$

  - Means $\underline{\boldsymbol{\mu_k}}$ differ for all k classes
  - Covariance matrix $\underline{\underline{\boldsymbol{\Sigma}}}$ same for all k classes
  - Classes separated by linear decision boundaries
- **Quadratic discriminant analysis (QDA)** model:
  - Classes k have different covariance matrix $\underline{\underline{\boldsymbol{\Sigma_k}}}$
  - Classes separated by quadratic boundaries

Bayes Theorem for LDA / QDA

- Consider a classification setup with multi-class output $Y \in \{1, ..., K\}$
- Build a **parametric model for the conditional distribution**:

$$\Pr(Y = k | X = \underline{\boldsymbol{x}})$$

- Use **Bayes theorem**:

$$\Pr(Y = k | X = \underline{\boldsymbol{x}}) = \frac{\Pr(X = \underline{\boldsymbol{x}} | Y = k) \cdot \Pr(Y = k)}{\Pr(X = \underline{\boldsymbol{x}})}$$

where:

  - $\Pr(X = \underline{\boldsymbol{x}} | Y = k)$: given a class, estimate probability of $\underline{\boldsymbol{x}}$
  - $\Pr(Y = k) = \pi_k$: probability of each class (prior)
  - $\Pr(X = \underline{\boldsymbol{x}})$: probability of each input

- **Estimate probabilities** from data

LDA / QDA: Boundary Decision (Optional)

- Consider the ratio between the probabilities of $Y = k$ vs $Y = j$:

$$r = \frac{\Pr(Y = k|X = \underline{x})}{\Pr(Y = j|X = \underline{x})}$$

- Using the model assumption and Bayes theorem:

$$\begin{aligned}\Pr(Y = k|X = \underline{x}) &\propto \Pr(X = \underline{x}|Y = k) \cdot \Pr(Y = k) \\ &= f_k(\underline{x}; \underline{\mu_k}) \cdot \pi_k\end{aligned}$$

where:

$$f_k(\underline{x}) = c \cdot \exp\left(-\frac{1}{2}(\underline{x} - \underline{\mu_k})^T \underline{\underline{\Sigma}}^{-1}(\underline{x} - \underline{\mu_k})\right)$$

- Apply $\log(\cdot)$ as a monotone transformation

$$r = \log \frac{f_k(\underline{x})}{f_j(\underline{x})} + \log \frac{\pi_k}{\pi_j}$$

LDA / QDA: Boundary Decision (Optional)

- Apply $\log(\cdot)$ as a monotone transformation

$$r = \log \frac{f_k(\underline{x})}{f_j(\underline{x})} + \log \frac{\pi_k}{\pi_j}$$

- **Second term** independent of $\underline{x}$ so you can ignore it
- **First term** proportional to:

$$(\underline{x} - \underline{\mu_k})^T \underline{\underline{\Sigma}}^{-1} (\underline{x} - \underline{\mu_k}) - (\underline{x} - \underline{\mu_j})^T \underline{\underline{\Sigma}}^{-1} (\underline{x} - \underline{\mu_j})$$

- Expand, simplify $\underline{x}^T \underline{\underline{\Sigma}}^{-1} \underline{x}$, and note:

$$2\underline{x}^T \underline{\underline{\Sigma}}^{-1} (\underline{\mu_k} - \underline{\mu_j})$$

Resulting in a linear relationship in $\underline{x}$

$$-\frac{1}{2} (\underline{\mu_k} + \underline{\mu_j})^T \underline{\underline{\Sigma}}^{-1} (\underline{\mu_k} - \underline{\mu_j}) + \underline{x}^T \underline{\underline{\Sigma}}^{-1} (\underline{\mu_k} - \underline{\mu_j})$$

LDA / QDA: Learn Model (Optional)

- In practice:
  - Ignore $\Pr(X = \underline{x})$; it's common for all classes
  - Know or estimate prior $\Pr(Y = k) = \pi_k$ from data
  - Estimate conditional probability $\Pr(X = \underline{x} | Y = k)$
- Model assumes Gaussian distribution for conditional probability:

$$\Pr(X = \underline{x} | Y = k) = f_k(\underline{x}; \underline{\mu_k}, \underline{\underline{\Sigma_k}})$$

where:

$$f_k(\underline{x}) = \frac{1}{(2\pi)^n |\underline{\underline{\Sigma_k}}|^{1/2}} \exp\left(-\frac{1}{2}(\underline{x} - \underline{\mu_k})^T \underline{\underline{\Sigma_k}}^{-1} (\underline{x} - \underline{\mu_k})\right)$$

- Estimate parameters $\underline{\mu_k}$, $\underline{\underline{\Sigma_k}}$ using sample mean and covariance

Evaluating LDA / QDA

- For new $\underline{\boldsymbol{x}} = \underline{\boldsymbol{x}}'$, compute for each class $Y = k$

$$\Pr(Y = k | X = \underline{\boldsymbol{x}}) \propto f_k(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu_k}}, \underline{\underline{\boldsymbol{\Sigma_k}}}) \cdot \pi_k$$

- Choose k that maximizes posterior probability
- If $f_i(\underline{\boldsymbol{x}})$ is from a multivariate Gaussian distribution with diagonal $\underline{\underline{\boldsymbol{\Sigma}}}$ (feature independence), you can simplify expressions further

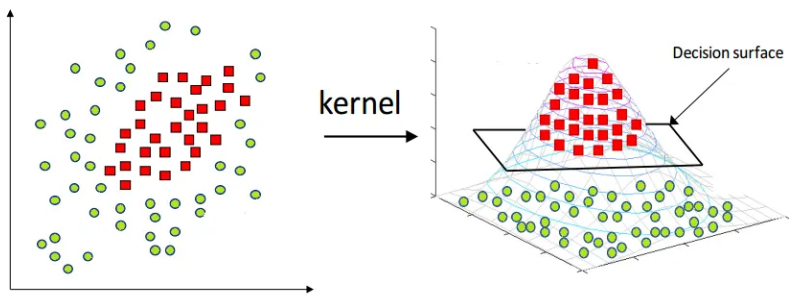
- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - ***Kernel Methods***
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Kernel: Definition

- Consider a transformation $\Phi : \mathcal{X} \rightarrow \mathcal{Z}$
  - E.g., transform features in space $\mathcal{X}$ non-linearly into higher-dimensional space $\mathcal{Z}$
- **Kernel of transformation** Φ yields inner product of two points $\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}' \in \mathcal{X}$ in transformed space $\mathcal{Z}$

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') \triangleq \langle \Phi(\underline{\boldsymbol{x}}), \Phi(\underline{\boldsymbol{x}}') \rangle = \Phi(\underline{\boldsymbol{x}})^T \Phi(\underline{\boldsymbol{x}}') = \underline{\boldsymbol{z}}^T \underline{\boldsymbol{z}}'$$

- Why doing this?



Kernel: Expression From the Transform

- If you have an expression for Φ, compute a closed formula for the kernel
- E.g., if transformation is $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^6$, it introduces interaction terms:

$$\underline{\boldsymbol{z}} = \Phi(\underline{\boldsymbol{x}}) = \Phi(x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

- Kernel of Φ is:

$$\begin{aligned} K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') &= (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)^T (1, x'_1, x'_2, x'^2_1, x'^2_2, x'_1 x'_2) \\ &= 1 + x_1 x'_1 + x_2 x'_2 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + x_1 x_2 x'_1 x'_2 \end{aligned}$$

Gaussian Kernel

- Aka "exponential kernel" or "Radial Basis Function" (RBF) kernel
- A **Gaussian kernel** has the form:

$$K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}'\|^2) = \exp(-\frac{\|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}'\|^2}{\sigma^2})$$

- It can be shown to be an inner product in an infinite dimension $\mathcal{Z}$

Kernel as Way to Measure Similarity

- **Intuition**: The Gaussian kernel

$$K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}'\|^2)$$

  measures "similarity" of point $\underline{\boldsymbol{x}}$ to point $\underline{\boldsymbol{x}}_i$:
  - $K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ is 1 when points are the same
  - Value is 0 when points are distant
  - Effect strength depends on γ
- Using kernels to compute features:
  - Kernels often rely on distance between vectors
    - E.g., euclidean norm $\|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}'\|^2$
  - Need to scale features for similar effects among coordinates

Linear Kernel

- Consider the transformation Φ as the identity function $\Phi(\underline{\boldsymbol{x}}) = \underline{\boldsymbol{x}}$
- The kernel function is:

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = \underline{\boldsymbol{x}}^T \underline{\boldsymbol{x}}'$$

  - A **linear kernel** means using no kernel
  - It is just a "pass-through"

Polynomial Kernel

- Given a point $\underline{\boldsymbol{x}} \in \mathbb{R}^n$, consider the function with two parameters k and d

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = (k + \underline{\boldsymbol{x}}^T \underline{\boldsymbol{x}}')^d$$

- It is called **polynomial** since if you expand the dot product you get a polynomial
- It can be proved that this is always a kernel

Kernel: Identifying a Function as a Kernel

- **Problem**:
  - You have a certain function $K(\underline{x}, \underline{x}')$ and you want to show that $K(\cdot)$ is an inner product in the form for some function $\Phi(\cdot)$

$$K(\underline{x}, \underline{x}') = \Phi(\underline{x})^T \Phi(\underline{x}') \quad \forall \underline{x}, \underline{x}'$$

    for a certain Φ and $\mathcal{Z}$
- In theory, a given function $K(\underline{x}, \underline{x}')$ is a valid kernel iff:
  - It is a symmetric, and
  - Satisfies the Mercer's condition: the matrix $K(\underline{x}_i, \underline{x}_j)$ is definite semi-positive

Kernel: Example of Identifying a Kernel

- Let's show that:

$$K(\underline{x}, \underline{x}') = (k + \underline{x}^T \underline{x}')^d$$

  is a **kernel** for any n, k, d

- According to the definition you need to show that there is always a transform Φ:

$$\Phi : \mathcal{X} = \mathbb{R}^n \rightarrow \mathcal{Z} = \mathbb{R}^q$$

  with $q \gg d$, such that $K_\Phi = (k + \underline{x}^T \underline{x}')^d$

- **Example**
  - $\mathcal{X} = \mathbb{R}^2$, $K(\underline{x}, \underline{x}') = (1 + \underline{x}^T \underline{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2$
  - Compute the full expression in terms of the coordinates:

$$K(\underline{x}, \underline{x}') = (1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2')$$

  - Choose:
    - $\mathcal{Z} = \mathbb{R}^6$
    - $\Phi(x_1, x_2) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$
  - This is a particular case of the polynomial kernel

A Kernel Is a Computational Shortcut

- In literature, the **kernel trick** is a **computational shortcut** for the dot product of transformed vectors
- Compare 2 ways to compute the inner product of transformed vectors for a polynomial kernel
  1. **Using definition**: compute images of vectors, then inner product in transformed space:

$$(1, x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2, ...)^T \cdot (1, x_1', ...)$$

     - Requires combinatorial powers and a large dot product
  2. **Kernel trick**: use kernel function for dot product in transformed space

$$(k + \underline{\boldsymbol{x}}^T \underline{\boldsymbol{x}}')^d$$

     - Requires inner product of small vectors, then power of a number
- Kernel trick is more computationally efficient for inner product computation

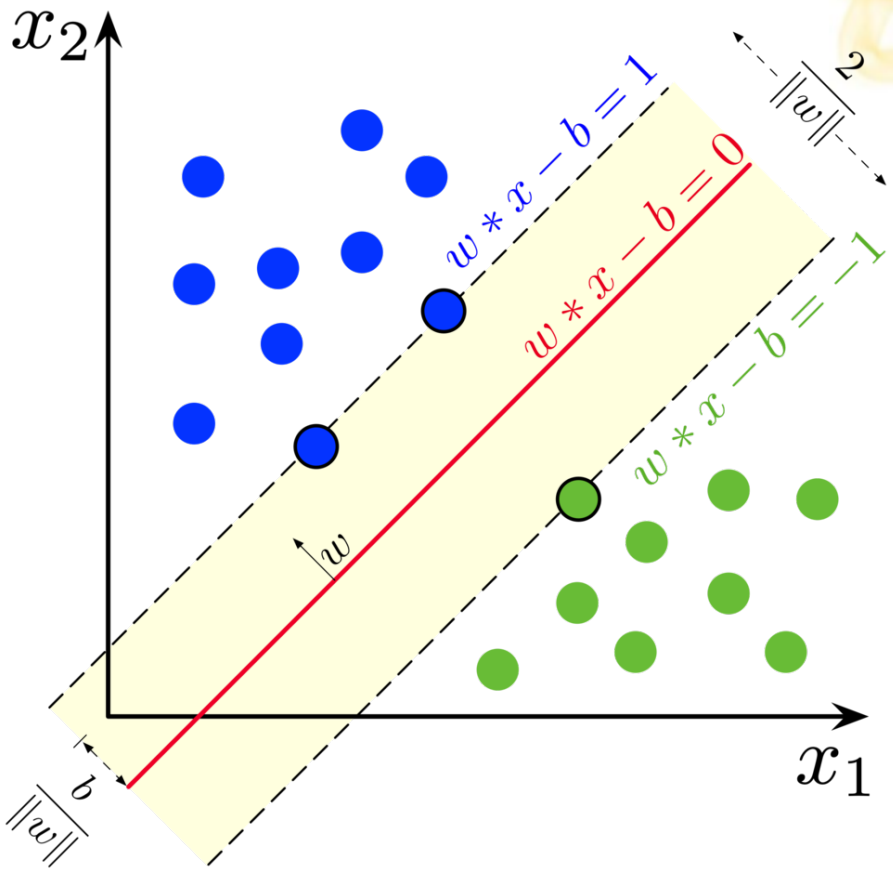
- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - ***Support Vector Machines (Optional)***
  - Similarity-Based Models
  - Clustering
  - Anomaly Detection

Support Vector Machines (SVM)

- Arguably one of the most successful classification algorithm, together with neural networks and random forests
- **Idea**: find a separating hyperplane that maximizes the distance from the class points (aka "margin")
- **All the rage in 2005-2015**
  - Robust classifier handling outliers automatically
  - Strong theoretical justification of out-of-bound error
  - Strong link with VC dimension
  - Cool geometric interpretation
  - Solve a very complex optimization problem with some neat tricks
  - Works for both regression and classification
- SVM for classification:
  - Does not output probabilities (like logistic regression), but predicts directly the class
  - Has a notion of confidence, as distance from the margin

SVM Is a Large Margin Classifier

- Why **large margin** classifier is good?
- Given a linearly separable data set, the optimal separating line maximizes the margin:
  - More robust to noise
  - Large margin reduces VC dimension of hypothesis set



SVM: Notation and Conventions

- Assume that:
  1. Outputs are encoded as $y_i \in \{-1, 1\}$
  2. Pull out w_0 from $\underline{\boldsymbol{w}}$
     - The bias $w_0 = b$ plays a different role
     - $\underline{\boldsymbol{w}} = (w_1, ..., w_d)$ and there is no $x_0 = 1$
     - $\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + b = 0$ is the equation of the separating hyperplane
  3. $\underline{\boldsymbol{x}}_n$ is the closest point to the hyperplane
     - It can be multiple points from different classes
- Normalize $\underline{\boldsymbol{w}}$ and b to get a canonical representation of the hyperplane imposing $|\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n + b| = 1$

SVM: Original Form of Problem

- The SVM problem is:

$$\begin{aligned} &\text{find } \underline{\boldsymbol{w}}, b \\ &\text{maximize } \frac{1}{\|\underline{\boldsymbol{w}}\|} && \text{(max margin)} \\ &\text{subject to } \min_{i=1,\ldots,n} |\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b| = 1 && \text{(hyperplane)} \end{aligned}$$

- This problem is not friendly to optimization since it has norm, min, and absolute value

Primal Form of SVM Problem

- You can rewrite it as:

$$\begin{aligned} &\text{find} && \underline{\boldsymbol{w}}, b \\ &\text{minimize} && \frac{1}{2}\underline{\boldsymbol{w}}^T\underline{\boldsymbol{w}} \\ &\text{subject to} && y_i(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b) \geq 1 \; \forall i = 1, ..., n \end{aligned}$$

- Note that under $\underline{\boldsymbol{w}}$ minimal and linear separable classes, it is guaranteed that for at least one $\underline{\boldsymbol{x}}_i$ in the second equation will be equal to 1 (as in the original problem)
  - In fact otherwise we could scale down $\underline{\boldsymbol{w}}$ and b (which does not change the plane) to use the slack, against the hypothesis of minimality of $\underline{\boldsymbol{w}}$

Dual (Lagrangian) Form of SVM Problem

$$\text{minimize with respect to } \underline{\boldsymbol{\alpha}} \qquad \mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \underline{\boldsymbol{x}}_i^T \underline{\boldsymbol{x}}_j$$

$$\text{subject to} \qquad \underline{\boldsymbol{\alpha}} \geq \underline{\mathbf{0}}, \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i$$

- The equation for $\underline{\boldsymbol{w}}$ is not a constraint, but it computes $\underline{\boldsymbol{w}}$ (the plane) given $\underline{\boldsymbol{\alpha}}$, while b is given by $\min|\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b| = 1$

Dual Form of SVM as QP Problem

- The dual form of SVM problem is a convex quadratic programming problem, in the form:

$$\begin{aligned} \text{minimize with respect to } \underline{\boldsymbol{\alpha}} \quad & \underline{\mathbf{1}}^T \underline{\boldsymbol{\alpha}} - \frac{1}{2} \underline{\boldsymbol{\alpha}}^T \underline{\underline{\boldsymbol{Q}}} \underline{\boldsymbol{\alpha}} \\ \text{subject to} \quad & \underline{\boldsymbol{\alpha}} \geq 0, \underline{\boldsymbol{y}}^T \underline{\boldsymbol{\alpha}} = 0 \end{aligned}$$

where:

- the matrix is $\underline{\underline{\boldsymbol{Q}}} = \{y_i y_j \underline{\boldsymbol{x}}_i^T \underline{\boldsymbol{x}}_j\}_{ij}$
- $\underline{\boldsymbol{\alpha}}$ is the column vector $(\alpha_1, \ldots, \alpha_N)$

Solving Dual Formulation of SVM Problem (1/2)

- **Solving convex problem** for α
  - Feeding this problem to a QP solver, you get the optimal vector $\underline{\boldsymbol{\alpha}}$
- **Compute hyperplane $\underline{\boldsymbol{w}}$**
  - From $\underline{\boldsymbol{\alpha}}$ recover the plane $\underline{\boldsymbol{w}}$ from the equation: $\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i$
  - Looking at the optimal α_i, you can observe that many of them are 0
  - This is because when you applied the Lagrange multipliers to the inequalities: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1$, you got the KKT condition:

$$\alpha_i(y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) - 1) = 0$$

  - From these equations, either
    - $\alpha_i = 0$ and $\underline{\boldsymbol{x}}_i$ is an *interior point* since it has non-null distance from the plane (i.e., slack) from the plane; or
    - $\alpha_i \neq 0$ and the slack is 0, which implies that the $\underline{\boldsymbol{x}}_i$ point touches the margin, i.e., it is a *support vector*

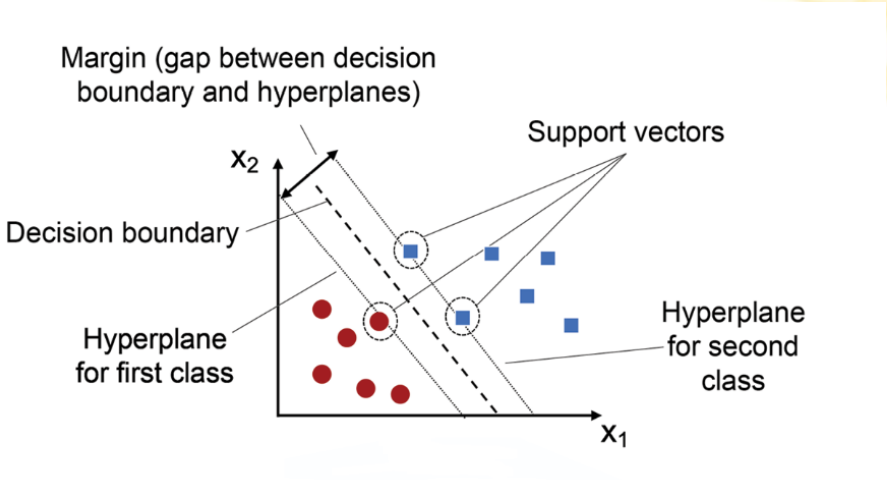
Solving Dual Formulation of SVM Problem (2/2)

- Thus the hyperplane is only function of the support vectors:

$$\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i = \sum_{\underline{\boldsymbol{x}}_i \in \text{SV}} \alpha_i y_i \underline{\boldsymbol{x}}_i$$

since only for the support vectors $\alpha \neq 0$

  - The $\alpha_i \neq 0$ are the real degree of freedom



- **Compute** b
  - Once $\underline{\boldsymbol{w}}$ is known, you can use any support vector to compute b:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) = 1$$

Support Vectors and Degrees of Freedom for SVM

- The number of support vectors is related to the degrees of freedom of the model
- Because of the VC dimension, you have an in-sample quantity to bound the out-of-sample error:

$$E_{out} \leq E_{in} + c\frac{\text{num of SVs}}{N-1}$$

- You are "guaranteed" to not overfit

Non-Linear Transform for SVM

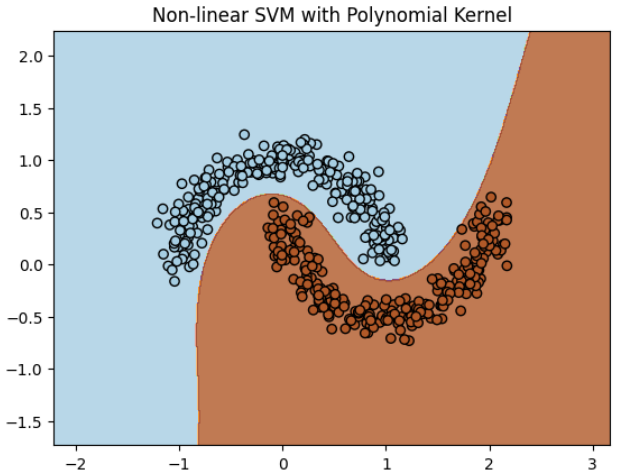
- $\Phi : \mathcal{X} \rightarrow \mathcal{Z}$ transforms $\underline{\boldsymbol{x}}_i$ into $\underline{\boldsymbol{z}}_i = \Phi(\underline{\boldsymbol{x}}_i) \in \mathbb{R}^{\tilde{d}}$ with $\tilde{d} > d$
- Transform vectors through Φ and apply SVM machinery
- Dual SVM formulation in $\mathcal{Z}$ space:

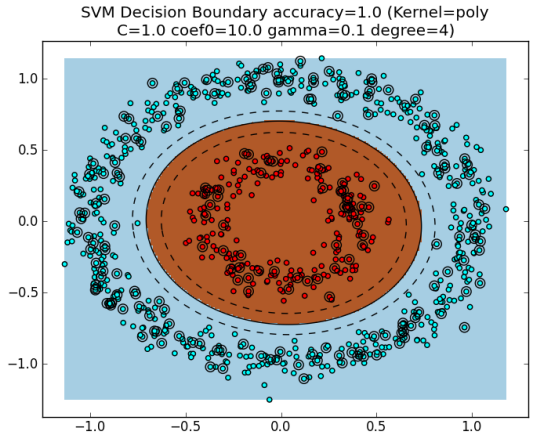
$$\mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \underline{\boldsymbol{z}}_i^T \underline{\boldsymbol{z}}_j$$

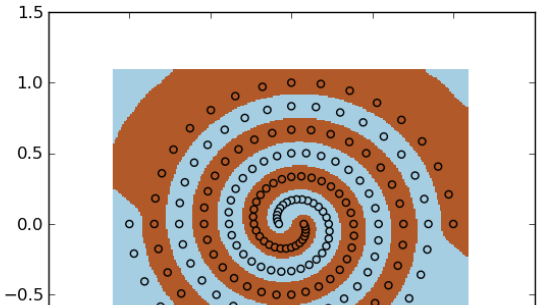
- Note:
  - Optimization problem has same number of unknowns as original space (number of points N)
  - Support vectors live in $\mathcal{Z}$: they have $\alpha = 0$. In $\mathcal{X}$, they are pre-images of support vectors
  - Decision boundary and margin can be represented in original space (not linear)

Non-Linear Transforms for SVM vs Others

- In SVM the non-linear transform does not change the number of unknowns and degrees of freedom of the model
- This is different from transforming the variables in a linear problem, since in that case the number of unknowns changes







SVM in Higher Dimensional Space

- **Pros**
  - You don't pay the price in terms of complexity of optimization problem
    - Number of unknowns is still N (different than a linear problem)
  - You don't pay the price in terms of increased generalization bounds
    - Number of support vectors is $\leq N$
    - This is because each hypothesis h can be complex but the cardinality of the hypothesis set $\mathcal{H}$ is the same
- **Cons**
  - You pay a price to compute $\Phi(\underline{x}_i)^T \Phi(\underline{x}_j)$, since Φ could be very complex
    - The kernel trick will remove this extra complexity by doing $\Phi(\underline{x}_i)^T \Phi(\underline{x}_j) = K_\Phi(\underline{x}_i, \underline{x}_j)$

Non-Linear Transform in SVM vs Kernel Trick

- The trivial approach is:
  - Transform vectors with $\Phi(\cdot)$
  - Apply all SVM machinery to the transformed vectors
  - **Cons**: Φ might be very complex, e.g., potentially exponential number of terms
- You can express the SVM problem formulation and the prediction in terms of a kernel

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = \Phi(\underline{\boldsymbol{x}})^T \Phi(\underline{\boldsymbol{x}}') = \underline{\boldsymbol{z}}^T \underline{\boldsymbol{z}}'$$

  - You only need the kernel $K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ of the transformation $\Phi(\cdot)$ and not $\Phi(\cdot)$ itself

SVM in Terms of Kernel: Optimization Step

- When you build the QP formulation for the Lagrangian to compute the α we can use $K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_j)$ instead of $\underline{\boldsymbol{z}}_i^T \underline{\boldsymbol{z}}_j$

$$\mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m K_\Phi(\underline{\boldsymbol{x}}_n, \underline{\boldsymbol{x}}_m)$$

- $\underline{\boldsymbol{z}}_n$ does not appear in the constraints

$$\underline{\boldsymbol{\alpha}} \geq \underline{\mathbf{0}}, \underline{\boldsymbol{\alpha}}^T \underline{\boldsymbol{y}} = 0$$

SVM in Terms of Kernel: Prediction Step

- You need only inner products to compute a prediction for a given $\underline{\boldsymbol{z}}$
- In fact to make predictions, you replace the expression of $\underline{\tilde{\boldsymbol{w}}} = \sum_{i:\alpha_i>0} \alpha_i y_i \underline{\boldsymbol{z}}_i$ in $h(\underline{\boldsymbol{x}}) = \text{sign}(\underline{\boldsymbol{w}}^T \Phi(\underline{\boldsymbol{x}}) + b)$, yielding:

$$h(\underline{\boldsymbol{x}}) = \text{sign}\Big(\sum_{i:\alpha_i>0} \alpha_i y_i K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}) + b\Big)$$

where b is given by $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{z}}_i + b) = 1$ for any support vector $\underline{\boldsymbol{x}}_m$ and thus

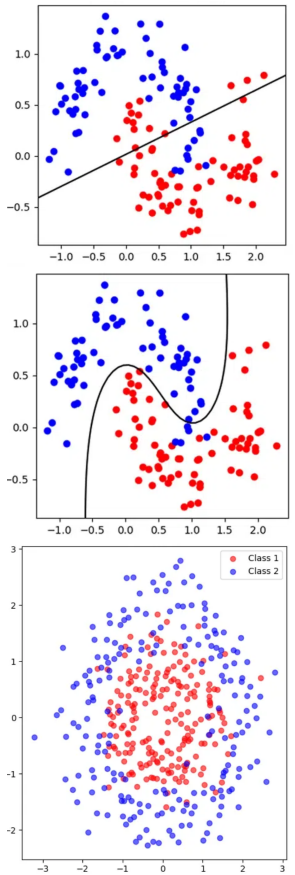
$$b = \frac{1}{y_m} - \sum_{i:\alpha_i>0} \alpha_i y_i K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_m)$$

Implications of Kernel Trick in SVM

- The "kernel trick" is a computational shortcut:
  - Use the kernel of the transformation instead of the transformation itself
- To use SVMs, compute inner products between transformed vectors $\underline{\boldsymbol{z}}$
- The kernel trick implies:
  - No need to compute $\Phi()$
    - Use the kernel K_Φ, not the transformation Φ
  - No need to know Φ
    - With function K_Φ as an inner product, use SVM machinery without knowing $\mathcal{Z}$ space or transformation Φ
  - Φ can be impossible to compute
    - K_Φ can correspond to a transformation Φ to an infinite dimensional space (e.g., Gaussian kernel)

Non-Linearly Separable SVM Problem

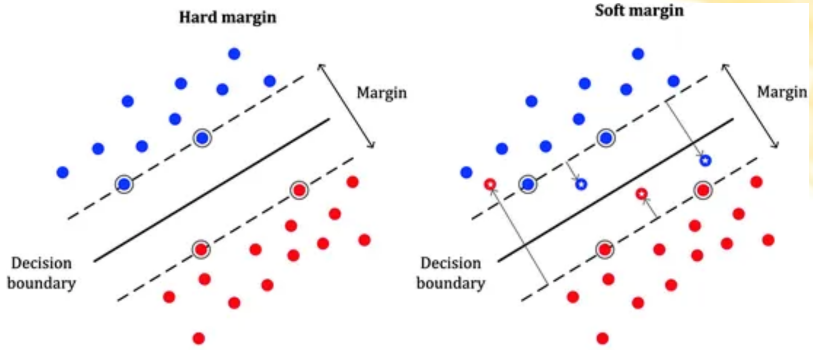
- In general there are different types of **non-linearly separable data sets**
- **Slightly non-separable**
  - Few points crossing the boundary
  - $\Longrightarrow$ use soft margin SVMs
- **Seriously non-separable**
  - E.g., the class inside the circle
  - $\Longrightarrow$ use non-linear kernels

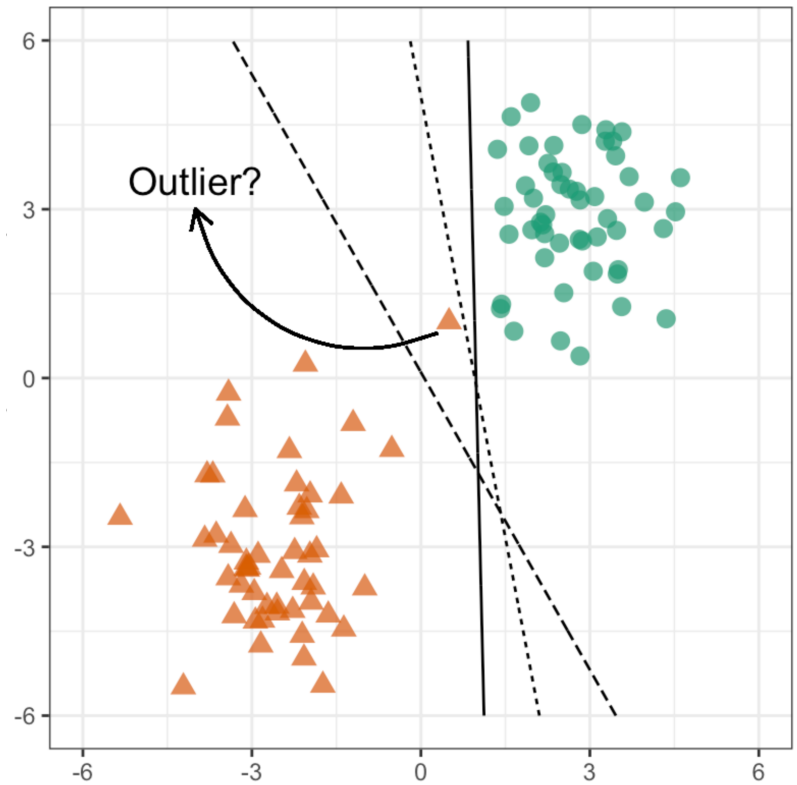


- In practice, both issues are present
  - Combine soft margin SVM and non-linear kernel transforms

Soft-Margin SVM: Advantages

- Even with linearly separable data, improve E_{out} using soft margin SVM at the cost of worse E_{in}
  - Trade-off between in-sample and out-of-sample performance
- E.g., outliers force a smaller margin
  - Ignoring outliers could increase margin
- Large C parameter in SVM requires minimizing error, trading off large margin for correct classification





Primal Formulation for Soft Margin SVM

- You want to introduce an error measure based on the margin violation for each point, so instead of the constraint:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 \text{ (hard margin)}$$

- You can use:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 - \xi_i, \text{ where } \xi_i \geq 0 \text{ (soft margin)}$$

- The cumulative margin violation is $C \sum_{i=1}^{N} \xi_i$
- The soft margin SVM optimization (primal form) is:

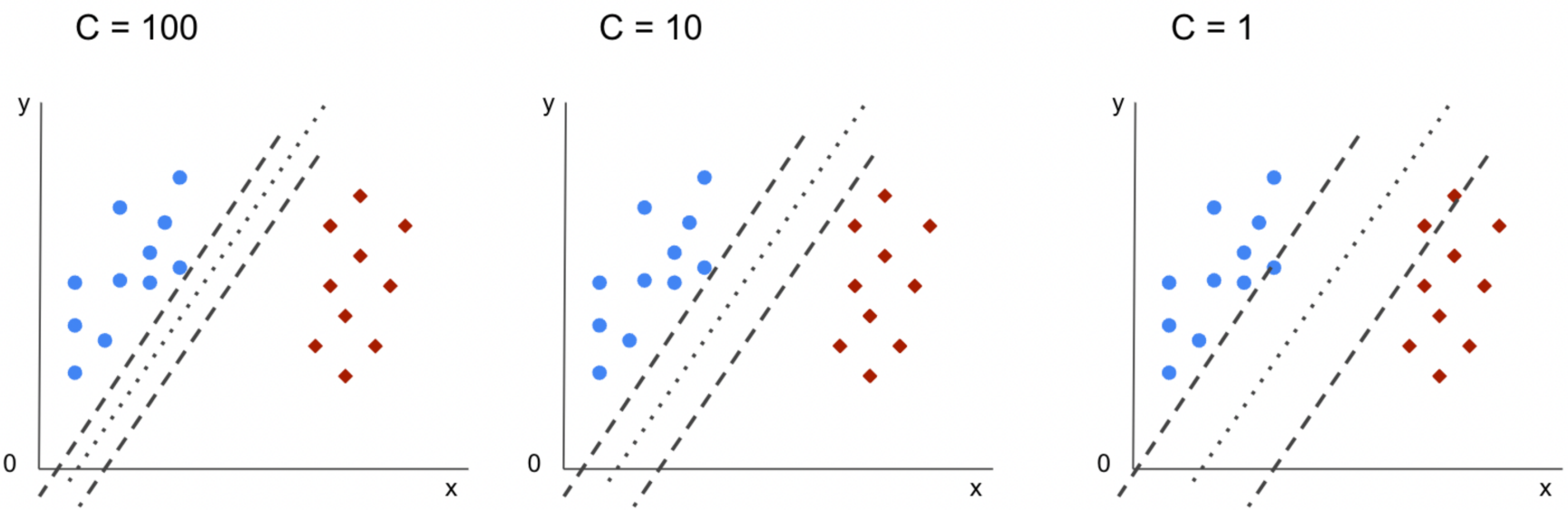
$$\begin{aligned} &\text{find} && \underline{\boldsymbol{w}}, b, \underline{\boldsymbol{\xi}} \\ &\text{minimize} && \frac{1}{2} \underline{\boldsymbol{w}}^T \underline{\boldsymbol{w}} + C \sum_{i=1}^{N} \xi_i \\ &\text{subject to} && y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 - \xi_i \ \forall i \\ & && \xi_i \geq 0 \end{aligned}$$

Classes of Support Vectors for Soft Margin SVM

- There are 3 classes of points:
  - **Margin support vectors**: they are exactly on the margin and define it
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) = 1 \iff \xi_i = 0$
    - In dual form: $0 < \alpha_i < C$
  - **Non-margin support vectors**: they are inside the margin and classified correctly or not
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) < 1 \iff \xi_i > 0$
    - In dual form: $\alpha_i = C$
  - **Non-support vectors**, i.e., interior points:
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) > 1$
    - In dual form: $\alpha_i = 0$

Intuition for C in SVM

- C represents how much penalty you incur for passing the margin
  - If C is large, then SVM will try to fit all the points to avoid being penalized
    - Lower bias / higher variance
    - $C \to \infty$ which yields hard-margin SVM
  - If C is small, then you allow margin violations
    - Higher bias / lower variance
- From another point of view $C \propto \frac{1}{\lambda}$, so large C means small λ and thus small regularization
  - C is chosen through cross validation, like any regularization parameter



Multi-Class Classification for SVM

- Often SVM packages have built-in multi-class classification
- Otherwise use the one-vs-all method:
  - Train K SVMs distinguishing each class from the rest using one-hot encoding
    - Get SVM parameters $(\underline{\boldsymbol{w}}_1, b_1), ..., (\underline{\boldsymbol{w}}_K, b_k)$
  - For a new example $\underline{\boldsymbol{x}}$ compute $\underline{\boldsymbol{w}}_i^T \underline{\boldsymbol{x}} + b_i$ for all the models
  - Pick the model that gives the largest positive value
    - I.e., more confident about its class vs the rest of the classes

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - ***Similarity-Based Models***
  - Clustering
  - Anomaly Detection

Similarity-Based Models: Intuition

- **Idea**: the model evaluated in one point $h(\underline{\boldsymbol{x}})$ is affected by:
  - Other data points in the training set $(\underline{\boldsymbol{x}}_n, y_n) \in \mathcal{D}$
  - The effect is based on the distance $d(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_n) = \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_n\|$
- The model is a **superposition of effects**
  - Sum of the effect of each point in the training set, scaled down by the distance

$$h(\underline{\boldsymbol{x}}) = \sum_i \text{ effect of } h(\underline{\boldsymbol{x}}_i) \text{ scaled by } d(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i)$$

- This approach allows to define complex decision boundaries

Similarity-Based Models: Gaussian Kernels

- Consider a Gaussian kernel with a "landmark" point $\underline{\boldsymbol{x}}_i$ and a similarity distance defined as:

$$K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i) = \exp(-\frac{\|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}{2\sigma^2})$$

- E.g., the hypothesis model has the form:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{3} y_i K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i) = y_1 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_1) + y_2 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_2) + y_3 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_3)$$

  - The response is weighting the responses $y_i = \{0, 1\}$ through the similarity of $\underline{\boldsymbol{x}}$ from the landmark points
  - This can be seen by plotting $h(\underline{\boldsymbol{x}})$ on a plane

Radial Basis Function Model for Regression

- Aka RBF
- The model form for **regression** is:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{N} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2)$$

where:

- If γ is small $\implies$ the exponential falls off slowly, and multiple training points affect a point between them
- If γ is large $\implies$ there are spikes centered in the training points and nothing outside

Radial Basis Function Model for Classification

- For **classification** use a similar approach to "linear regression for classification"
  - Fit a regression model:

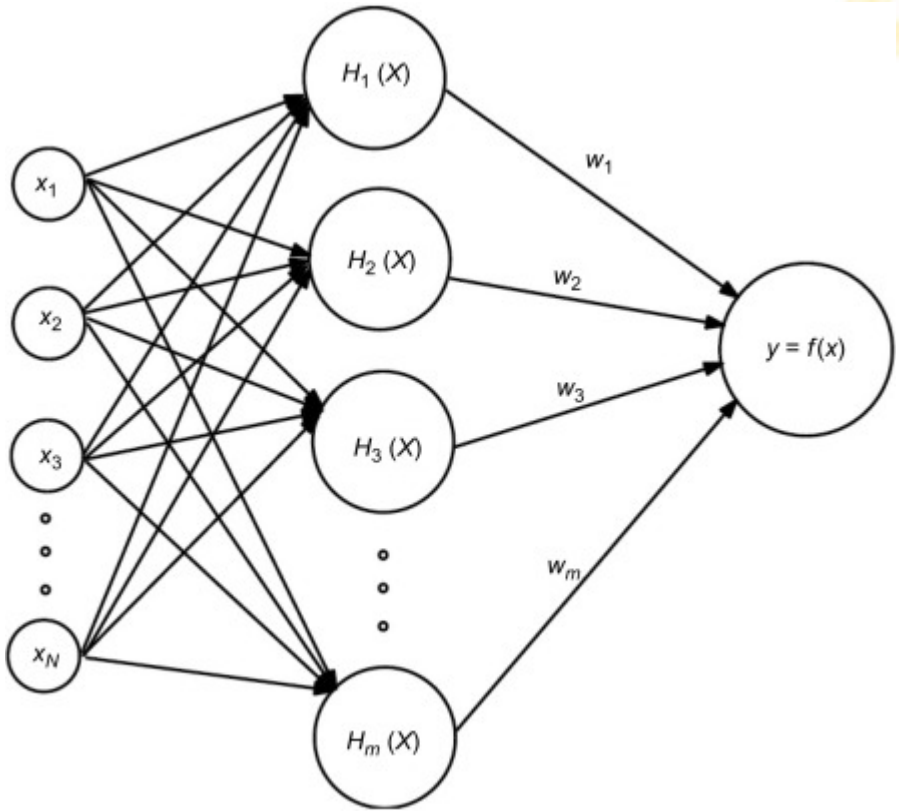
$$s(\underline{\boldsymbol{x}}) = \sum_{i=1}^{N} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2)$$

  - Take the sign to make predictions:

$$h(\underline{\boldsymbol{x}}) = \text{sign}(s(\underline{\boldsymbol{x}}))$$

RBF: Block Diagram

- The params(fixed by learning) are one for each training point
  - The weights depending on the distance of the input to the examples
  - The weighted params are summed together



RBF: Number of Parameters

- Some variants for RBF:
  - Add bias term
  - Use different γ_i for each point
  - Increase degrees of freedom
- RBF has lots of parameters
  - Parameters $\underline{\boldsymbol{w}}$ equal data points N (e.g., $N = 10^9$)
  - One parameter w_i per training point (e.g., $N = 10^6$)
  - **Cons**: Negative impact on generalization error

RBF: Reducing Model VC Dimension

- **To reduce number of parameters**
  - Pick $K \ll N$ centers $\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$ instead of $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N$
    - Use k-means clustering to find centers
    - Unsupervised learning; doesn't use labels
  - Same as RBF model using distances from cluster centers:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{K} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}}_i\|^2)$$

- **Still many parameters** because:
  - K (scalar) weights w_k
  - K reference points $\underline{\boldsymbol{\mu}}_k$ (d-dimensional vectors)

RBF: Learning Models (1/2)

- **Learn** w_i, γ, with fixed centers $\underline{\boldsymbol{\mu}}_i$, for an RBF model:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{K} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}}_i\|^2)$$

- **Minimize:**

$$E_{in} = \sum_i (h_{\underline{\boldsymbol{w}},\gamma}(\underline{\boldsymbol{x}}_i) - y_i)^2 = f(\underline{\boldsymbol{w}}, \gamma)$$

  - Use an iterative approach (e.g., EM, coordinate descent)

RBF: Learning Models (2/2)

- Use iterative approach (similar to EM algorithm):
  - Fix γ, solve for $\underline{\boldsymbol{w}}$ (one-step learning)
  - Fix $\underline{\boldsymbol{w}}$, solve for γ (gradient descent)
- **Step 1**
  - Assume γ is known and fixed
  - Learn $\underline{\boldsymbol{w}}$
- Impose perfect interpolation:

$$E_{in} = \frac{1}{n}\sum(h(\underline{\boldsymbol{x}}_i) - y_i)^2 = 0$$

- **Problem**:

$$h(\underline{\boldsymbol{x}}_j) = \sum_i w_i \exp(-\gamma\|\underline{\boldsymbol{x}}_i - \underline{\boldsymbol{x}}_j\|^2) = \sum_i w_i \phi_{i,j} = \underline{\phi}_j^T \underline{\boldsymbol{w}} = y_i$$

  - N equations (one per point) and N unknowns $\underline{\boldsymbol{w}}$
  - $\underline{\underline{\boldsymbol{\Phi}}}$ is known, function of data set and γ

Learning RBF Models

- The problem in matrix form is:

$$\underline{\underline{\mathbf{\Phi}}} \cdot \underline{\boldsymbol{w}} = \underline{\boldsymbol{y}}$$

  - If $\underline{\underline{\mathbf{\Phi}}}$ is invertible, then $\underline{\boldsymbol{w}} = \underline{\underline{\mathbf{\Phi}}}^{-1}\underline{\boldsymbol{y}}$
    - Desired values on training points
    - Exponential interpolates other points
  - If $\underline{\underline{\mathbf{\Phi}}}$ is not invertible, optimize in least square sense:

$$\text{argmin}_{\underline{\boldsymbol{w}}} E_{in} = \sum_i (h(\underline{\boldsymbol{x}}_j) - y_i)^2$$

    - Compute pseudo-inverse (assuming $\underline{\underline{\mathbf{\Phi}}}^T\underline{\underline{\mathbf{\Phi}}}$ is invertible)
    - Assign weights:

$$\underline{\boldsymbol{w}} = (\underline{\underline{\mathbf{\Phi}}}^T\underline{\underline{\mathbf{\Phi}}})^{-1}\underline{\underline{\mathbf{\Phi}}}^T\underline{\boldsymbol{y}}$$

- **Step 2**
  - Assume $\underline{\boldsymbol{w}}$ is known and fixed
  - Learn γ

RBF Network vs Neural Networks

- The regression model for Neural Networks and RBF model is **similar**:
  - **RBF**:
$$h(\underline{x}) = \sum_i w_i e^{-\gamma \|\underline{x} - \underline{x}_i\|^2} = \underline{w}^T \underline{\phi}$$
  - **Neural networks**:
$$h(\underline{x}) = \Theta(\underline{w}^{(L)T} \underline{x}^{(L)}) = \Theta(\underline{w}^{(L)T} \underline{\Theta}(\underline{\underline{W}}^{(L-1)} ...))$$
- **Difference**:
  - RBF has a single layer
  - Neural networks have multiple layers
- **Similarities**:
  - Combine features with weights using dot product
  - Extract features from inputs
    - RBF features: $e^{-\gamma \|\underline{x} - \underline{x}_i\|^2}$, always > 0
    - NN hidden layers: features can be > 0 or < 0

RBF Network vs SVM

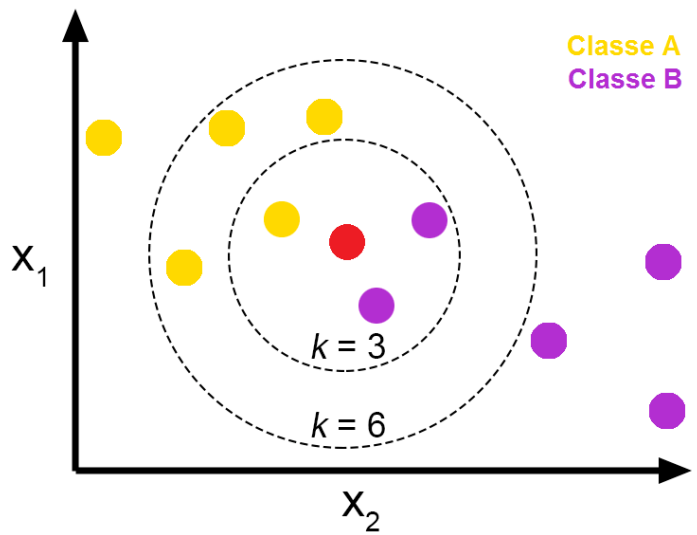
- The **model form** is the same:
  - RBF:
$$h(\underline{\boldsymbol{x}}) = \text{sign}(\sum_i w_i e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}$$
  - SVM:
$$h(\underline{\boldsymbol{x}}) = \text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + b)$$
- The interpretation is completely different (interpolation vs large margin)
  - RBF: all vectors (or centers of few clusters) contribute to the model
  - SVM: only support vectors contribute to the model

K-Nearest Neighbor (KNN) Model

- The **model form** is like:

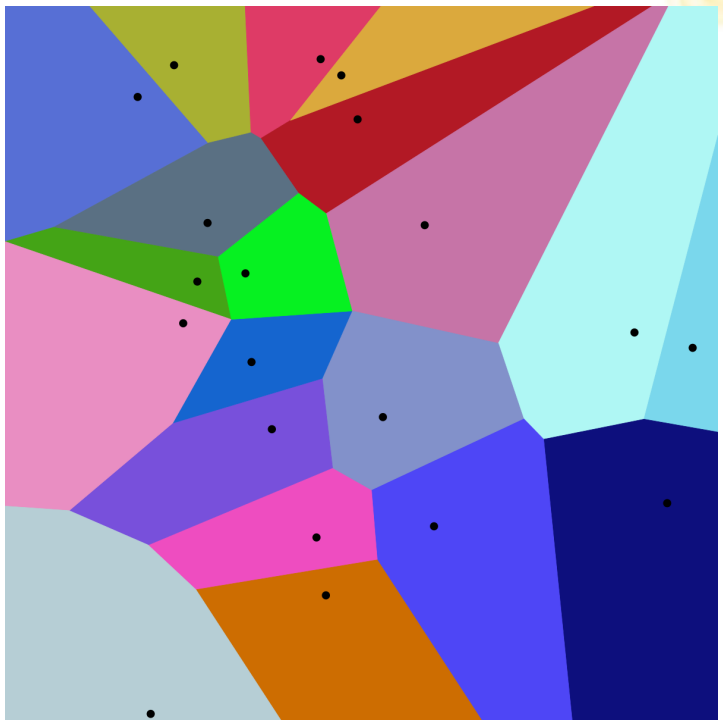
$$h_{\underline{w}}(\underline{x}) = \frac{1}{n} \sum_{\underline{x}_i \text{closest to } \underline{x}} w_i$$

- **Idea**:
  - Closeness implies a distance (e.g., euclidean metric) or similarity (e.g., a kernel)
  - Consider the k closest points to the evaluation point $\underline{x}$
  - Take an average of their response



KNN: Intuition of Number Degrees of Freedom

- Nearest neighbor model ($k = 1$)
  - Use response of closest point to $\underline{\boldsymbol{x}}$
  - Similar to Voronoi tessellations: each point has a region where it is the closest and assigns its output to that region
- k is **the only parameter** for KNN
  - For $k = 1 \implies N$ neighborhoods, one around each training point
  - For $k = N \implies$ single neighborhood
  - Effective number of parameters: $\frac{N}{k}$, imagining N/k non-overlapping neighborhoods



KNN: Assumptions on the Data

- KNN makes **no assumption on data**
  - Opposite of linear model with strong data assumption
- KNN assumes **locality in parameter space**
  - Model is constant in example's neighborhood
  - E.g., $k = 1$: Voronoi tessellation (low-bias/high-variance)
  - E.g., $k = N$: Average value (high-bias/low-variance)

KNN: Training and Test Error

- For $k = 1$
  - No error on training set (low bias / high variance) assuming non-noisy target
  - E_{out} larger than E_{in}
- Increasing k
  - Training error E_{in} increases
  - Test error E_{out} decreases, then increases
  - Typical model complexity behavior in bias-variance diagrams

KNN vs RBF Models

- **Similarities**
  - K-Nearest Neighbor is a *discrete* version of the RBF model
- **Differences:**
  - Consider only the *k closest examples* to the point $\underline{\boldsymbol{x}}$ (not all examples in the training set)
  - Use a *constant kernel* (responses are not weighted by distance)

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - ***Clustering***
  - Anomaly Detection

K-Means Clustering: Problem Formulation

- N *unlabeled* points $\{\underline{\boldsymbol{x}}_1, \underline{\boldsymbol{x}}_2, ..., \underline{\boldsymbol{x}}_N\}$
- Partition points into K clusters $S_1, ..., S_K$
  - Each cluster defined by center $\underline{\boldsymbol{\mu}}_k$
  - Each point $\underline{\boldsymbol{x}}_i$ assigned to cluster $c(\underline{\boldsymbol{x}}_i)$
  - Unknowns are $c(\underline{\boldsymbol{x}}_1), ..., c(\underline{\boldsymbol{x}}_N), \underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$
- Minimize distance between each $\underline{\boldsymbol{x}}_i$ and assigned center $\underline{\boldsymbol{\mu}}_k$ where $k = c(\underline{\boldsymbol{x}}_i)$

$$J(c_1, ..., c_N, \mu_1, ..., \mu_K) = \sum_{k=1}^{K} \sum_{\underline{\boldsymbol{x}}_n \in S_k} \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_k\|^2 \text{(scanning the clusters)}$$

$$= \sum_{i=1}^{N} \|\underline{\boldsymbol{x}}_i - \underline{\boldsymbol{\mu}}_{c(\underline{\boldsymbol{x}}_i)}\|^2 \qquad \text{(scanning the points)}$$

- **K-means clustering is NP-hard** (combinatorial) and thus intractable
  - In fact there are K^N possible assignments

K-Means Clustering: Lloyd's Algorithm

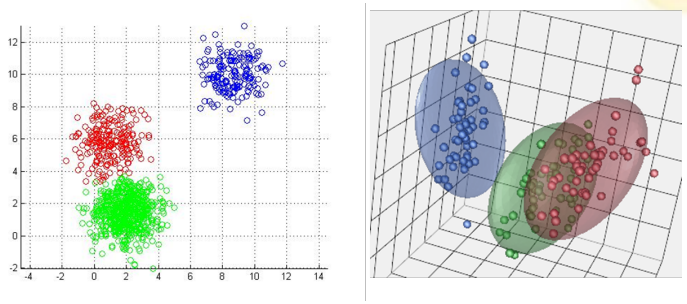
- **Start with a random assignment** of N points to K clusters
  - Better than picking random centroids
- **Each iteration** has 2 steps
  - **Step 1**: Move centroid
    - Move each cluster's centroid to the mean point
    - Iterate over K clusters
    - $\underline{\boldsymbol{\mu}}_k \leftarrow \frac{1}{|S_k|} \sum_{\underline{\boldsymbol{x}}_n \in S_k} \underline{\boldsymbol{x}}_n$
  - **Step 2**: Cluster assignment
    - Assign each $\underline{\boldsymbol{x}}_n$ to the closest cluster center
    - Iterate over N points
    - $S_k \leftarrow \{\underline{\boldsymbol{x}}_n : \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_k\| \leq \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_l\| \; \forall l \neq k\}$

K-Means Clustering: Convergence

- **K-means algorithm converges** since:
  - Finite number of possible partitions (and values of objective functions)
  - Objective function $J(\cdot)$ always decreases
- **Objective function always decreases**
  - Cost function $J(\underline{\mu}_1, ..., \underline{\mu}_K, c_1, ..., c_N)$ depends on:
    - Centroids $c_1, .., c_N$
    - Point assignments $\underline{\mu}_1, ..., \underline{\mu}_K$
  - K-means minimizes J by:
    - Adjusting centroids (fixed assignments)
    - Adjusting assignments (fixed centroids)
  - Similar to coordinate descent
- Generally converges to a **local minimum**
  - Run K-means multiple times with different random initializations
  - Choose best result

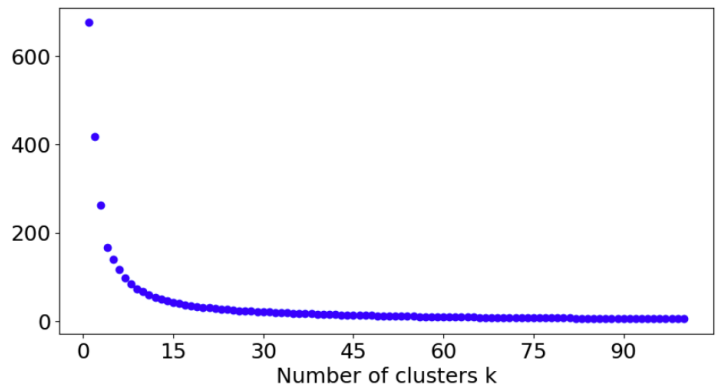
K-Means Clustering: Non-Separable Clusters

- For simplicity, you tend to imagine a clear separation between clusters
  - Clusters, especially in high dimensions, are not obviously separable
- Using K-means on data not obviously separated
  - E.g., market segmentation
  - E.g., t-shirt sizing
    - Collect height and width of customers
    - Run K-means
    - Find optimal way to split population into 3 sizes (S, M, L)



Choosing the Number of Clusters

- Often unclear how many clusters K exist
  - Visual analysis can be inconclusive in 2D or 3D
  - More difficult in high dimensions
- **Elbow Method**
  - Vary clusters K
  - Compute optimal cost function $J(\cdot)$
  - Choose K at "elbow" point if visible
  - Elbow absent if curve resembles hyperbole $\approx 1/K$



- **End-to-end approach**
  - Choose K to optimize later stages
  - E.g., more t-shirt sizes (more clusters) $\Longrightarrow$
    - Satisfy customers
    - Complicates manufacturing
    - Increases inventory management

Interpretation of Clusters

- Cluster meaning **often interpreted manually** (difficult to automate)
  - Examine cluster centroids
    - Centroid values show "typical" point in each cluster
    - High, low, or zero feature values highlight key characteristics
  - Analyze distribution of features per cluster
    - Plot histograms or boxplots for each feature
    - Identify features that vary sharply across clusters
  - Visualize clusters in 2D or 3D
    - E.g., PCA, t-SNE, UMAP
    - Understand separation and internal structure
  - Compare clusters to external labels if available
    - See if clusters align with known real-world groups
  - Train a classifier like decision tree
    - Important features for predicting cluster reveal their meaning
- **Example: Customer Segmentation**

Cluster	Age	Annual Income	Spending Score	Label
Cluster 1	25 yrs	30K	90	Young Big Spenders
Cluster 2	50 yrs	80K	40	Comfortable Mid-Lifers
Cluster 3	35 yrs	120K	20	High Income, Low Spending Customers

- Models
  - Naive Bayes Model
  - Decision Trees
  - Random Forests
  - Linear Models
  - Perceptron
  - Logistic Regression
  - LDA, QDA
  - Kernel Methods
  - Support Vector Machines (Optional)
  - Similarity-Based Models
  - Clustering
  - ***Anomaly Detection***

Anomaly Detection: Problem Formulation

- **Problem**:
  - $\{\underline{x}_1, ..., \underline{x}_N\}$ examples with features $\underline{x} \in \mathbb{R}^P$ for good instances
  - Detect bad/anomalous instances
- **Algorithm**:
  - Unknown characteristics of *"bad instances"*
  - Learn common traits of *"good instances"* using unsupervised learning
    - Find distribution for $\underline{x}_i$: Pr($\underline{x}$ is good)
  - Pick features
    - Find "sensitive" features, e.g., ratio between CPU load and network traffic
  - Estimate distribution Pr($\underline{x}$ is good)
  - Choose threshold ε
  - For new instance $\underline{x}_{new}$, if Pr($\underline{x}_{new}$ is good) $\leq \varepsilon$ flag as anomaly

Anomaly Detection: Example of Aircraft Engines

- **Problem**
  - Test aircraft engines to identify anomalies in a new engine
- **Solution**:
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - Heat generated
    - Vibration intensity
    - ...
  - Collect data for all engines
  - Model $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Decide if a new engine is acceptable $\Pr(\underline{\boldsymbol{x}}_{new}$ is good$) \leq \varepsilon$ or needs more testing

Anomaly Detection: Example of Hacked Account

- **Problem**
  - Find if an account for a given user i was hacked
- **Solution**:
  - Model features that represent "user i activity"
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - How many times s/he logs a day
    - How many times s/he fails to enter the password
    - How fast s/he types
    - How many pages s/he visits
    - How many times s/he posts comments
    - ...
  - Model $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Identify unusual users by checking $\Pr(\underline{\boldsymbol{x}}_{new}$ is good$) \leq \varepsilon$

Anomaly Detection: Example of Data Center

- **Problem**
  - Monitor servers in a data center to find malfunctioning or hanged servers
- **Solution**:
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - Memory in use
    - CPU load
    - Network traffic
    - Number of reads/writes per sec
    - CPU load / network activity
    - ...
  - Model Pr($\underline{\boldsymbol{x}}$ is good)
  - Identify unusual systems by checking Pr($\underline{\boldsymbol{x}}_{new}$ is good) $\leq \varepsilon$

Using a Gaussian Model for Anomaly Detection

- Aka "density estimation"
- Given N examples $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N \in \mathbb{R}^p$
- Ensure that the **features have a Gaussian distribution**
  - If not, apply some transformations, e.g., $\log(x_i + k)$
- **Estimate the parameters** of the Gaussian model $f_X(\underline{\boldsymbol{x}})$
- Given a new example $\underline{\boldsymbol{x}}_{new}$ **compute** $\Pr(\underline{\boldsymbol{x}}_{new}$ is good$) \leq \varepsilon$ to flag an anomaly

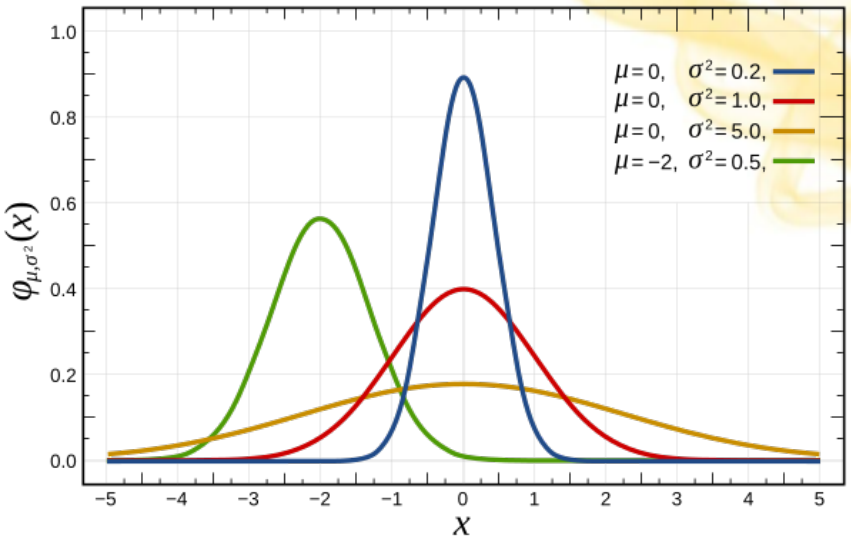
Estimate Univariate Gaussian Model

- You have N (scalar) examples $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N \in \mathbb{R}$ representing *"good instances"*
- Assume the data is generated by a Gaussian distribution

$$X \sim \mathcal{N}(\mu, \sigma)$$

which has a PDF:

$$f_X(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$



- Estimate mean and sigma with maximum likelihood:

$$\mu = \frac{1}{N}\sum_i x_i$$

$$\sigma^2 = \frac{1}{N-1}\sum_i (x_i - \mu)^2$$

Estimate Multivariate Independent Gaussian Model

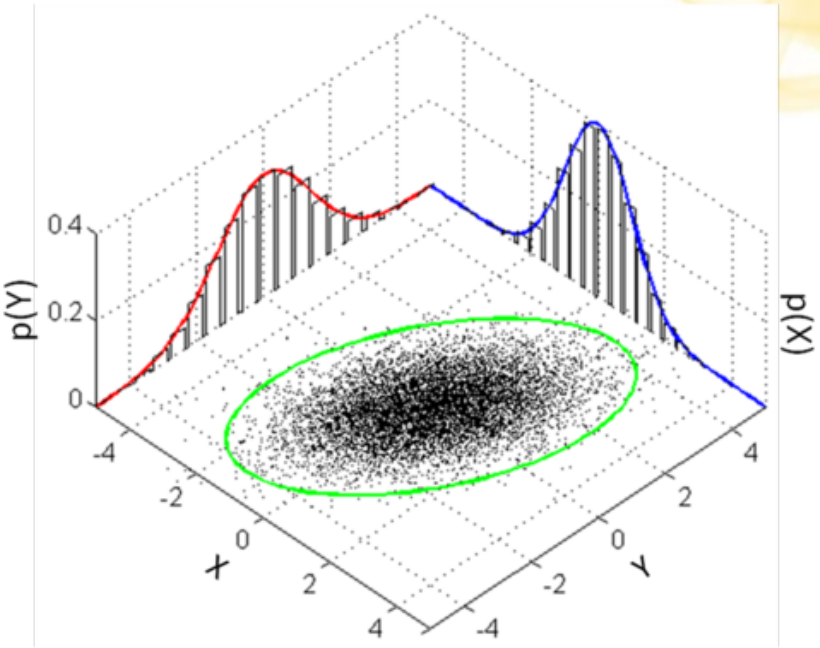
- You have N examples $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N \in \mathbb{R}^p$ for *"good instances"*
- Assume independence of the features, the PDF of a multi-variate Gaussian X is:

$$f_X(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu}}, \underline{\boldsymbol{\sigma}}) = \prod_{i=1}^{p} f_{X_i}(x_i; \mu_i, \sigma_i)$$

- Infer the parameters μ_i and σ_i using discrete formulas to get the complete model

Estimate a Multi-Variate Gaussian Model

- **Problem**:
  - Often features vary together (e.g., network use and CPU load), causing mis-classifications with independent assumptions
  - Components of $\underline{\boldsymbol{x}}_{new}$ are within range but nonsensical together
    - E.g., low network use with high CPU load
- **Solution 1**:
  - Engineer features to highlight anomalies
  - Address variable correlation not modeled in independent Gaussian models
- **Solution 2**:
  - Estimate the entire multivariate model instead of assuming independence



Estimate a Multi-Variate Gaussian Model

- The PDF of a multi-variate Gaussian is:

$$f_X(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu}}, \underline{\underline{\boldsymbol{\Sigma}}}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\underline{\underline{\boldsymbol{\Sigma}}}|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}})^T \underline{\underline{\boldsymbol{\Sigma}}}^{-1}(\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}}))$$

- Estimate:

$$\underline{\boldsymbol{\mu}} \in \mathbb{R}^d = \frac{1}{N} \sum_{k=1}^{N} \underline{\boldsymbol{x}}_k$$

$$\underline{\underline{\boldsymbol{\Sigma}}} \in \mathbb{R}^{d \times d} = \{s_{ij}\} = \frac{1}{N-1} \sum_{k=1}^{N} (\underline{\boldsymbol{x}}_k - \underline{\boldsymbol{\mu}})(\underline{\boldsymbol{x}}_k - \underline{\boldsymbol{\mu}})^T$$

- Model requires more examples to train due to more parameters
- Independence between variables decomposes multivariate Gaussian into product of Gaussian distributions

Evaluate Anomaly Detection Systems

- To evaluate models one needs to:
  - Compare different models
  - Tune hyperparameters (e.g., ε) of models
  - Estimate out-of-sample error
- As always we should use a single real number for comparison
  - Use any classification metric, e.g.,
    - True/false positive/negative rate
    - Precision or recall
  - F-score
- *Labeled* data is still needed to rate models

$$y = 0 \text{ good}$$
$$y = 1 \text{ anomalous}$$

Evaluate Anomaly Detection Systems

- Often, anomalous examples $y = 1$ are much fewer than good examples $y = 0$
  - E.g., 10,000 good vs 20 bad examples
  - Address class imbalance for accurate model performance
- **Algorithm**:
  - Pick 60% of $y = 0$ data to train (only on good examples)
  - Split remaining $y = 0$ and $y = 1$ into validation and test sets
    - Ensure both sets represent the overall dataset
    - Train, validation, and test sets should not overlap but have the same characteristics
    - Helps evaluate model performance accurately
  - Use validation set to compare models, estimate hyperparameters
    - E.g., ε is the threshold for anomaly detection
  - Use test set to evaluate final model
    - Train on normal data, test on both normal and anomalous data
- **Aircraft engine example**

Dataset	$y = 0$ (Good Engines)	$y = 1$ (Bad Engines)
Total Example	10,000	20
Train Set	6,000	0
Validation Set	2,000	10
Test Set	2,000	10

Anomaly Detection vs Supervised Learning

- Even in unsupervised learning, you need labeled data for model evaluation
- Difference with supervised learning:
  - Anomaly detection/unsupervised learning: Train only on good examples
  - Supervised learning: Train on both good and bad examples
- Use:
  - Anomaly detection/unsupervised learning:
    - Learn from good examples due to few anomalous examples
    - Strong prior on the model
    - Future anomalous examples unknown
  - Supervised learning: Less skewed classes in training set
  - Not a clear-cut decision