MSML610: Advanced Machine Learning

Machine Learning Models

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

References:

- ***Models***
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

- Models
  - ***Naive Bayes***
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Naive Bayes

- We need to predict one of the classes $H_1, ..., H_n$ using the vector of evidence $\underline{\boldsymbol{E}}$:
  - Use Bayes' rule of conditional probability to decide which class $H_1, ...H_n$ to pick given the 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 the evidence $\underline{\boldsymbol{E}}$ is the vector of features
  - The training data provides an estimate of joint probability $\Pr(H_i, \underline{\boldsymbol{E}})$
- **Naive Bayes 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 whether kids are going to play outside or not using past observations of weather condition
- **ML formulation**
  - Supervised learning set-up
  - Predictor vars are:
    - `outlook = {sunny, overcast, rainy}`
    - `temperature = {hot, mild, cold}`
    - `humidity = {high, normal}`
    - `windy = {true, false}`
  - Response var is:
    - `play = {yes, no}`
  - Training set:
    - We have samples for predictors and response from observations
    - There might be noise in the data
    - E.g., different kids have different preference about weather, one kid is sick or has lots of 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

- We use Bayes' rule of conditional probability to decide which class H_j to pick:

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

  where:
  - H_j is one of the events we want to predict
    - E.g., `play = yes`
  - $\underline{\boldsymbol{E}}$ is the event corresponding to a combination of 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
  - $\Pr(H_j)$: probability of the outcome before evidence $\underline{\boldsymbol{E}}$ ("prior probability")
    - E.g., $\Pr(\text{play} = \text{yes})$
    - Computed from the 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 the 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):

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

- Bayes theorem shows that 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
  - To evaluate the model choose the most frequent output given the current value of the feature
- In the weather problem:
  - Assume you pick `outlook` as single feature
  - We know the value of the predictor vars, e.g., `outlook = sunny`
  - Compute the probability of `play = yes` given the `outlook = sunny` using the training set:

$$\Pr(\texttt{play = yes} | \texttt{outlook = sunny})$$

  - Output the predicted var

Naive Bayes: why independence assumption is useful

- The independence assumption allows to factor out the 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 the computation of the probability
  - Helps with generalization, since there is an underlying model (independence) and one needs fewer samples to fit it
- Cons:
  - Not necessarily true assumption

Intuition of estimating probabilities

- Maximum likelihood estimate (MLE) estimates the probability of an event by counting the occurrences of that event among all the 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:

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

- **Problem**
  - A particular value e' of a feature E_i does not occur in the training set in conjunction with an output class H_j
  - The estimated probability $\Pr(E_i = e'|H_j) = 0$
  - Then plugging $\Pr(E_i = e'|H_j) = 0$ into

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

  - It yields $\Pr(H_j|\underline{\boldsymbol{E}}) = 0$
  - It is impossible to decide

Laplace estimator for probabilities

- To address events not in the training set, we can use the Laplace estimator for conditional probabilities instead of MLE
- The maximum likelihood estimate (MLE) is:

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

- The Laplace estimator
  - Adds 1 to each count and V (number of feature values) to the denominator to normalize the probability to 1:

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

  - Blends a prior that feature values are equiprobable with the estimated probabilities using MLE

- Models
  - Naive Bayes
  - ***Decision trees***
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Decision tree

- Characteristics:
  - Supervised learning method
  - For both classification and regression
  - Non-parametric (i.e., no functional form)
- Model
  - A set of decision rules organized in a tree
- Training
  - Infer a set of decision rules from data set
  - Worst case complexity is $O(2^n)$ with number of variables
  - Greedy divide-and-conquer make the average complexity better
- Evaluation
  - The model is evaluated from root to leaves, similar to a flow chart
  - Cost of prediction is $O(log(n))$ with numbers of training points

Typical decision trees

- Each node tests a particular feature against a constant (i.e., splitting the input space with hyperplanes parallel to the axes)
- Each features
  - Is tested only once
  - Is labeled with $(x^{(j)}, <, =, >, t)$
  - Checks a feature $x^{(j)}$ against a threshold t
  - Depending on the result we follow one or the other branch
- There are no re-converging paths: it is a tree!
- The leaves at the bottom of the tree represent decision in terms of:
  - Class labels (e.g., classification)
    - Can predict class or its probability
  - Regression function
- The deeper the tree, the more complex the decision rules, and fitter the model
- Trees are non-linear models since they use interaction of variables
  - Each predicted class can be expressed in a OR-AND form:

Decision trees: pros

1. Can be used for both regression and classification
2. Simple to understand and interpret
    - White box model: explanation of the decision can be reported (vs black box model like neural network)
    - Can be visualized
    - Can be created by hand
3. Requires little data preparation
    - Invariant under feature scaling, e.g.,
        - No data normalization
        - No dummy variables
    - Handles both numerical and categorical data simultaneously
    - Robust to irrelevant features (e.g., feature selection is implicit)
    - Handles NAs
4. Scalable
    - Performs well with large datasets

Decision trees: cons

1. Learning a decision tree is NP-complete
   - Worse than complexity of OLS
   - Algorithms use heuristics (e.g., greedy algorithms)
2. Risk of overfitting
   - Solutions:
     - Pruning
     - Minimum samples at a leaf node
     - Max depth of trees
3. Some training sets are hard to learn
   - E.g., XORs, parity
4. Unstable
   - Small data variations can 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 the outcomes into two groups ("leaves") remaining items into two leaves
    - Best = samples with same labels are grouped together
  - Continue splitting until:
    - Groups are small enough
    - Maximum depth is reached
    - Sufficiently "pure"

Mathematical formulation of splitting at one node

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

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

- The impurity of a split is defined as:

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

- The goal is to find the split $(j, t_m)^*$ that minimizes $H(j, t_m)$
- Then we recurse on $Q_L(j, t_m)^*$ and $Q_R(j, t_m)^*$

Measures of node impurity for classification

- **Measures of node impurity:**
  - Are 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
  - E.g., less impurity means smaller probability of misclassification
- Examples of measures of node impurity
  1. Misclassification error
  2. Gini impurity index
  3. Information gain / deviance

Probability of classification in a node

- Assume that in the m-th node there are N_m objects x_i that belong to different K classes
- Compute the probability of each class in the 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

- We have several class probabilities p and need a single probability
  - Consider the worst case, i.e., the most common class k' in the node
- The misclassification error is defined as:

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

- For binary classifier
  - Best case (perfect purity)
    - Only one class in the node
    - $H_M(p) = 0$
  - Worst case (perfect impurity)
    - There is a 50-50 split between classes in the node
  - $H_M(p) = 0.5$
- For multi-class classifier with K classes
  - The misclassification error has the upper bound of $1/K$

Gini impurity index: definition

- Given the distribution of elements p in the m-th node, the Gini index $H_G(p)$ is the probability of picking an element randomly and classify it incorrectly
- By using the law of total probability:

$$\begin{aligned} H_G(p) &= \Pr(\text{pick elem of } k\text{-th class}) \cdot \Pr(\text{misclassification} \mid \text{elem of } k \text{ class}) \\ &= \sum_{k=1}^{K} p_k \cdot (1 - p_k) \end{aligned}$$

- For binary classifier
  - $H_G(p)$ is between 0 (perfect purity) and 0.5 (perfect impurity)
- For multi-class classifier with K classes
  - $H_G(p)$ has upper bound of $1 - K\frac{1}{K}^2 = 1 - \frac{1}{K}$

Information gain: definition

- Aka cross-entropy (remember entropy is $-p \log p$)
- Information gain entropy is defined as:

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

- For binary classifier
  - H_{IG} varies between 0 (perfect purity) and 1 (perfect impurity)
- For multi-class classifier with K classes
  - H_{IG} has upper bound of $\log K$

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 given N_m observations at a certain node, one can minimize the 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 tend to overfit on data with many features
  - Need to get the right ratio of training samples to features
  - Consider dimensionality reduction (PCA, feature selection) to remove non-discriminative features
- Use maximum tree depth to prevent overfitting
- Control number of examples at a leaf node with min number of samples in a leaf or in a split
  - Small number means overfit
  - Large number means no learning
- Balance dataset before training to avoid bias towards dominant classes
  - E.g., normalize sum of sample weights for each class

Feature selection with trees

- Intuition:
  - Features at the top of the tree contribute to predicting a larger fraction of input samples
- Importance of a variable
  - The fraction of training samples a feature "controls" estimates the feature's relative importance
  - The depth of a feature as a decision node in a tree assesses its relative importance
- Since trees are unstable, reduce estimation variance by averaging the depth of a variable over several randomized trees (random forest)

Embeddings with trees

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

- Models
  - Naive Bayes
  - Decision trees
  - ***Random forests***
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

From decision trees to random forests

- **Decision trees have:**
  - Low bias
  - High variance (i.e., high capacity models)
- Apply ensemble methods to trees $\rightarrow$ "random forests"
- Bagging:
  - Is effective with "unstable" non-linear models to reduces variance
    - Learning trees is unstable
  - Is best with complex models (e.g., fully grown trees)
    - Vs boosting methods work best with weak models (e.g., shallow decision trees, aka tree stumps)
  - Works for both regression and classification
  - Can be customized for trees
    - Different types of randomization in trees
- Bias-variance trade-off in random forests
  - The bias of the forest could increase compared to a single non-random tree
  - The variance of the forest is reduced by averaging, usually compensating for the increase in bias

Randomization in trees

- Perturb-and-combine techniques specifically designed for trees
- One can bag (i.e., bootstrap aggregate)
  1. Training samples
     - E.g., with / without replacement
  2. Picking features (aka random subspaces)
  3. Decision split thresholds
  4. All the above
- Random forests
  - Each tree in the ensemble is built from samples drawn with replacement from the training set (aka bootstrap sample)
  - The split is picked as best among a random subset of the features
- Extremely randomized trees (aka "Extra-Trees")
  - Also the thresholds are randomized
  - More randomness with respect to random forests
  - The result is trading off even more bias for variance
- Combine random forests for classification:
  - Using average / majority vote on class

Random forests: pros and cons

- The 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
  - Decision trees
  - Random forests
  - ***Linear models***
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Linear regression model

- Data set is $(\underline{\boldsymbol{x}}_1, y_1), ..., (\underline{\boldsymbol{x}}_N, y_N)$
  - There are N examples
  - There are P features, i.e., $\underline{\boldsymbol{x}}_i \in \mathbb{R}^P$
  - In regression problems the output of model is a real-value number: $y = h(\underline{\boldsymbol{x}}) \in \mathbb{R}$
- Linear regression model form is:

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

- You can add a bias term w_0 by including an additional component $x_0 = 1$ to the data and the model

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

In sample error for linear regression

- For **regression**, we use the squared error for the in-sample error of hypothesis h:

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

- For **linear regression:**

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

- 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)

Find optimal model for linear regression

- We 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 because it is a generalization of 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 sometimes called "one-step learning"
  - In contrast with iterative methods, e.g., gradient descent
- We need to invert a square matrix of size P, which is related to the number of parameters and not the number of examples N
  - The complexity of one-step learning is $O(P^3)$

Linear models are linear in what?

- A model is linear when the signal $s = \sum_{i=0}^{P} w_i x_i = \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}$ used to make decision is linear with its variables
  - The (unknown) variables are the weights w_i
  - The inputs x_i are fixed
- If you apply a non-linear transform to the inputs $z_i = \Phi(x_i)$, the model

$$s = \sum_{i=0}^{P} w_i \Phi(x_i) = \underline{\boldsymbol{w}}^T \Phi(\underline{\boldsymbol{x}})$$

  is still linear
  - Positive / negative part (e.g., $z_i = RELU(x_i), RELU(-x_i)$)
  - Waterfall (i.e., conditioning the model to different range of features)
  - Thresholding (e.g., $z_i = \min(x_i, T)$)
  - Indicator variables (e.g., $z_i = I(x_i > 0)$)
  - Winsorizing (i.e., replace outliers with a large but constant value)
  - ...
- If you apply a non-linear transform to the weights $z_i = \Phi(w_i)$, the model

$$s = \Phi(\underline{\boldsymbol{w}})^T \underline{\boldsymbol{x}}$$

  is not linear

Non-linear transformations with linear models

- Transform variables:
  - Use a $\Phi : \mathcal{X} \to \mathcal{Z}$
  - Transform each point $\underline{\boldsymbol{x}}_n \in \mathcal{X} = \mathbb{R}^d$ into a point in the 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 a linear model in the $\mathcal{Z}$ space, obtaining $\underline{\tilde{\boldsymbol{w}}}$ for a separating hyperplane in $\mathcal{Z}$
- Predict:
  - Evaluate the model on a new point in the $\mathcal{Z}$ space with:

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

- Compute the decision boundary:
  - In the $\mathcal{X}$ space if Φ is invertible; or
  - By brute force classifying any point $\underline{\boldsymbol{x}} \in \mathcal{X}$ by going to the $\mathcal{Z}$ space

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - ***Perceptron***
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Example of classification problems

- Binary classification problem
  - $y \in \{0, 1\}$
    - Encoding is arbitrary, although we tend to assign 1 to what we want to detect
  - Email: spam / not spam
  - Online transaction fraudulent: yes / no
  - 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

- You can use linear regression for classification
  - Transform inputs $+1, -1 \in \mathbb{R}$
  - 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 (aka perceptron)
- This is not optimal since outliers can influence the fit due to the square distance metric
  - Use the weights from linear regressions to initialize a learning algorithm for classification (e.g., PLA)

Perceptron learning algorithm (PLA)

- First learning algorithm discovered
- **Algorithm**
  - Initialize weights $\underline{\boldsymbol{w}}$
    - Random values
    - Use linear classification value as seed
  - Pick a misclassified point $\text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i) \neq y_i$ from the training set $\mathcal{D} = \{(\underline{\boldsymbol{x}}_1, y_1), ..., (\underline{\boldsymbol{x}}_n, y_n)\}$
  - 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
- The algorithm is guaranteed to converge (slowly) for linearly separable data
- **Pocket version of PLA**
  - Idea
    - Continuously update the solution
    - Keep the best solution "in the back pocket"
  - We have a solution if we stop after a max number of iterations

Using non-linear transformations for classifications

- In practice, a classification problem can have various degrees of non-linear boundary:
  1. Data clustered in a way that is not linearly separable
     - E.g., with $+$ in the center and $-$ in the corner in a scatter plot of 2 features
  2. Mostly linearly separable classes with few outliers to capture
  3. A higher-order decision boundary
     - E.g., quadratic
  4. A non-linear relationship between data and features
     - E.g., threshold for a variable

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - ***Logistic regression***
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Logistic regression is a probabilistic classifier

- In logistic regression, we learn $\Pr(y|\underline{\boldsymbol{x}})$, i.e., the probability of each class given the input, instead of predicting the class directly
- **Parametric approach:**
  - Assume $\Pr(y|\underline{\boldsymbol{x}}; \underline{\boldsymbol{w}})$ has a known functional form

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

  - Learn by optimizing parameters $\underline{\boldsymbol{w}}$ using maximum likelihood

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

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

Logistic function

- Aka "sigmoid"
- Logistic function $\text{logit}(s)$ is defined as

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

- $\text{logit}(s)$
  - 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

- The functional form is the same as the linear classifier
  - Logistic regression: $h(\underline{\boldsymbol{w}}) = \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}})$
  - Linear classifier (or perceptron): $h(\underline{\boldsymbol{w}}) = \text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}})$
- The difference is in the probabilistic interpretation and the way it is fit
- In logistic regression, we don't have samples of the probability function to interpolate
  - Rather we have realizations of the random variable
  - We look for parameters of a given model that maximize the likelihood of the data
- In linear classification we assume that the class value is a linear function of the inputs

Error for probabilistic binary classifiers

- For probabilistic binary classification we can use the log-probability error as point-wise error
- Log-probability is defined as:

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

  - We need to negate since we want positive errors and $\log([0, 1]) \in [-\infty, 0)$
- Log probability is a generalization of the 0-1 error
  - Consider the case $y = 1$
    - If we always output a value $h(\underline{\boldsymbol{x}})$ close to 1 $\implies$ the probability is 1 $\implies$ the log is 0 $\implies$ $e(\cdot) = 0$
    - If we always output a value close to 0 $\implies$ $e() = -\log(0) \to +\infty$
  - Analogous behavior is obtained in the case of $y = 0$

One-liner error for probabilistic binary classifiers

- The point-wise error for one 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: $y = x \cdot a + (1 - x) \cdot b$
- Thus we can write the point-wise error (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

- For a binary classifier the we can write the point-wise error as:

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

- We can make further simplification in the case of logit function

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

- The point-wise error for a logistic regression is equal to the cross-entropy error

Cross-entropy error

- The point-wise error for logistic regression:

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

  is called "cross-entropy error"
  - Note: it does not have $-$ before the $\log(\cdot)$ but before $y \cdot \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}$
- Cross-entropy error is a generalization of 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 very large $\rightarrow$ the error goes to 0
  - If they disagree in sign $\rightarrow$ the error goes towards ∞
- As usual we define the in-sample error on the training set as the average of the point-wise errors:

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

Fitting logistic regression

- A plausible error measure of an hypothesis is based on likelihood of the data $\Pr(\mathcal{D}|h = f)$
  - "How likely is the data $\mathcal{D}$ we have under the assumption that $h = f$?"
  - "How likely is that the data $\mathcal{D}$ was generated by h?"
- Maximizing the likelihood that $\mathcal{D}$ is generated from a logistic regression $\Pr(y = 1|\underline{\boldsymbol{x}}; \underline{\boldsymbol{w}})$ is equivalent to minimizing in-sample error on the training set using cross-entropy error

Fitting logistic regression (opt)

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

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

- The model form is:

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

- Assuming independence among training examples

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

- Note that we can fold y_n in the expression since:
  - When $y_n = 1$ then $\Pr(y_n = y_n') = \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$
  - When $y_n = -1$ then $\Pr(y_n = y_n') = 1 - \text{logit}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n) = \text{logit}(-\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$
- Thus in both cases we can write $\Pr(y_n = y_n') = \text{logit}(y_n \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n)$

Fitting logistic regression (opt)

- We have

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

- We can re-write this optimization problem similar to minimizing the sum of the point-wise errors $E_{in} = \sum e(h(\underline{\boldsymbol{x}}_n), y_n)$
  - We can maximize with respect to $\underline{\boldsymbol{w}}$ the log(...) since the log argument is always > 0 and log() is monotone
- Equivalently, we can 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
- Assume we have n classes $c_1, ..., c_n$ 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 probabilistic classification

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

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

- In the innermost summation we consider the error on each class / digit
  - E.g., for $k = 4$ 1000 vs 0100
  - The digits that are equal don't contribute to the error
  - The different digits give a positive contribution

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - ***LDA, QDA***
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Basic idea of parametric models

- We assume a model is generating the data
  - The functional form of the model is known
  - The model is parametrized with unknown parameters to estimate
- **Pros**
  - Utilize structure in the data
  - Models are easy to fit: few parameters to estimate
  - Accurate predictions if model assumptions are correct
- **Cons**
  - Strong assumptions about the data
  - Low accuracy if model assumptions are 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 that the class generating process $f_k(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\mu_k}}, \underline{\underline{\boldsymbol{\Sigma_k}}})$ is from a multivariate Gaussian
- Linear discriminant analysis has a model:

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

  where:

  - Means $\underline{\boldsymbol{\mu_k}}$ are different for all k classes
  - The covariance matrix $\underline{\underline{\boldsymbol{\Sigma}}}$ is the same for all k classes
  - It can be proven that the classes are separated by linear decision boundaries

- Quadratic discriminant analysis has a model:

  - Classes k have different covariance matrix $\underline{\underline{\boldsymbol{\Sigma_k}}}$
  - It can be proved that the classes are separated by quadratic boundaries

Bayes theorem for LDA / QDA

- Consider a classification setup with multi-class output $Y \in \{1, ..., K\}$
- We need to 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 the probability of $\underline{\boldsymbol{x}}$
  - $\Pr(Y = k) = \pi_k$: estimate the probability of each class (prior)
  - $\Pr(X = \underline{\boldsymbol{x}})$: estimate the probability of each input

- Estimate probabilities from data

LDA / QDA: Boundary Decision (*)

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

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

- Using the model assumption and Bayes theorem:

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

where:

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

- We can apply a $\log(\cdot)$ since it is a monotone transformation

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

LDA / QDA: Boundary Decision (*)

- We can apply a $\log(\cdot)$ since it is a monotone transformation

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

- The second term does not depend on $\underline{\boldsymbol{x}}$
- The first term is proportional to:

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

- By expanding the expression, simplifying $\underline{\boldsymbol{x}}^T \underline{\underline{\boldsymbol{\Sigma}}}^{-1} \underline{\boldsymbol{x}}$ noticing that $\underline{\boldsymbol{x}}^T \underline{\underline{\boldsymbol{\Sigma}}}^{-1} (\underline{\boldsymbol{\mu_k}} - \underline{\boldsymbol{\mu_j}})$ plus its transposed value (because $\underline{\underline{\boldsymbol{\Sigma}}}$ is symmetrical) is equal to $2\underline{\boldsymbol{x}}^T \underline{\underline{\boldsymbol{\Sigma}}}^{-1} (\underline{\boldsymbol{\mu_k}} - \underline{\boldsymbol{\mu_j}})$ we get:

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

  which is a term linear in $\underline{\boldsymbol{x}}$

LDA / QDA: Learn model

- In practice:
  - We don't care about $\Pr(X = \underline{\boldsymbol{x}})$ since this is common for all classes
  - We assume to know the prior $\Pr(Y = k) = \pi_k$ or we estimate it from the data
  - We need to estimate the conditional probability $\Pr(X = \underline{\boldsymbol{x}} | Y = k)$
- The model assumes that the conditional probability has a Gaussian distribution:

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

  where:

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

- We can estimate the parameters $\underline{\boldsymbol{\mu_k}}$, $\underline{\underline{\boldsymbol{\Sigma_k}}}$ from the data using sample mean and covariance

Evaluating LDA / QDA

- When we get a new $\underline{\boldsymbol{x}} = \underline{\boldsymbol{x}}'$ we compute for each class $Y = k$

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

  and choose the k that maximizes the posterior probability

- Sometimes $f_i(\underline{\boldsymbol{x}})$ is from a multivariate Gaussian distribution where $\underline{\underline{\boldsymbol{\Sigma}}}$ is diagonal because of feature independence
  - Then the expressions can be simplified even more

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - ***Kernel methods***
  - Support vector machines
  - Similarity-based models
  - Clustering
  - Anomaly detection

Kernel: Definition

- Consider a transformation $\Phi : \mathcal{X} \to \mathcal{Z}$
  - E.g., features in a space $\mathcal{X}$ are transformed in a non-linear way to a higher dimensional space $\mathcal{Z}$
- The kernel $K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ of the transformation Φ is a function that yields the inner product of two points $\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}' \in \mathcal{X}$ in the transformed space $\mathcal{Z}$

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

Kernel: Expression from the transform

- If we have an expression for Φ, we can compute a closed formula for the kernel
- E.g., if $\Phi : \mathbb{R}^2 \to \mathbb{R}^6$

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

  - The transformation Φ introduces interaction terms
  - Then the kernel of Φ is:

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = 1 + x_1 x_1' + x_2 x_2' + x_1^2 x'^2_1$$

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

- The Gaussian kernel expression:

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

  provides intuition that a kernel measures the "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
  - The value is 0 when points are very distant
  - The strength of the effect depends on γ
- Using kernels to compute features:
  - Kernels often rely on the distance between vectors
    - E.g., Euclidean norm $\|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}'\|^2$
  - Need to scale features to ensure similar effects among various coordinates

Linear kernel

- The transformation Φ is 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 can be proved that this is always a kernel

Kernel: Identifying a function as a kernel

- In theory, a given function $K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ is a kernel $K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ is a valid kernel iff:
  - It is a symmetric, and
  - Satisfies the Mercer's condition: the matrix $K(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_j)$ is definite semi-positive
- We have a certain function $K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ and we want to show that $K(\cdot)$ is an inner product in the form for some function $\Phi(\cdot)$

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

  for a certain Φ and $\mathcal{Z}$

Kernel: example of identifying a kernel

- Let's show that:

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

  is a kernel for any n, k, d

- We need to show that there is always a transform Φ:

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

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

- E.g.,
  - $\mathcal{X} = \mathbb{R}^2$
  - $K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = (1 + \underline{\boldsymbol{x}}^T \underline{\boldsymbol{x}}')^2 = (1 + x_1 x_1' + x_2 x_2')^2$

- What is the function Φ and the transformed space $\mathcal{Z}$?
  - Computing the full expression in terms of the coordinates:

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

  - 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 people refer to the "kernel trick" as a computational shortcut to compute the dot product of transformed vectors
- Compare the 2 ways of computing the inner product of two transformed vectors for a polynomial kernel
  1. Using definition: compute the images of the vectors and then inner product in the transformed space:

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

     - It requires a combinatorial number of powers and then a huge dot product
  2. Kernel trick: use the kernel function to compute the dot product in transformed space

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

     - It requires a inner product of small vectors and then the power of a number
  - The kernel trick is much more computationally efficient to compute the inner product

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - ***Support vector machines***
  - 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")
- 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
- 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 is a large margin classifier

- Why large margin classifier is good?
- Given a linearly separable data set, the optimal separating line is the one that maximizes the margin:
  - Since it is more robust to the noise
  - By restricting ourselves to functions with a large margin reduces the VC dimension of the 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

- We can rewrite it as:

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

- 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

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

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

- Note that 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:

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

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

Solving dual formulation of SVM problem (1/2)

Solving for α

- Feeding this problem to a QP solver, we get the optimal vector $\underline{\boldsymbol{\alpha}}$

Compute hyperplane $\underline{\boldsymbol{w}}$

- From $\underline{\boldsymbol{\alpha}}$ we can 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, we can observe that many of them are 0
- This is because when we applied the Lagrange multipliers to the inequalities: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1$, we 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 $\alpha \neq 0$
- So the $\alpha_i \neq 0$ are the real degree of freedom

Compute b

- Once $\underline{\boldsymbol{w}}$ is known, we 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
- Thus we have an in-sample quantity to bound the out-of-sample error:

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

Non-linear transform for SVM

- We have a $\Phi : \mathcal{X} \to \mathcal{Z}$ that transforms $\underline{\boldsymbol{x}}_i$ into $\underline{\boldsymbol{z}}_i = \Phi(\underline{\boldsymbol{x}}_i) \in \mathbb{R}^{\tilde{d}}$ with $\tilde{d} > d$
- We can transform all the vectors through Φ and then apply all SVM machinery
- By exactly the same math using the points $\underline{\boldsymbol{z}}_i$ we get to the dual SVM formulation in the $\mathcal{Z}$ 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 that:
  - The optimization problem has the same number of unknowns as in the original space (i.e., the number of points N)
  - Support vectors live in $\mathcal{Z}$: they are the ones with $\alpha = 0$. In $\mathcal{X}$ there are the pre-images of the support vectors
  - The decision boundary and the margin can be represented in the original space (although they are 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 and cons

Pros - We don't pay the price in terms of complexity of optimization problem - Number of unknowns is still N (different than a linear problem) - We 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 - We pay a price to compute $\Phi(\underline{\boldsymbol{x}}_i)^T \Phi(\underline{\boldsymbol{x}}_j)$, since Φ could be very complex - The kernel trick will remove this extra complexity by doing $\Phi(\underline{\boldsymbol{x}}_i)^T \Phi(\underline{\boldsymbol{x}}_j) = K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_j)$

Trivial approach for 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
- The issue is that Φ might be very complex, e.g., potentially exponential number of terms
- If we can express the SVM problem formulation and the prediction in terms of a kernel

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

we would need the kernel of the transformation $\Phi(\cdot)$ (and not $\Phi(\cdot)$) itself

SVM formulation in terms of kernel: optimization step

- When we 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 formulation in terms of kernel: prediction step

- We need only inner products to compute a prediction for a given $\underline{\boldsymbol{z}}$
- In fact to make predictions, we 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
- We have seen that in order to use SVMs we need only to be able to compute inner products between transformed vectors $\underline{\boldsymbol{z}}$
- The kernel trick implies:
  - No need to compute $\Phi()$: we just need the kernel of the transformation K_Φ and not the transformation Φ itself
  - No need to know Φ: if we have a function K_Φ and we know that is an inner product in some space, we can still use all the SVM machinery, even if we don't know what is the $\mathcal{Z}$ space or what is the transformation Φ
  - Φ can be impossible to compute: K_Φ can even correspond to a transformation Φ to an infinite dimensional space (e.g., Gaussian kernel)

Non-linearly separable SVM problem

- In general there are 2 types of non-separable data sets:

1. Slightly non-separable
   - Few points crossing the boundary $\implies$ use soft margin SVMs
2. Seriously non-separable
   - E.g., the class inside the circle $\implies$ use non-linear transforms / kernels

- In practice, both issues are present and one can combine soft margin SVM and non-linear transforms

Soft-margin SVM for better generalization on linearly-separable data sets

- Sometimes, even if the data is linearly separable, one can get better E_{out} using soft margin SVM at the cost of worst E_{in}
  - Usual trade off between in-sample and out-of-sample performance
  - E.g., in the data set there are a few of outliers that are forcing a smaller margin than what we could obtain if we ignore them, in order to get all the points classified correctly
- If C parameter is very large the SVM optimization requires to make the error very small, and this might trade off a large margin with getting all the classification right

Primal formulation for soft margin SVM

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

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

we 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 defining 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 we 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 we 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, getting 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
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - ***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 $(\underline{\boldsymbol{x}}_n, y_n) \in D$
  - The effect is based on the distance $d(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_n) = \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_n\|$
- In other words, the model is the sum of the effect of each point in the training set, scaled down by the distance
  - The model is a superposition of effects

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

- 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

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

- 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

- One can represent graphically an RBF model
  - The (fixed by learning) params, one for each training point
  - The weights depending on the distance of the input to the examples
  - The weighted params are summed together

RBF: Reducing Model VC Dimension

- Some variants for RBF:
  - Add a bias term
  - Use different γ_i for each point, i.e., different influence of different points
  - Then the number of degrees of freedom increases even more
- In RBF there are many parameters:
  - There are as many parameters $\underline{\boldsymbol{w}}$ as data points N (e.g., $N = 10^9$)
  - One parameter w_i per training point (e.g., N can be 10^6)
  - Cons: negative consequences on generalization error
- To reduce number of parameters
  - Pick $K \ll N$ centers $\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$ instead of $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N$
    - We can use k-means clustering to find the centers
    - Note: this doesn't burn the training set since this is unsupervised learning and we don't use the labels
  - Same as RBF model using the distances from the centers of the clusters:

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

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

RBF: Learning Models

- We want to 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)

Learning RBF models

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

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

and get the problem:

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

  - We have N equations (one per point) and N unknowns $\underline{\boldsymbol{w}}$
  - $\underline{\underline{\boldsymbol{\Phi}}}$ is known since it is function of the data set and γ
- The problem in matrix form is like: $\underline{\underline{\boldsymbol{\Phi}}} \cdot \underline{\boldsymbol{w}} = \underline{\boldsymbol{y}}$

Learning RBF models

- The problem in matrix form is like

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

  - If $\underline{\underline{\mathbf{\Phi}}}$ is invertible, then $\underline{\boldsymbol{w}} = \underline{\underline{\mathbf{\Phi}}}^{-1}\underline{\boldsymbol{y}}$
    - We have the desired values on the training points and the exponential interpolates in the other points
  - If $\underline{\underline{\mathbf{\Phi}}}$ is not invertible, optimize the problem in a least square sense:

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

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

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

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

RBF network vs Neural Networks

- Consider the case of regression model for Neural Networks and RBF model
  - RBF:
$$h(\underline{\boldsymbol{x}}) = \sum_i w_i e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2} = \underline{\boldsymbol{w}}^T \underline{\boldsymbol{phi}}$$
  - Neural networks:
$$h(\underline{\boldsymbol{x}}) = \Theta(\underline{\boldsymbol{w}}^{(L)T} \underline{\boldsymbol{x}}^{(L)}) = \Theta(\underline{\boldsymbol{w}}^{(L)T} \underline{\boldsymbol{Theta}}(\underline{\underline{\boldsymbol{W}}}^{(L-1)} \ldots))$$
- Difference:
  - RBF has a single layer
  - Neural networks have multiple layers
- Similarities:
  - Combine features together with weights with a dot product
  - Have features extracted from the inputs
    - For RBF features are $e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}$ fixed and always > 0
    - For NN hidden layers synthesize features that 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)
  - In RBF all vectors (or centers of few clusters) contribute to the model
  - In SVM only support vectors contribute to the model

K-Nearest Neighbor (KNN) Model

- The model 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 ($k = 1$)
  - Adopt the response of the closest point to the point we are evaluating $\underline{\boldsymbol{x}}$
  - Like Voronoi tessellations: each point has a region for which it is the closest point and assigns its output to that region
- One could think there is a single parameter for KNN, i.e., k, since this is the hyperparameter we need to learn
  - For $k = 1$ there are N neighborhoods, one around each point of the training set
  - For $k = N$ there is a single neighborhood
  - So the intuition is that the effective number of parameters is $\frac{N}{k}$ since one can imagine there are N/k non-overlapping neighborhoods

KNN: Assumptions on the Data

- KNN makes no assumption on the data
  - This is the opposite of the linear model where there is a strong assumption on the data
- KNN assumes locality in parameter space
  - The model is constant in the neighborhood of an example
  - E.g., $k = 1$ we consider the Voronoi tesselation low-bias / high-variance
  - E.g., $k = N$ we consider the average value (high-bias / low-variance)

Training and test error for KNN

- For $k = 1$ a KNN model
  - Makes no error on the training set (assuming a non-noisy target), since it memorizes the training set (low bias / high variance)
  - E_{out} is larger than E_{in}
- If k increases the training error E_{in} increases but the test error E_{out} decreases until it starts increasing again
  - We have the typical behavior of model complexity as 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
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - ***Clustering***
  - Anomaly detection

K-means clustering: Problem Formulation

- We have N *unlabeled* points $\{\underline{\boldsymbol{x}}_1, \underline{\boldsymbol{x}}_2, ..., \underline{\boldsymbol{x}}_N\}$
- We want to partition the points into K clusters $S_1, ..., S_K$
  - Each cluster is defined by its center $\underline{\boldsymbol{\mu}}_k$
  - Each point $\underline{\boldsymbol{x}}_i$ is assigned to cluster $c(\underline{\boldsymbol{x}}_i)$
  - The unknowns are: $c(\underline{\boldsymbol{x}}_1), ..., c(\underline{\boldsymbol{x}}_N), \underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$
- We want to minimize the distance between each $\underline{\boldsymbol{x}}_i$ and the 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

- We start picking a random assignment of N points to the K clusters
  - Better than picking randomly the centroids of the clusters
- Each iteration does 2 steps
- **Step 1**: Move centroid
  - We move the centroid of each cluster to the mean point of the current cluster
  - This loop iterates over the 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
  - Each $\underline{\boldsymbol{x}}_n$ is assigned to the closest cluster based on its center
  - This loop iterates over the 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:
  - There is a finite (though large K^N) number of possible partitionings, and thus a finite number of possible values of the objective functions
  - The objective function $J(\cdot)$ is always decreased
- The objective function is always decreasing
  - The cost function $J(\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K, c_1, ..., c_N)$ can be seen as a function of:
    - The centroids $c_1, .., c_N$
    - The point assignments $\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$
  - At each step, K-means minimizes J with respect to:
    - The centroids (keeping the assignments fixed); then
    - The assignments (keeping the centroids fixed)
  - It is like coordinate descent
- Generally, it converges to a local minimum
  - Run K-means multiple times using different random initializations
  - Pick the best result

K-means clustering: non-separable clusters

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

Choosing the number of clusters

- It's often unclear how many clusters K exist in the data
  - E.g., visual analysis can be inconclusive with 2D or 3D data
  - Even more difficult in high dimensional spaces

1. Elbow Method
   - Vary the number of clusters K
   - Compute the optimal cost function $J(\cdot)$
   - Choose K at the "elbow" point if visible
   - The elbow is absent if the curve resembles a hyperbole $\approx 1/K$
2. End-to-end approach
   - Choose K to optimize later processing stages
   - E.g., More t-shirt sizes (i.e., more clusters) $\implies$
     - Satisfy customers
     - Complicates manufacturing
     - Increases stocking and inventory management

Clustering: Interpretation of Clusters

- Often we want to give a meaning to clusters
- Cluster meaning is difficult to automate: it must be interpreted manually
  - Examine the cluster centroids
    - Centroid values show the "typical" point in each cluster
    - High, low, or zero feature values highlight key characteristics
  - Analyze the 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
    - Helps understand separation and internal structure
  - Identify common traits in each cluster
    - For categorical features, count dominant categories
  - 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
  - Features: (Age, Annual Income, Spending Score)
  - Cluster 1
    - (25 yrs, 30K, 90) $\rightarrow$ "Young Big Spenders"

- Models
  - Naive Bayes
  - Decision trees
  - Random forests
  - Linear models
  - Perceptron
  - Logistic regression
  - LDA, QDA
  - Kernel methods
  - Support vector machines
  - Similarity-based models
  - Clustering
  - ***Anomaly detection***

Anomaly detection: problem formulation

- **Problem**:
  - We have $\{\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N\}$ examples with features $\underline{\boldsymbol{x}} \in \mathbb{R}^P$ for good / non-anomalous instances
  - We want to find a way to detect bad / anomalous instances
- **Algorithm**:
  - We don't know what makes a *"bad instances"*
  - We learn what *"good instances"* have in common using unsupervised learning
    - I.e., find the distribution for *"good instances"* $\underline{\boldsymbol{x}}_i$, $\Pr(\underline{\boldsymbol{x}} \text{ is good})$
  - Pick features
    - The goal is to find "sensitive" features, i.e., features that might take large or small values in case of an anomaly
    - E.g., ratio between CPU load and network traffic
  - Estimate the distribution $\Pr(\underline{\boldsymbol{x}} \text{ is good})$
  - Choose the threshold ε
  - For a new instance $\underline{\boldsymbol{x}}_{new}$, if

$$\Pr(\underline{\boldsymbol{x}}_{new} \text{ is good}) \leq \varepsilon$$

    we flag it as an 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 a PDF $\Pr(\underline{\boldsymbol{x}} \text{ is good})$
  - Decide if a new engine is acceptable $\Pr(\underline{\boldsymbol{x}}_{good}) \leq \varepsilon$ 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 a PDF $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Identify unusual users by checking $\Pr(\underline{\boldsymbol{x}}_{new}) \leq \varepsilon$

Anomaly detection: example of computers in 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 a PDF $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Identify unusual users by checking $\Pr(\underline{\boldsymbol{x}}_{new}) \leq \varepsilon$

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, we can 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:

$$\Pr(\underline{\boldsymbol{x}}_{new} \text{ is good}) \leq \varepsilon$$

  to flag an anomaly

Estimate univariate Gaussian model

- We have N (scalar) examples $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N \in \mathbb{R}$ for "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

- We 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
- Vectorize the computation

Estimate a multi-variate Gaussian model

- **Problem**:
  - Sometimes features vary together (e.g., network use and CPU load), so using the independent assumption might cause misclassifications
  - E.g., the components of an example $\underline{\boldsymbol{x}}_{new}$ are within the expected range but together make no sense
    - E.g., low network use with high CPU load
- **Solution 1**:
  - Engineer features to create features that are high in case of an anomaly
  - This bridges the gap for the correlation between variables (that can't be modeled in independent Gaussian models)
- **Solution 2**:
  - Estimate the entire multivariate model, instead of assuming independence and estimating one marginal Gaussian at a time

Estimate a multi-variate Gaussian model

- The PDF of a multi-variate Gaussian is:

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

- We estimate:

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

and

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

- This model requires more examples to train since there are more parameters to fit
- If there is independence between the variables, then the multivariate Gaussian is decomposed into a 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 the number of anomalous examples $y = 1$ is much smaller than good examples with $y = 0$
  - E.g., 10,000 good vs 20 bad examples
  - Important to address class imbalance for accurate model performance
- **Algorithm**:
  - Pick 60% of data with $y = 0$ to train (only on the good examples)
  - Split the remaining data $y = 0$ and $y = 1$ into validation and test sets
    - Ensure both sets are representative of the overall dataset
    - Train, validation, and test sets should have no overlap but have the same characteristics
    - This helps in evaluating the model's performance accurately
  - Use the validation set to compare models, estimate the hyper parameters
    - E.g., ε is the threshold for anomaly detection
  - Use the test set to evaluate the final model
    - The model is trained on normal data and tested on both normal and anomalous data
- **Example**:
  - In aircraft engine example there are 10,000 good engines ($y = 0$), 20 bad engines ($y = 1$)
  - Train set: 6,000 $y = 0$ examples
  - Validation set: 2,000 $y = 0$ and 10 $y = 1$
  - Test set: 2,000 $y = 0$ and 10 $y = 1$

Anomaly detection vs supervised learning

- Even in unsupervised learning, we need *labeled* data for model evaluation
- What's the difference with supervised learning?
  - In anomaly detection/unsupervised learning, we train only on good examples
  - In supervised learning, we train on both good and bad examples
- Use:
  - Anomaly detection/unsupervised learning:
    - When learning only from good examples due to few anomalous examples
    - When having a strong prior on the model
    - When future anomalous examples are unknown (no prior)
  - Supervised learning when the training set has less skewed classes
  - It is not a clear-cut decision