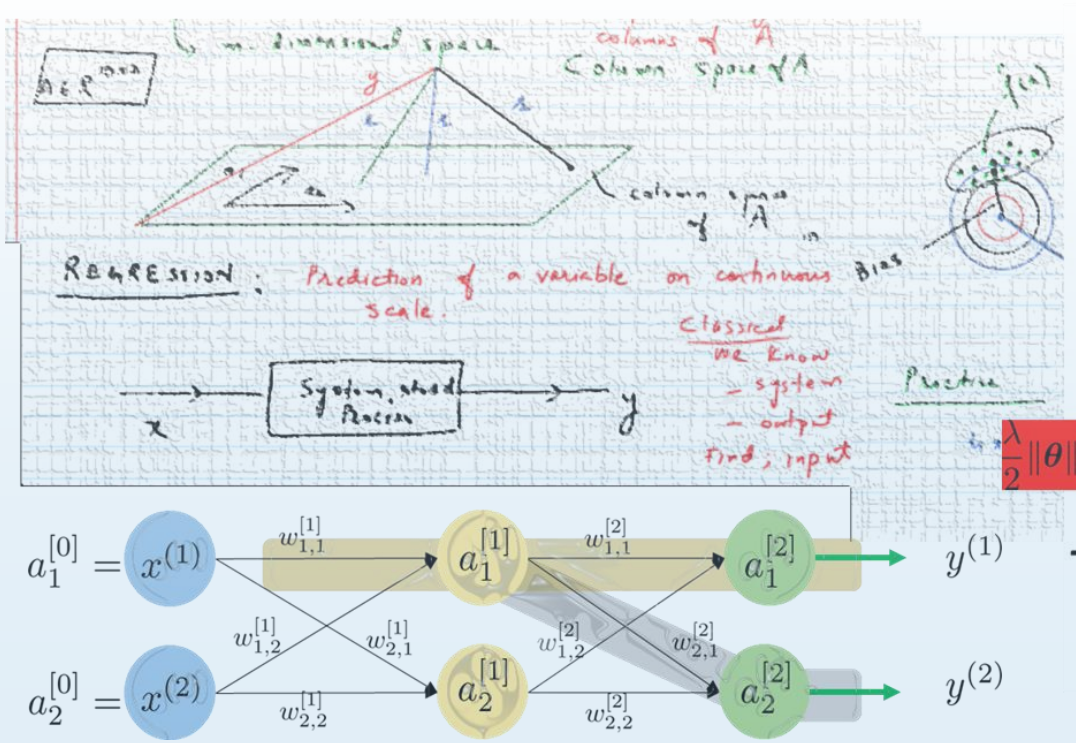
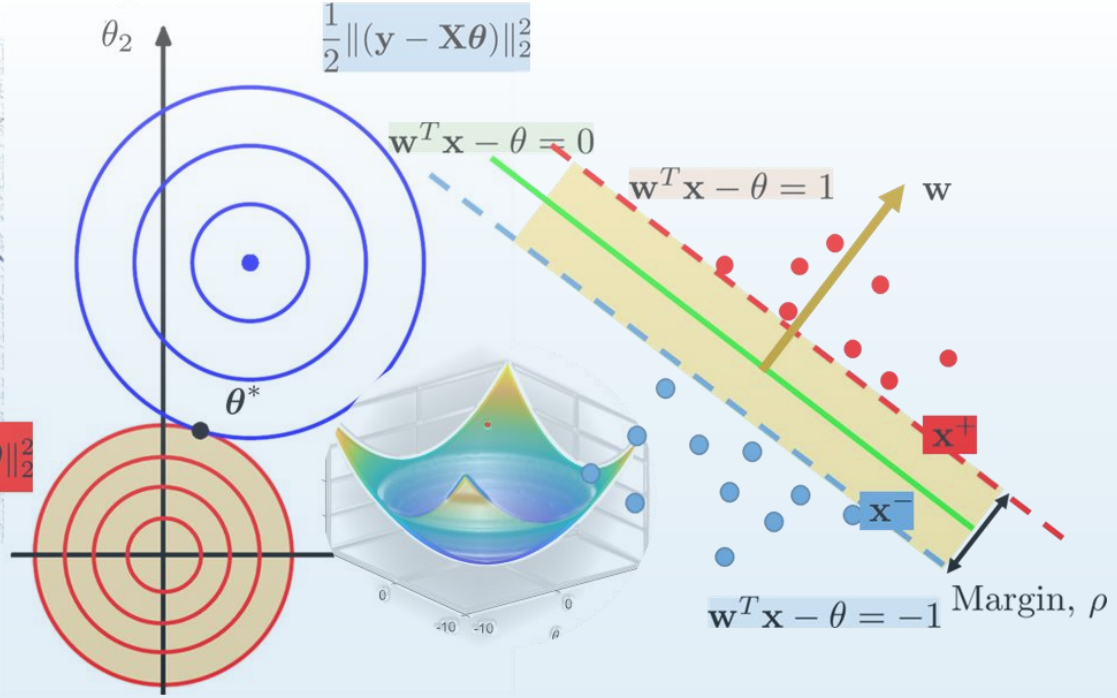




AI-501 Mathematics for AI

Machine Learning – Supervised Learning and kNN

Zubair Khalid
School of Science and Engineering

https://www.zubairkhalid.org/ai501_2024.html

LUMS
A Not-for-Profit University

Supervised Learning Setup

Nomenclature

In these regression or classification problems, we have

- Inputs – referred to as Features
- Output – referred to as Label
- Training data – (input, output) for which the output is known and is used for training a model by ML algorithm
- A Loss, an objective or a cost function – determines how well a trained model approximates the training data
- Test data – (input, output) for which the output is known and is used for the evaluation of the performance of the trained model

Supervised Learning Setup

Nomenclature - Example

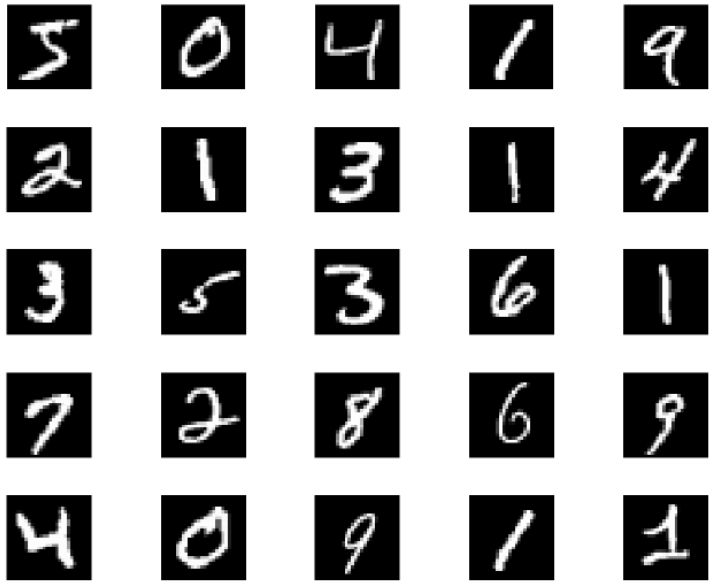
Predict Stock Index Price

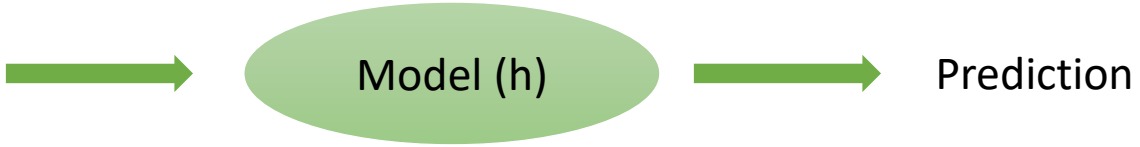
- *Features (Input)*
- *Labels (Output)*
- *Training data*

Interest_Rate	Unemployment_Rate	Stock_Index_Price
2.75	5.3	1464
2.5	5.3	1394
2.5	5.3	1357
2.5	5.3	1293
2.5	5.4	1256
2.5	5.6	1254
2.5	5.5	1234
2.25	5.5	1195
2.25	5.5	1159
2.25	5.6	1167
2	5.7	1130
2	5.9	1075
2	6	1047
1.75	5.9	965
1.75	5.8	943
1.75	6.1	958
1.75	6.2	971
1.75	6.1	949
1.75	6.1	884
1.75	6.1	866
1.75	5.9	876
1.75	6.2	?
1.75	6.2	?
1.75	6.1	?

Supervised Learning Setup

Example





MNIST Data:

- Each sample 28x28 pixel image
- 60,000 training data
- 10,000 testing data

Supervised Learning Setup

Formulation

We assume that we have d columns (features) of the input. In this example, we have two features; interest rate and unemployment rate, that is, $d = 2$.

In general, we use $\mathbf{x_i}$ to refer to featues of the i-th sample, that is,

$$\mathbf{x_i} = [x_{i,1}, x_{i,2}, x_{i,3}, \ldots x_{i,d}]$$

If y_i is the label associated with the i-th sample $\mathbf{x}_i$, we formulate training data in pairs as

$$(\mathbf{x_i}, y_i), \quad i = 1, 2, \ldots, n$$

Here, n denotes the number of samples in the training data. In this example, we have $n = 21$

Interest_Rate	Unemployment_Rate	Stock_Index_Price
2.75	5.3	1464
2.5	5.3	1394
2.5	5.3	1357
2.5	5.3	1293
2.5	5.4	1256
2.5	5.6	1254
2.5	5.5	1234
2.25	5.5	1195
2.25	5.5	1159
2.25	5.6	1167
2	5.7	1130
2	5.9	1075
2	6	1047
1.75	5.9	965
1.75	5.8	943
1.75	6.1	958
1.75	6.2	971
1.75	6.1	949
1.75	6.1	884
1.75	6.1	866
1.75	5.9	876
1.75	6.2	?
1.75	6.2	?
1.75	6.1	?

Supervised Learning Setup

Formulation

Using the adopted notation, we can formalize the supervised machine learning setup. We represent the entire training data as

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

Here $\mathcal{X}^d$ - d dimensional feature space and $\mathcal{Y}$ is the label space.

Regression: $\mathcal{Y} = \mathbf{R}$ (prediction on continuous scale)

Classification: $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, 1\}$ or $\mathcal{Y} = \{1, 2\}$ (Binary classification)

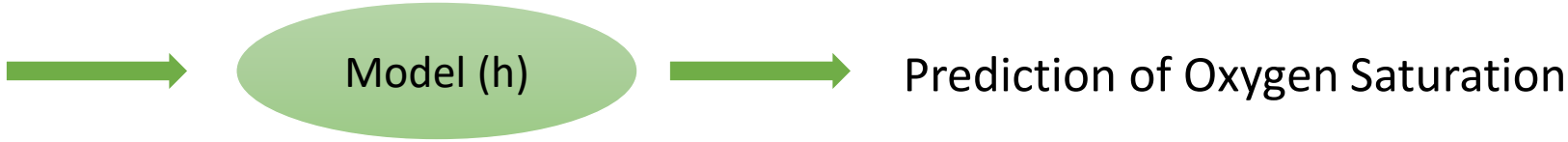
$\mathcal{Y} = \{1, 2, \ldots, M\}$ (M-class classification)

Supervised Learning Setup

Example

Data of 200 Patients:

- Age of the patient
- Cholesterol levels
- Glucose levels
- BMI
- Height
- Heart Rate
- Calories intake
- No. of steps taken



Supervised Learning Setup

Learning

Recall a problem in hand. We want to develop a model that can predict the label for the input for which label is unknown.

We assume that the data points $(\mathbf{x_i}, y_i)$ are drawn from some (unknown) distribution $P(X, Y)$.

Our goal is to learn the machine (model, function or hypothesis) h such that for a new pair $(\mathbf{x}, y)$ P, we can use h to obtin

$$h(\mathbf{x}) = y$$

with high probability or

$$h(\mathbf{x}) \approx y$$

in some optimal sense.

Supervised Learning Setup

Hypothesis Class

We call the set of possible functions or candidate models (linear model, neural network, decision tree, etc.) "the hypothesis class".

Denoted by $\mathcal{H}$

For a given problem, we wish to select hypothesis (machine) $h \in \mathcal{H}$.

Q: How?

A: Define hypothesis class $\mathcal{H}$ for a given learning problem.

Evaluate the performance of each candidate function and choose the best one.

Supervised Learning Setup

Q: How do we evaluate the performance?

A: Define a loss function to quantify the accuracy of the prediction.

Loss Function

Loss function should quantify the error in predicting y using hypothesis function h and input $\mathbf{x}$.

Denoted by $\mathcal{L}$.

Supervised Learning Setup

0/1 Loss Function:

Zero-one loss is defined as $$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} 1 - \delta_{h(\mathbf{x_i})-y_i}$$

Here $\delta_{h(\mathbf{x_i})-y_i}$ is the delta function defined as

$$\delta_k = \begin{cases} 1, & k = 0 \\ 0 & \text{otherwise} \end{cases}$$

Interpretation:

- Note normalization by the number of samples. This makes it the loss per sample.
- Loss function counts the number of mistakes made by hypothesis function on D.
- **Not used frequently due to non-differentiability and non-continuity.**

Supervised Learning Setup

Squared Loss Function:

Squared loss is defined as (also referred to as **mean-square error, MSE**)

$$\mathcal{L}_{\text{sq}}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(h(\mathbf{x_i}) - y_i\right)^2$$

Interpretation:

- Again note normalization by the number of samples.
- Loss grows quadratically with the absolute error amount in each sample.

Root Mean Squared Error (RMSE):

RMSE is just square root of squared loss function:

$$\mathcal{L}_{\text{rms}}(h) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(h(\mathbf{x_i}) - y_i\right)^2}$$

Supervised Learning Setup

Absolute Loss Function (Mean Absolute Error):

Absolute loss is defined as

$$\mathcal{L}_{\text{abs}}(h) = \frac{1}{n}\sum_{i=1}^{n} |h(\mathbf{x_i}) - y_i|$$

Interpretation:

- Loss grows linearly with the absolute of the error in each prediction.
- Used in regression and suited for noisy data.

* All of the losses are non-negative

Supervised Learning Setup

Learning

We wish to select hypothesis (machine) $h \in \mathcal{H}$ such that

$$h^* = \min_{h \in \mathcal{H}} \mathcal{L}(h)$$

(Optimization problem)

Recall We assume that the data points $(\mathbf{x_i}, y_i)$ are drawn from some (unknown) distribution $P(X, Y)$.

We can come up with a function h after solving this minimization problem that gives low loss on our data.

Q: How can we ensure that hypothesis *h* will give low loss on the input not in *D*?

Supervised Learning Setup

To illustrate this, let us consider a model h trained on every input in D, that is, giving zero loss. Such function is referred to as memorizer and can be formulated as follows

$$h(\mathbf{x}) = \begin{cases} y_i, & \exists\ (\mathbf{x}_i, y_i) \in D, \quad \mathbf{x}_i = \mathbf{x}, \\ 0, & \text{otherwise} \end{cases}$$

Interpretation:

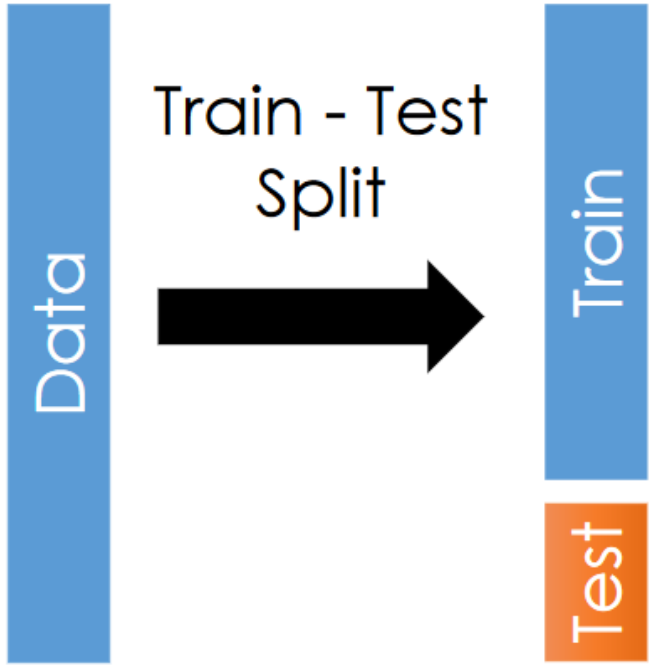
- 0% loss error on the training data (Model is fit to every data point in D).
- Large error for some input not in D
- First glimpse of ***overfitting.***

Revisit:

Q: How can we ensure that hypothesis *h* will give low loss on the input not in *D*?

A: Train/Test Split

Supervised Learning Setup



Generalization: The Train-Test Split

To resolve the overfitting issue, we usually split D into train and test subsets:

- D_{TR} as the training data, (70, 80 or 90%)
- D_{TE} as the test data, (30, 20, or 10%)

How to carry out splitting?

- Split should be capturing the variations in the distribution.
- Usually, we carry out splitting using i.i.d. sampling and time series with respect to time

You can only use the test dataset once after deciding on the model using training dataset

Supervised Learning Setup

Learning (Revisit after train-test split)

We had the following optimiztion problem as

$$h^* = \min_{h \in \mathcal{H}} \mathcal{L}(h)$$

We generalize it as

$$h^* = \min_{h \in \mathcal{H}} \frac{1}{|D_{\text{TR}}|} \sum_{(\mathbf{x}, y) \in D_{\text{TR}}} \mathcal{L}(\mathbf{x}, y)|h)$$

Evaluation

Loss on the testing data is given by

$$\epsilon_{\text{TE}} = \frac{1}{|D_{\text{TE}}|} \sum_{(\mathbf{x}, y) \in D_{\text{TE}}} \mathcal{L}(\mathbf{x}, y)|h*)$$

Supervised Learning Setup

Generalization: The Train-Test Split

At times, we usually split D into three subsets, that is, the training data is further divided into traaining and validation datasets:

- D_{TR} as the training data, (80%)
- D_{VA} as the validation data, (10%)
- D_{TE} as the test data, (10%)

Q: Idea:

Validation data is used to evaluate the loss for a function h that is determined using the learning on the training data-set. If the loss on validation data is high for a given h, the hypothesis or model needs to be changed.

Supervised Learning Setup

Generalization: The Train-Test Split

More explanation* to better understand the difference between validation and test data:

- **Training set:** A set of examples used for learning, that is to fit the parameters of the hypothesis (model).

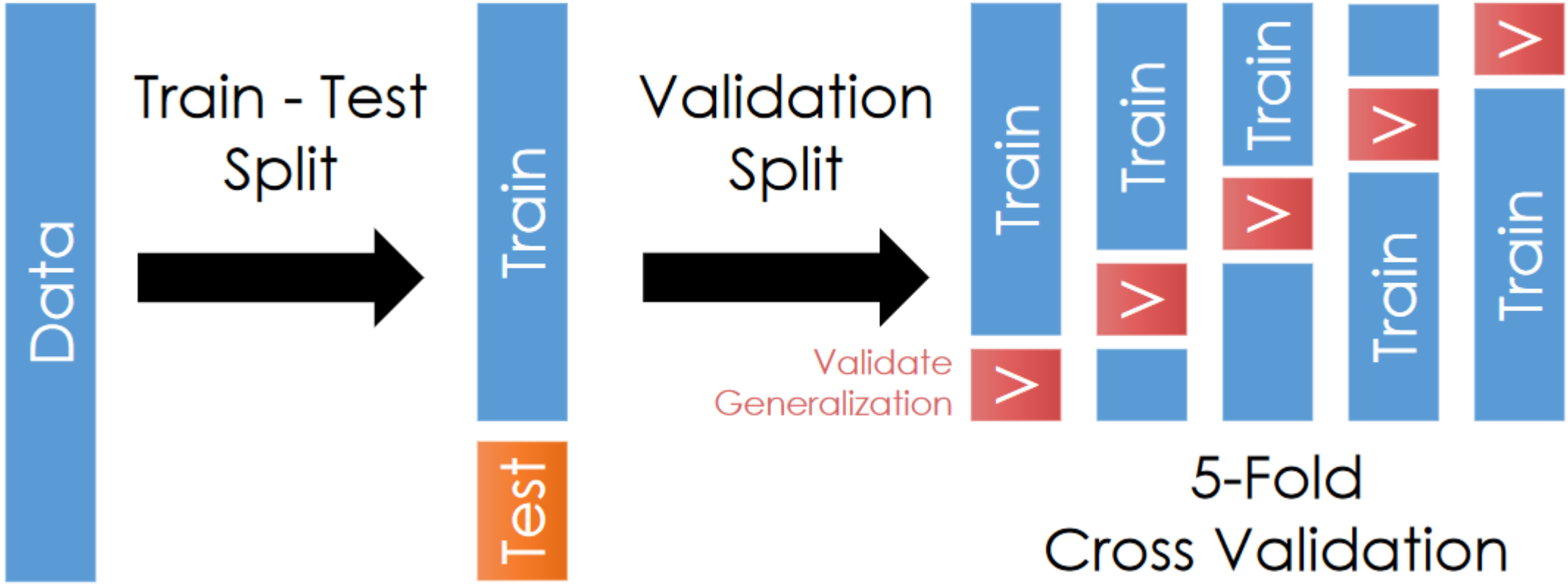
- **Validation set:** A set of examples used to tune the hyper-parameters of the hypothesis function, for example to choose the number of hidden units in a neural network OR the order of polynomial approximating the data.

- **Test set:** A set of examples **used** only to assess the performance of a fully-specified model or hypothesis.

Adapted from *Brian Ripley, Pattern Recognition and Neural Networks, 1996

Supervised Learning Setup

Generalization: The Train-Test Split (Example)

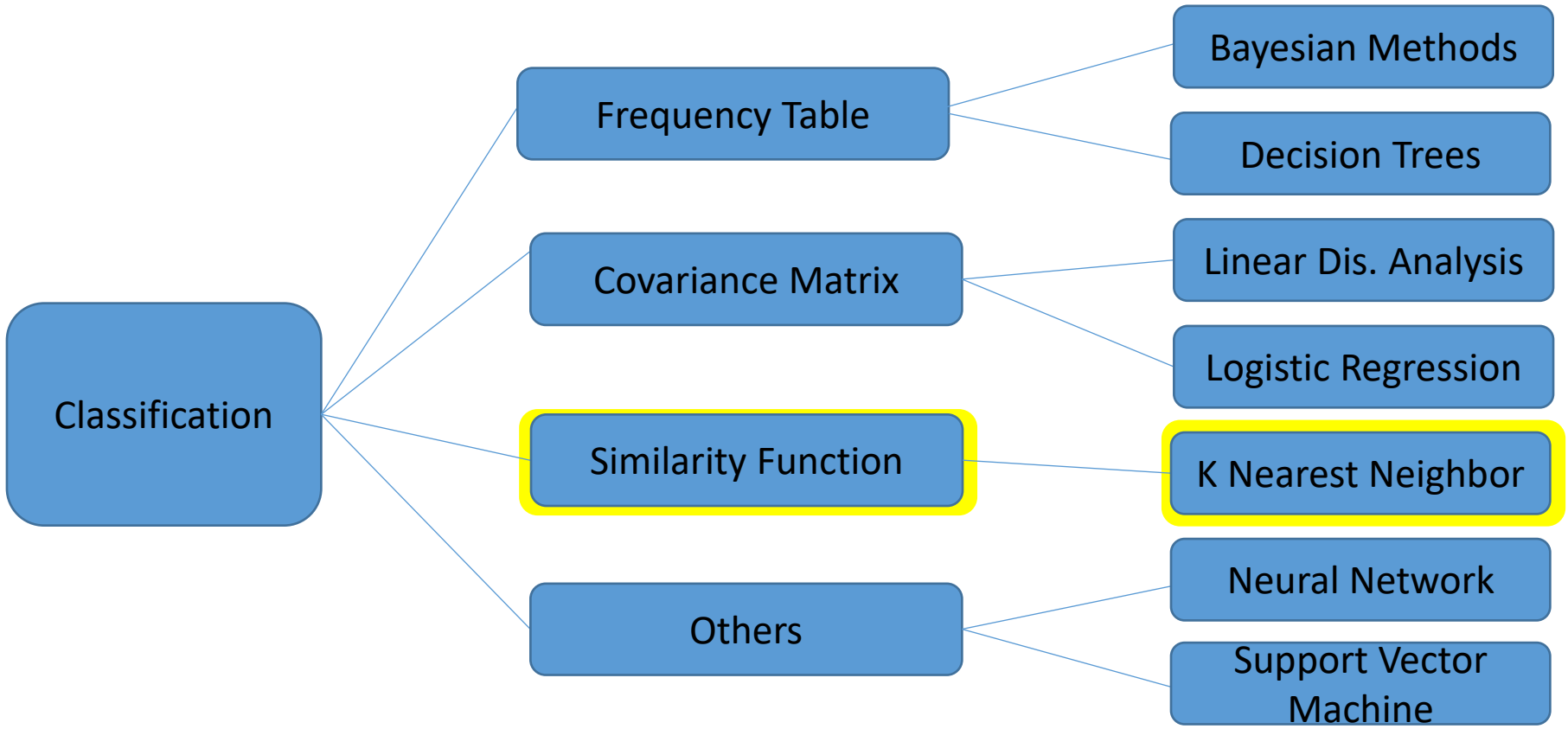


Cross validation simulates multiple train-test splits on the training data

Supervised Learning

Classification Algorithms or Methods

Predicting a categorical output is called classification

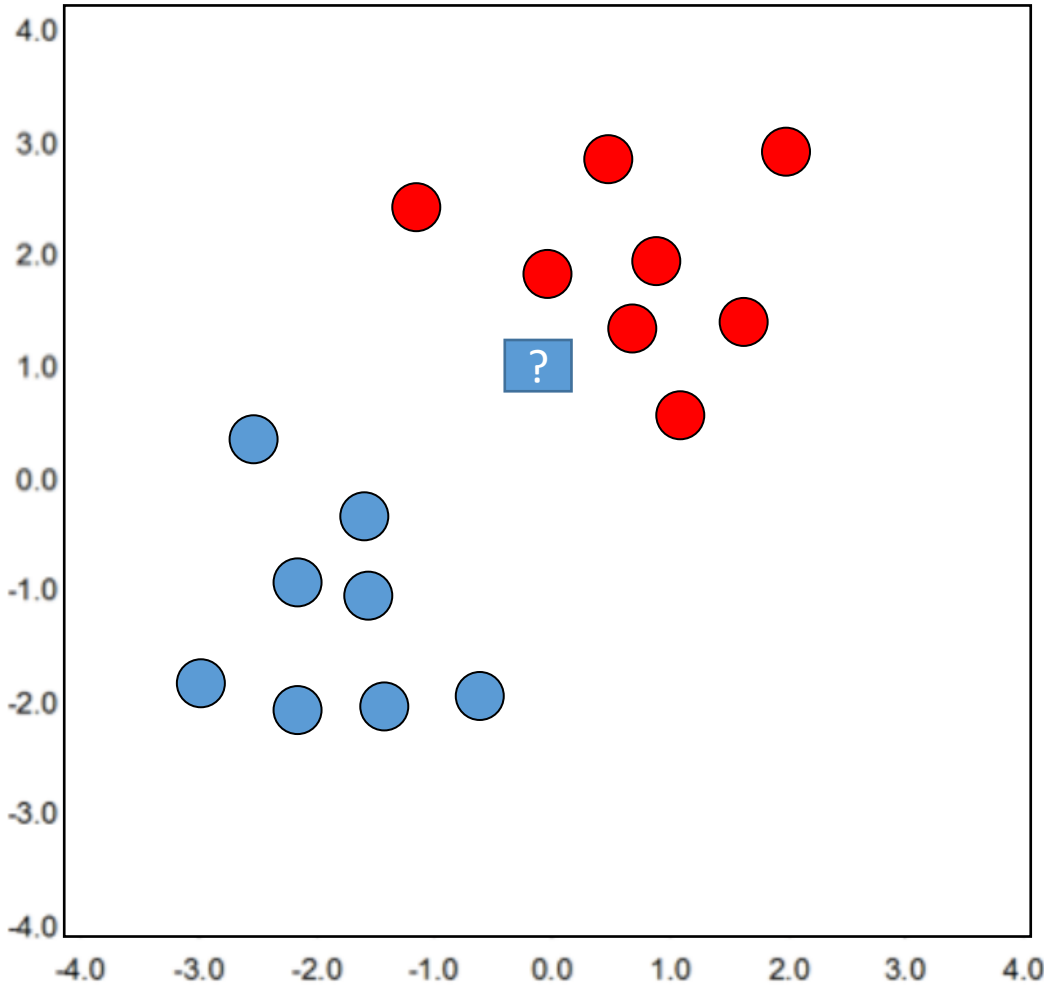


Outline

- k-Nearest Neighbor (kNN) Algrorithm Overview
- Algorithm Formulation
- Choice of k
- Storage, Time Complexity Analysis

k-Nearest Neighbor (kNN) Algorithm

Idea:



- Two classes, two features
- We want to assign label to unknown data point?
- Label should be red.

k-Nearest Neighbor (kNN) Algorithm

Idea:

- We have similar labels for similar features.
- We classify new test point using similar training data points.

Algorithm overview:

- Given some new test point x for which we need to predict the class y.
- Find **most similar** data-points in the training data.
- Classify x "like" these **most** similar data points.

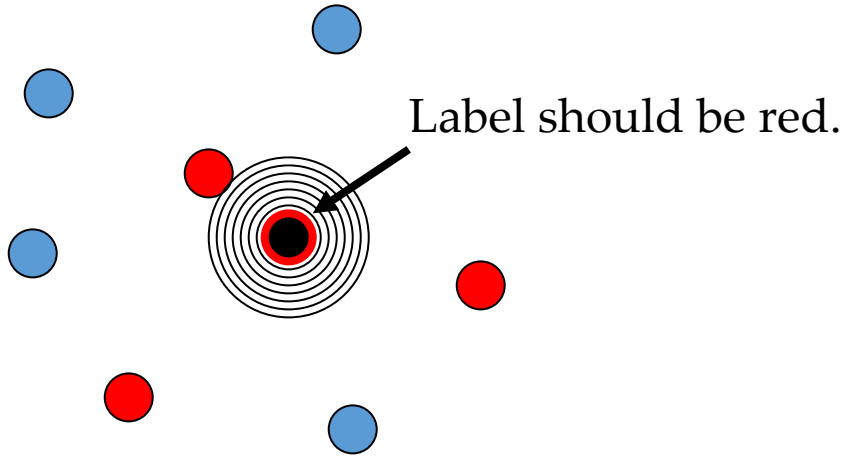
Questions:

- How do we determine the similarity?
- How many similar training data points to consider?
- How to resolve inconsistencies among the training data points?

k-Nearest Neighbor (kNN) Algorithm

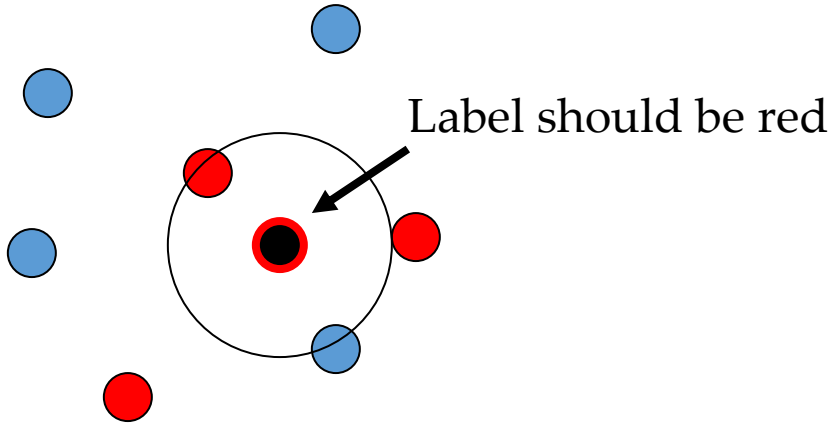
1-Nearest Neighbor:

Simplest ML Classifier
*Idea: Use the label of the **closest** known point*

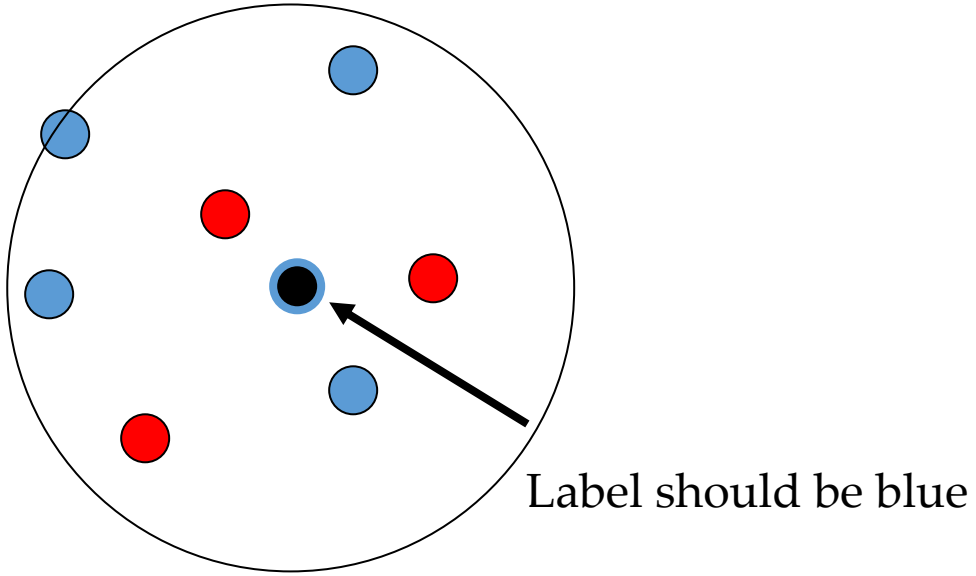


Generalization:

*Determine the label of **k** nearest neighbors and assign the most frequent label*



k=3



k=7

k-Nearest Neighbor (kNN) Algorithm

Formal Definition:

- We assume we have training data D given by

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- $\mathcal{Y} = \{1, 2, \ldots, M\}$ (M-class classification)
- For a point $\mathbf{x} \in \mathcal{X}^d$, we define a set $S_\mathbf{x} \subseteq D$ as a set of k neighbors.
- Using the function 'dist' that computes the distance between two points in $\mathcal{X}^d$, we can define a set $S_\mathbf{x}$ of size k as (A set of k nearest neighbors)

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_\mathbf{x}} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \backslash S_\mathbf{x}$$

Interpretation:

Every point in D but not in $S_\mathbf{x}$ is at least as far away from $\mathbf{x}$ as the furthest point in $S_\mathbf{x}$.

k-Nearest Neighbor (kNN) Algorithm

Formal Definition:

- Using the $S_{\mathbf{x}}$, we can define a classifier as a function that gives us most frequent label of the data points in $S_{\mathbf{x}}$

$$h(\mathbf{x}) = \text{mode}(\{y'' : (x'', y'') \in S_{\mathbf{x}}\})$$

- *Instance-based learning algorithm; easily adapt to unseen data*

k-Nearest Neighbor (kNN) Algorithm

Decision Boundary:

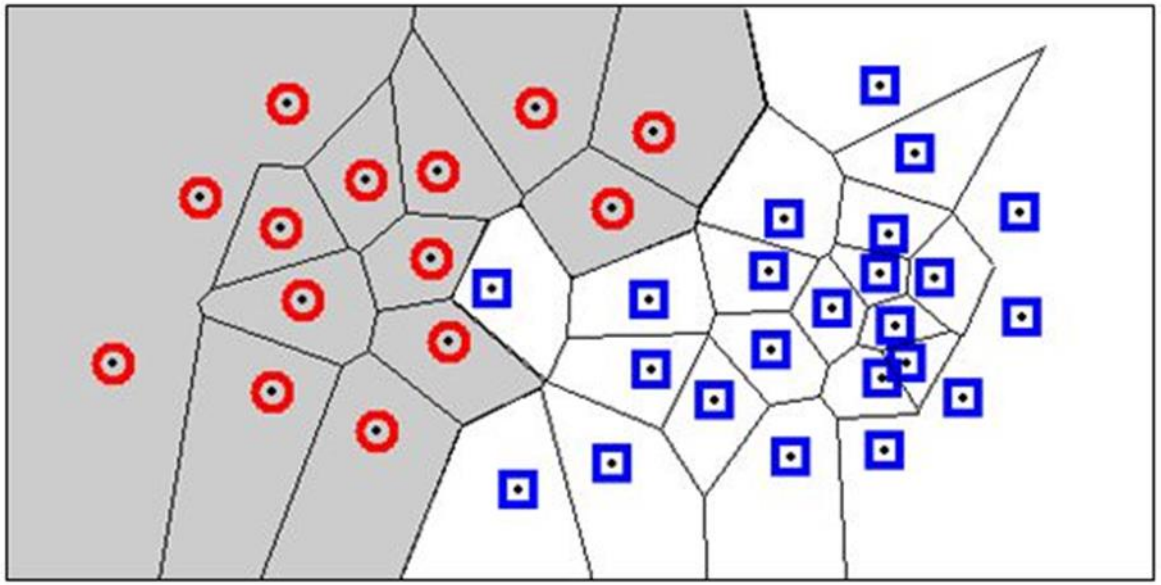
For $k = 1$, kNN defines a region, called decision boundary or region, in the space. Such division of the featue space is referred to s Voronoi partitioning.

We can define a region R_i associated with the feature point $\mathbf{x}_i$ as

$$R_i = \{\mathbf{x} : \text{dist}(\mathbf{x}, \mathbf{x}_i) < \text{dist}(\mathbf{x}, \mathbf{x}_j),\ i \neq j\}$$

For example, Voronoi partitioning using Euclidean distance in two-dimensional space.

Classification boundary changes with the change in k and the distance metric.



k-Nearest Neighbor (kNN) Algorithm

Decision Boundary:

Demonstration

https://demonstrations.wolfram.com/KNearestNeighborKNNClassifier/

k-Nearest Neighbor (kNN) Algorithm

Characteristics of kNN:

- No assumptions about the distribution of the data
- Non-parametric algorithm
  - No parameters

- Hyper-Parameters
  - k (number of neighbors)
  - Distance metric (to quantify similarity)

k-Nearest Neighbor (kNN) Algorithm

Characteristics of kNN:

- Complexity (both time and storage) of prediction increases with the size of training data. We will review this shortly.

- Can also be used for regression (average or inverse distance weighted average)
  - For example,

$$y = \frac{1}{k}\sum_{i=1}^{k} y_i, \quad (\mathbf{x}_i, y_i) \in S_{\mathbf{x}}$$

k-Nearest Neighbor (kNN) Algorithm

Practical issues:

- For binary classification problem, use odd value of k. Why?

- In case of a tie:
  - Use prior information
  - Use 1-nn classifier or k-1 classifier to decide

- Missing values in the data
  - Average value of the feature.

k-Nearest Neighbor (kNN) Algorithm

Distance Metric:

- For categorical vaiable, use Hamming Distance

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} 1 - \delta_{x_i - x_i'}$$

k-Nearest Neighbor (kNN) Algorithm

Practical issues in computing distance:

- Mismatch in the values of data
  - Issue: Distance metric is mapping from d-dimensional space to a scaler. The values should be of the same order along each dimension.

  - Solution: Data Normalization

Outline

- k-Nearest Neighbor (kNN) Algroithm Overview
- Algorithm Formulation
- **Choice of k**
- Storage, Time Complexity Analysis

k-Nearest Neighbor (kNN) Algorithm

Choice of k:

- k=1

Sensitive to noise

High variance

Increasing k makes algorithm less sensitive to noise

- k=n

Decreasing k enables capturing finer structure of space

Idea: Pick k not too large, but not too small (depends on data)

How?

k-Nearest Neighbor (kNN) Algorithm

Choice of k:

- Learn the best hyper-parameter, k using the data.
- Split data into training and validation.
- Start from k=1 and keep iterating by carrying out (5 or 10, for example) cross-validation and computing the loss on the validation data using the training data.
- Choose the value for k that minimizes validation loss.
- This is the only learning required for kNN.

Outline

- k-Nearest Neighbor (kNN) Algroithm Overview
- Algorithm Formulation
- Choice of k
- **Storage, Time Complexity Analysis**

k-Nearest Neighbor (kNN) Algorithm

Algorithm Computational and Storage Complexity:

Input/Output:

- We have a feature vector, $\mathbf{x}$ for which we want to predict label y.
- We have k and dist function.

Steps:

- We defined a set $S_{\mathbf{x}}$ of size k as

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \backslash S_{\mathbf{x}}$$

- Classifier that gives us most frequent label of the data points in $S_{\mathbf{x}}$

$$h(\mathbf{x}) = \text{mode}(\{y'' : (x'', y'') \in S_{\mathbf{x}}\})$$

k-Nearest Neighbor (kNN) Algorithm

Algorithm:

Steps:

Computational Complexity

1. *Find distance between given test point and feature vector of every point in D.*

   Noting n number of data points we have and each feature vector $\mathbf{x}$ is d-dimensional. $\mathcal{O}(dn)$

2. *Find k points in D closest to the given test point vector to form a set S_X.*

   Finding k-th smallest distance using median of medians method. $\mathcal{O}(n)$

   Finding k data-points in D with distance less than the k-th smallest distance $\mathcal{O}(n)$

3. *Find the most frequent label in the set S_x and assign it to the test point.* $\mathcal{O}(k)$

Computational Complexity: $\mathcal{O}(dn)$

Space Complexity: $\mathcal{O}(dn)$