Classification

{Using sensor data to infer semantic information}

* Classification Problem

- Given a set of K classes

$$\Omega = \{\omega_1, \omega_2, \cdots \omega_K\}$$

- and feature e
- learn a function f that assigns a class given feature

$$c = f(e) \text{ with } c \in \Omega$$

* Learning Needs Information

- Learning the function f requires additional information.
- This additional information/Knowledge can be provided through distributions
- Manually specifying such knowledge is often hard and thus should be learned from data.

- Training data are pairs of feature vector and classes:

$$(\bar{e}, \omega) \xrightarrow[\{\text{Learning}\}]{} f$$

* Supervised Learning

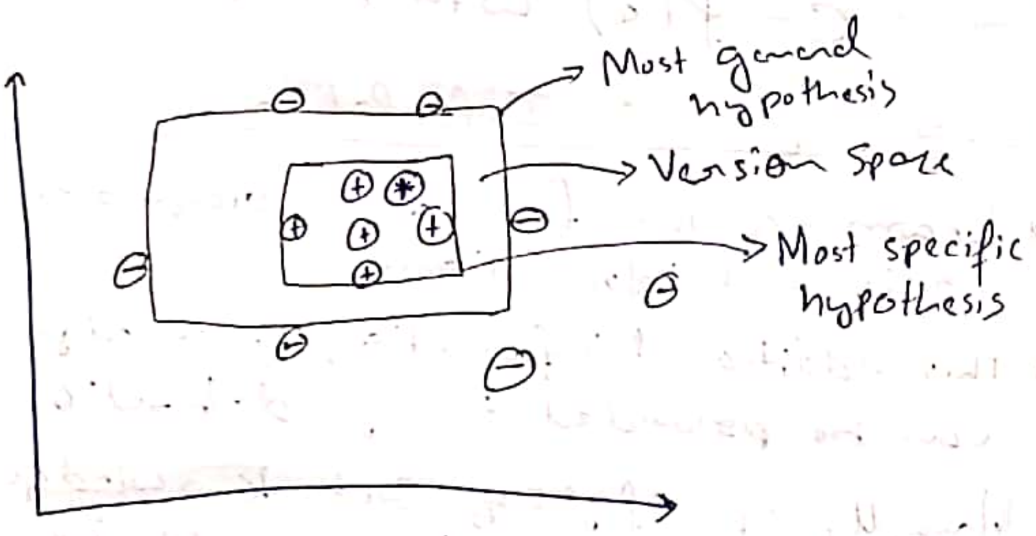
⇒ Learn a function f that generates /generalizes this decision to new (unseen) data.

* Classification

- Output is always a class label.
- Goal: find a Separation of the input space that correspond to the class.

* Regression

- Output is Continuous Variable
- Goal: Function that fits a set of data points.



⇒ Choose the hypothesis that maximizes the margin to the most specific and General one.

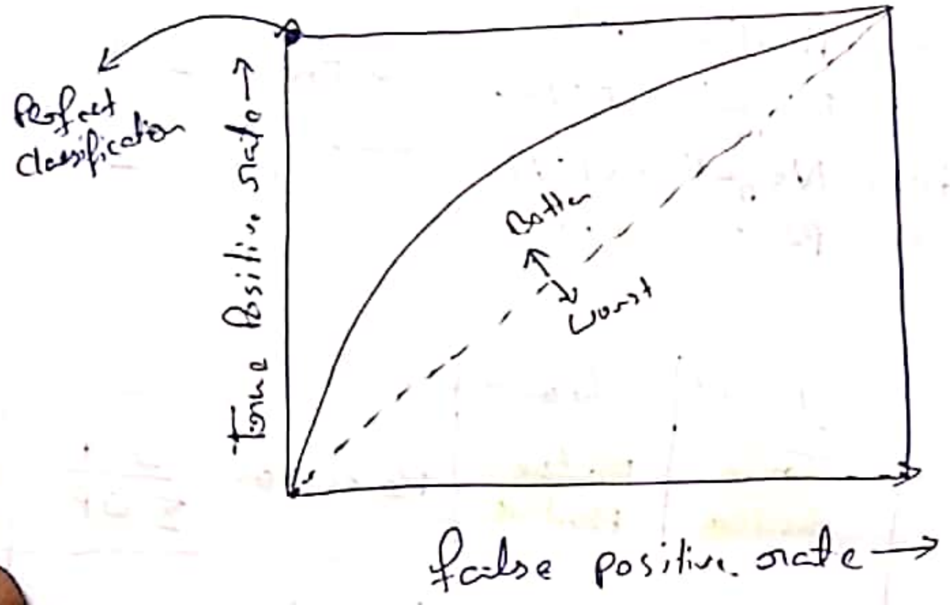
* Classification Error

⇒ Possible outcomes:-

1. True Positive (TP)
2. False Negative (FN) → Error I
3. True Negative (TN) → Error II.
4. False positive (FP)

	Condition +ve	Condition -ve	
Test outcome Positive	True Positive	False Positive	Precision $= \frac{\sum TP}{\sum OP}$
Test outcome negative	False negative	True Negative	Negative Prediction value $= \frac{\sum TN}{\sum ON}$
	Sensitivity $= \frac{\sum TP}{\sum CP}$	Specificity $= \frac{\sum TN}{\sum CN}$	Accuracy $= \frac{\sum TP + \sum TN}{\sum TP}$
	also called recall or true Positive rate	also called true negative rate	

★ Receiver Operating Characteristic (ROC Curve)



Classification

★ Designing a Classifier

Training Phase

Collecting labeled training data

⇓

Selecting and Computing appropriate features

⇓

Learning the distribution of features for each class or discriminant function.

Testing Phase (On different dataset)

Extract features (as in the training phase)

⇓

Determine class based the classifier (trained with a different dataset)

⇓

Evaluate the Performance

Operational Phase

Extract features

⇓

Determine class based the classifier

* Components of the Generalization Error

- **Bias**: Describes how much the average model over all training set differ from the true model.
- **Variance**: Describes how much models estimated from different training set differ from each other.

$$E(MSE) = noise^2 + bias^2 + Variance$$

{Estimated mean Squared Error}

bias-Variance trade-off

* Underfitting

"Model is too "simple" to represent the relevant characteristics".

→ High bias and Low Variance

→ High training error and high test error.

* Overfitting

Model is to "Complex" and fit irrelevant characteristics (noise) in the data.

→ Low bias and high variance
→ Low training error & high test error.

* Rules of Thumb

⇒ Try Simple classifiers first
⇒ Use increasingly more powerful classifiers with more training data.

⇒ Find good features

↳ Better to have Smart features and simple classifiers, than Simple features and Smart classifiers.

* 5×2 Cross Validation

- Randomly split up labeled dataset into 2 parts of equal size
- Use one for training, the second one for testing (validation)
- Swap both sets
- Repeat 5 times (called folds)
- Analyze the classification error

⇒ Results in different 10 classifiers.

★ Nearest Neighbor Approach to Classification

⇒ The feature distribution is modeled by the training data (or a subset)

⇒ The class is assigned based on the closest feature in the training data.

Problem: Not scalable

- → Problematic in the presence of noise.
- → Problematic for classes/features with different variances.
- → Discriminant function is a Voronoi diagram

★ NN and K-NN Approach

NN

→ The feature distribution is modeled by the training data (or a Subset)

→ The class is assigned based on the closest feature in the training data.

K-NN

→ The k nearest neighbors are considered for the classification decision (majority vote or weighted).

→ Choise of K is often done heuristically.

(11b)

Classification (cont)

★ Decision Tree

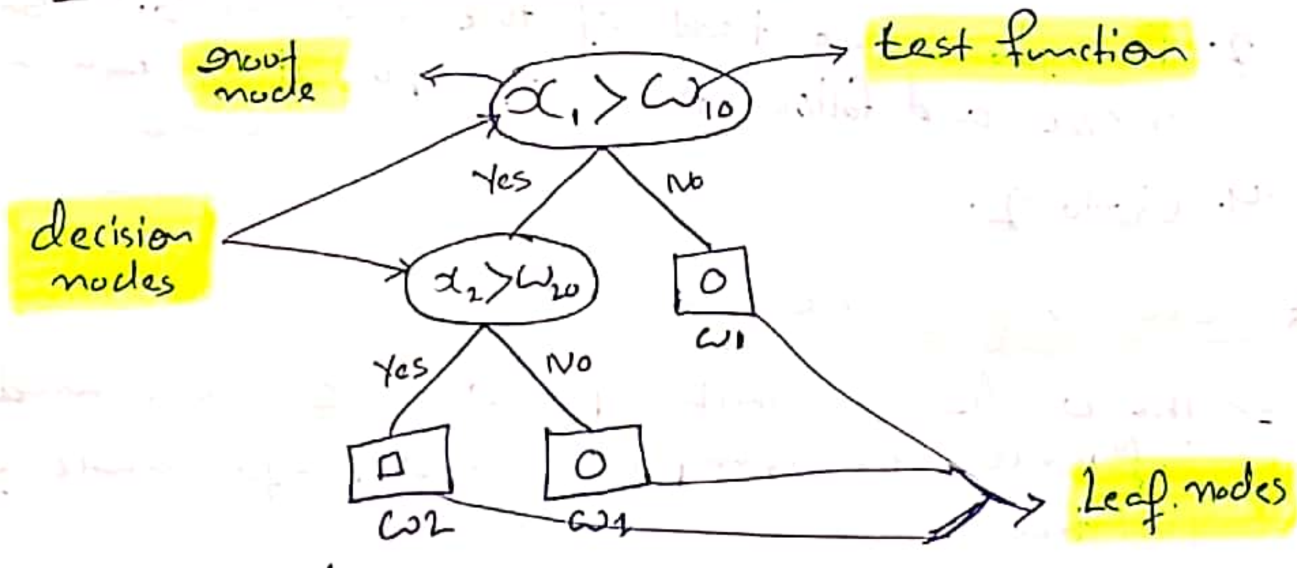
- Idea ⇒ Sequence of splits of the input space define regions that correspond to classes.

⇒ Hierarchical data structure realizing a divide and conquer strategy.

⇒ Setup of the tree through training data.

⇒ Efficient nonparametric method for classification (and regression)

★ Elements of a Decision Tree



★ Decision Nodes

⇒ Each decision node implements a test function with discrete outcomes.

⇒ The test function of each decision node splits the input space into regions.

⇒ Also called split node.

* Leaf Node

⇒ A leaf node Symbolizes the end of a Sequence of decisions.

⇒ A Single (output) Class is associated to leach leaf node

⇒ A leaf node defines a localized region in the input space where instances falling in this region have the Same label.

* Classification for a given decision Tree

1. Start at the root node.
2. If current node is a leaf node, return its class label.
3. Perform the test of the current decision node and follow the corresponding branch.
4. Goto 2.

* Learning a Decision Tree

⇒ The order in which split decisions are made influences the complexity and performance of the tree.

⇒ Finding the optimal arrangement of tests is NP hard, thus heuristics are needed.

* Which Decision to Make Next

⇒ Select the test that best separates the class labels in the data.

⇒ The "purer" the children, the better the split.

⇒ For any "pure" child branch, we can create a leaf node (no further splits).

* Impurity

⇒ Purity (or impurity) can be defined through the uncertainity in the distribution over the class label in the current vs the split-up region.

⇒ <u>Goal</u>: Always select the split that minimizes impurity.

⇒ Different impurity measures:
- Entropy
- Gini index

* Entropy as Impurity

↳ {Measure of uncertainity in probability distribution.}

$$H(\Omega) = -\sum_{\omega_i \in \Omega} P(\omega_i) \ln P(\omega_i)$$

⇒ Select the split that reduces the entropy at most:

$$\underbrace{\Delta H}_{\text{change in entropy}} = \underbrace{H(\Omega)}_{\text{entropy before split}} - \underbrace{\left(P_{left} H_{left}(\Omega) + (1 - P_{left}) H_{right}(\Omega)\right)}_{\text{entropy after split}}$$

★ Gini Index

→ Alternative Criterion to entropy.

→ Gini index

$$G(\Omega) = 1 - \sum_{\omega_i \in \Omega} p(\omega_i)^2$$

★ When to Stop?

- Intuitive idea: add a leaf note of a split leads to a pure node.

- Overfitting problem: The tree perfectly splits the classes on the training dataset but does not generalize well to other dataset.

⇒ Standard approach: Stop after a certain level of purity is reached.

⇒ A leaf stores the posterior probabilities of classes, instead of best label.

★ Decision Tree for Classification

→ Comparably easy to understand and implement.

→ Works well to high-dimensional data.

→ Allows to handle numerical and categorical Variables easily.

→ Finding the optimal split is NP hard.

→ Heuristics are used (e.g., entropy)

* Maximum-A Posteriori Classification

→ Classification is decision making

→ Probability theory as the framework for making decisions under uncertainity.

→ Based on Bayes' rule.

$$P(\omega|e) = \frac{P(e|\omega)\,P(\omega)}{P(e)}$$

Class

feature

a-priori Probability for the occurance of the class

"What is the probablity of a class given an observed feature"

(Probablity for the feature occurance can be obtained by:)

(Likelihood function for the class ω. It is also called observation model)

$$P(e) = \sum_{K=1}^{K} P(e|\omega_K)\,P(\omega_K)$$

$$\text{Posterior} = \frac{\text{likelihood} \times \text{Prior}}{\text{evidence}}$$

$$= \eta \times \text{likelihood} \times \text{Prior}$$

⇒ Distributions $P(e|\omega)$ and $P(\omega)$ must be learned from training data.

★ Map Classification

1. Compute for each class

$$P(\omega_i|e) = \frac{P(e|\omega_i)\,P(\omega_i)}{\sum_{k=1}^{K} P(e|\omega_k)P(\omega_k)}$$

2. Select the most likely class

ω_{i*} with $i^* = \arg\max_i P(\omega_i|e)$

★ MAP Classification with Gaussian Distributed Features

⇒ Let us look into features that follow a Gaussian given a class

$e = x \qquad x|\omega_i \sim g(\mu_i, \Sigma_i) \qquad i = 1, 2$

⇒ Thus, we can write

$$P(\omega_i|x_i) = \eta\, g(x, \mu_i, \Sigma_i)\, P(\omega_i)$$

⇒ and the negative log likelihood

$$-\ln P(\omega_i|x) = -\ln \eta - \ln g(x, \mu_i, \Sigma_i) - \ln P(\omega_i)$$

⇒ The MAP classifier directly yields

$$\arg\max_{\omega_i} P(\omega_i|x)$$

$$\Rightarrow \arg\min_{\omega_i} \left(-\ln g(x, \mu_i, \Sigma_i) - \ln P(\omega_i)\right)$$

⇒ and the classification boundary are points in which the function

$$D(x) = -\ln(P(\omega_i|x)) + \ln(P(\omega_2|x))$$

change its sign

⇒ The Shape of $D(x) = 0$ depends on the Eigen-values of $\Sigma_1^{-1}, \Sigma_2^{-1}$.

~~* Designing a Classifier: Training Phase~~

★ Learning a Classifier

→ In practice the distributions $P(e|\omega)$ and $P(\omega)$ are not known.

→ Both must be learned from data

★ Training data

- Sample set of size N.
- N_i Sample corresponds to class ω_i.

* Prior Distribution $(P(\omega))$

⇒ The prior distribution $P(\omega)$, which models the probability that a random sample corresponds to class ω_i is

$$P(\omega_i) = \frac{N_i}{N}$$

* The Likelihood Function

$$P(x_i|\omega_i) = g(x_i, \mu_i, \Sigma_i)$$

$$\mu_i = \frac{1}{N_i}\sum_{j=1}^{N_i} x_{ij}$$

$$\Sigma = \frac{1}{N_i - 1}\sum_{j=1}^{N_i} (x_{ij} - \mu_i)(x_{ij} - \mu_i)^T$$

Generative Approach → Use data to calculate the posterior densities that get the discriminant function.

Discriminative approach → Bypasses the estimation of densities & directly estimate the discriminants

———X———X———