CZ 4041 - Machine Learning

Suyash Lakhotia

AY 16/17, Semester 2

Contents

1 Introduction **5**
1.1 Definition . . . 5
1.2 Different Paradigms . . . 5
1.2.1 Supervised Learning . . . 5
1.2.2 Unsupervised Learning . . . 5
1.2.3 Semi-Supervised Learning . . . 5
1.2.4 Active Learning . . . 5
1.2.5 Transfer Learning . . . 5
1.2.6 Reinforcement Learning . . . 5

2 Overview of Supervised Learning **6**
2.1 Classification vs. Regression . . . 6
2.2 Typical Learning Procedure . . . 6
2.3 Evaluation of Supervised Learning . . . 6

3 Bayesian Decision Theory **7**
3.1 Bayesian Classifiers . . . 7
3.2 Math Review - Probability Concepts . . . 7
3.2.1 Decision with Priors . . . 7
3.3 Bayes Theorem . . . 7
3.3.1 General Bayesian Classifier . . . 8
3.4 Losses & Risks . . . 8
3.4.1 Formulae . . . 8
3.4.2 0/1 Loss . . . 8
3.4.3 Reject or Doubt . . . 8
3.4.4 Optimal Decision Rule . . . 8

4 Naive Bayes Classifier **10**
4.1 Bayesian Classifiers . . . 10
4.2 Math Review - Independence & Conditional Independence . . . 10
4.3 Naive Bayes Classifier . . . 10
4.3.1 Estimate Priors . . . 10
4.3.2 Estimate Conditional Probabilities for Discrete Features . . . 10
4.3.3 Estimate Conditional Probabilities for Continuous Features . . . 11
4.4 Laplace Estimate & M-Estimate . . . 11
4.5 Notes . . . 11

5 Bayesian Belief Networks **12**
5.1 BBN Directed Acyclic Graph . . . 12
5.2 Conditional Independence . . . 12
5.3 BNN Representation . . . 12
5.4 BNN Model Building . . . 12
5.5 Inference from BNN . . . 12
5.6 Notes . . . 12

6 Decision Tree **14**
6.1 Induction Algorithms . . . 14
6.2 Hunt's Algorithm . . . 14
6.3 Specifying the Split . . . 14
6.4 Determining the Best Split . . . 14
6.4.1 Best Split on Information Gain . . . 14
6.4.2 Gain Ratio . . . 15
6.5 Stopping Criteria for Tree Induction . . . 15
6.6 Notes . . . 15

7 Generalization **16**
7.1 Underfitting & Overfitting . . . 16
7.2 Generalization Error . . . 16
7.2.1 Occam's Razor . . . 16
7.2.2 Pessimistic Error Estimate . . . 16
7.2.3 Using a Validation Set . . . 16
7.3 Addressing Overfitting . . . 16
7.3.1 Pre-Pruning (Early Stopping Rule) . . . 16
7.3.2 Post-Pruning . . . 16

8 Artificial Neural Networks **17**
8.1 Perceptron . . . 17
8.1.1 Learning (Gradient Descent Approach) . . . 17
8.1.2 Notes . . . 18
8.2 Multilayer Neural Networks . . . 18
8.2.1 Integration Functions . . . 18
8.2.2 Activation Functions . . . 18
8.2.3 General Algorithm for Learning . . . 18
8.2.4 Backpropagation Algorithm . . . 18
8.3 Design Issues for ANN . . . 19

9 Support Vector Machines **20**
9.1 Maximum Margin . . . 20
9.2 Math Review . . . 20
9.2.1 Direction of Vectors . . . 20
9.2.2 Inner Product . . . 21
9.3 Linear SVM: Separable Case . . . 21
9.3.1 Direction of $\boldsymbol{w}$ . . . 21
9.3.2 Equations for Parallel Hyperplanes (i.e. Support Vectors) . . . 21
9.3.3 Deriving the Margin . . . 21
9.3.4 Learning in Linear SVM . . . 22
9.4 Linear SVM: Nonseperable Case . . . 22
9.4.1 Soft Error . . . 22
9.5 Non-linear SVM . . . 22
9.5.1 Kernel Trick . . . 23
9.5.2 Lagrange Multiplier Method . . . 23
9.5.3 Non-linear SVM via Kernel Trick . . . 24

10 Additional Notes for Multi-Class Classification Problems **25**

11 k-Nearest Neighbor Classifier **26**
11.1 Distance Metric . . . 26
11.2 Value of k . . . 26
11.3 Voting Approaches . . . 26
11.4 Normalization . . . 26
11.4.1 Min-Max Normalization . . . 26
11.4.2 Standardization . . . 26

11.5 Notes . . . 26

12 Model Evaluation 28
12.1 Precision . . . 28
12.2 Recall . . . 28
12.3 F_1-measure . . . 28
12.4 Receiver Operating Characteristic (ROC) . . . 28

13 Regression 30
13.1 Linear Regression Model . . . 30
13.1.1 Linear Regression Model Learning . . . 30
13.2 Math Review . . . 30
13.2.1 Matrix Transformation . . . 30
13.2.2 Identity Matrix . . . 30
13.3 Linear Regression Model (cont.) . . . 30
13.3.1 Linear Regression Model Learning . . . 31
13.4 Linear Basis Function Models . . . 31
13.4.1 Examples of Basis Functions . . . 31
13.4.2 Linear Basis Function Learning . . . 32
13.4.3 Limitations . . . 32
13.4.4 Kernelized Regression . . . 32
13.5 Evaluation . . . 32

14 Ensemble Learning 33
14.1 Why do ensembles work? . . . 33
14.1.1 Increased Accuracy . . . 33
14.1.2 Approximation by Ensemble Averaging . . . 33
14.2 Methods for Constructing Ensembles . . . 34
14.3 Structure of Ensemble Classifiers . . . 34
14.3.1 Parallel . . . 34
14.3.2 Cascading or Hierarchical . . . 34
14.4 Combination Strategies . . . 34
14.4.1 Static Combiners . . . 34
14.4.2 Adaptive Combiners . . . 35
14.5 Bagging . . . 35
14.6 Boosting . . . 36
14.6.1 AdaBoost . . . 36

15 Cluster Analysis 37
15.1 What is Cluster Analysis? . . . 37
15.1.1 Measurements & Distances . . . 37
15.1.2 Application of Clustering in Image Segmentation . . . 37
15.1.3 Types of Clusterings . . . 37
15.2 K-means Clustering . . . 37
15.2.1 Further Details . . . 37
15.2.2 Problems with Selecting Initial Points . . . 38
15.2.3 Evaluating K-means Clusters . . . 38
15.2.4 Limitations of K-means . . . 38
15.3 Hierarchical Clustering . . . 38
15.3.1 Strengths of Hierarchical Clustering . . . 38
15.3.2 Types of Hierarchical Clustering . . . 38
15.3.3 Agglomerative Hierarchical Clustering . . . 38
15.4 Measures of Cluster Validity . . . 39
15.4.1 Internal Measures . . . 39
15.4.2 External Measures . . . 40
15.5 Visual Object Classes Challenge . . . 40

16 Dimensionality Reduction **41**
16.1 Subset Selection . . . 41
16.2 Math Review - Eigenvalues & Eigenvectors . . . 41
16.3 Principal Component Analysis (PCA) . . . 41
16.3.1 Projecting the Data . . . 42
16.3.2 Dimensionality Reduction . . . 42
16.3.3 Choosing k . . . 42
16.4 Math Review - Spectral Decomposition . . . 43
16.5 Feature Embedding . . . 43
16.6 Math Review - Singular Value Decomposition . . . 43
16.7 Multidimensional Scaling . . . 43
16.8 Linear Discriminant Analysis . . . 43
16.8.1 Fisher's Linear Discriminant . . . 43
16.9 Notes . . . 44

17 Non-Parametric Density Estimation **45**
17.1 Histograms . . . 45
17.1.1 Pros/Cons . . . 45
17.2 General Theory for Non-Parametric Density Estimation . . . 45
17.3 Parzen Windows . . . 46
17.3.1 Drawbacks . . . 46
17.4 Smooth Kernels . . . 46
17.5 Bandwidth Selection . . . 47
17.5.1 Subjective Choice . . . 47
17.5.2 Reference to Standard Distribution . . . 47
17.5.3 Likelihood Cross-Validation . . . 47
17.6 Multivariate Density Estimation . . . 47
17.7 k Nearest Neighbor . . . 48
17.7.1 Disadvantages . . . 48
17.7.2 Approximation to Bayes Classifier . . . 48
17.8 Curse of Dimensionality . . . 48
17.9 Naive Bayes Classifier . . . 48

18 Graphical Models **49**
18.1 Factorization . . . 49
18.2 Independence . . . 49
18.2.1 Tail-to-Tail Node (Common Cause) . . . 49
18.2.2 Head-to-Tail Node (Causal Effect) . . . 49
18.2.3 Head-to-Head Node (Common Effect) . . . 50
18.3 Inference in Bayesian Networks . . . 50
18.3.1 Computational Complexity . . . 50
18.4 d-Separation . . . 50

Chapter 1

Introduction

1.1 Definition

Machine learning is a type of artificial intelligence that provides computers with the ability to learn from examples and/or experience without being explicitly programmed.

1.2 Different Paradigms

1.2.1 Supervised Learning

- The examples presented to the computer are pairs of inputs (i.e. features) and the corresponding outputs (i.e. labels). Labelled training data.
- If the labels are discrete, it is a **classification** problem.
- If the labels are continuous, it is a **regression** problem.
- The goal is to *learn* a **model** that maps inputs to the correct labels.

$$f : label = f(input)$$

1.2.2 Unsupervised Learning

- The examples presented to the computer are a set of inputs without any outputs. Unlabelled training data.
- The goal is to *learn* an **intrinsic structure** of the examples. For example, the goal in such problems may be to discover groups of similar examples within the data (i.e. clustering) or to determine the distribution of data within the input space (i.e. density estimation).

1.2.3 Semi-Supervised Learning

- The examples presented to the computer include both labelled data and unlabelled data.
- The goal is to the utilize the unlabelled data to help supervised learning.
- More precise `f`.

1.2.4 Active Learning

- The examples presented to the computer are unlabelled.
- An active learner (i.e. computer) may pose queries in the form of unlabelled examples to be labelled by an oracle (e.g. human annotator).

1.2.5 Transfer Learning

- The goal is to adapt the same learning to different tasks or environments.

1.2.6 Reinforcement Learning

- Learning by **interacting** with an environment to achieve a goal.
- The goal is to learn an optimal policy mapping states to actions.

Chapter 2

Overview of Supervised Learning

- In supervised learning, the examples presented to a computer are pairs of inputs and the corresponding outputs (i.e. labelled training data).
- The goal is to *learn* a **model** that maps inputs to the correct labels.
- Given a set of $\{\boldsymbol{x_i}, y_i\}$ (i.e. labelled training data) for $i = 1, 2, \ldots N$ where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, \ldots x_{im}]$ (i.e. a vector of features) and y_i (i.e. output) is a scalar, the goal is to learn a mapping $f : x \rightarrow y$ (i.e. classifier) by requiring $f(\boldsymbol{x_i}) = y_i$.
- The learned mapping f is expected to be able to make precise predictions on any unseen $\boldsymbol{x}^*$ as $f(\boldsymbol{x}^*)$.

2.1 Classification vs. Regression

- In classification problems, y is discrete.
  - If y is binary, then it is a binary classification problem.
  - If y is nominal but not binary, then it is a multi-class classification problem.
- In regression problems, y is continuous.

2.2 Typical Learning Procedure

Inductive Learning consists of two phases:

1. Training Phase
   - Given labelled training data, apply supervised learning algorithms to learn a model f such that $f(\boldsymbol{x_i}) = y_i$.
2. Testing or Prediction Phase
   - Given unseen test data $\boldsymbol{x_i^*}$, use the trained model f to make predictions for $f(\boldsymbol{x_i^*})$.

Lazy Learning involves taking an unseen test data $\boldsymbol{x_i^*}$ and using an algorithm to find the appropriate $\{\boldsymbol{x_i}, y_i\}$ pair from the training data to output a predicted label y^*.

2.3 Evaluation of Supervised Learning

- The learned predictive model should be able to make predictions on previously unseen data as accurately as possible.
- The whole training data is usually divided into two sets for training and testing.
- The training set is used to learn the predictive model and the test set is used to validate the performance of the learned model.

Common Evaluation Metrics:

- Classification: Accuracy, Error Rate, F-Score etc.
- Regression: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE)

Splitting Training Data to Disjoint Training & Evaluation Sets:

- Random Subsampling: Randomly sample a fraction of the data for evaluation and the rest for training. Repeat this process for several iterations.
- Cross Validation: Divide the entire dataset into `k` subsets of the same size. Hold aside one subset for evaluation and use the others to build the model. Traverse through all the subsets, changing the training subset at each iteration. For example, a five-fold cross validation will have five iterations.
- Bootstrap: Similar to random subsampling, however, the training set is sampled with replacement (i.e. duplicate samples allowed).

Chapter 3

Bayesian Decision Theory

3.1 Bayesian Classifiers

In many applications, the relationship between the input features and output labels is non-deterministic (i.e. uncertain). Bayesian classifiers aim to model probabilistic relationships between the input features and output class.

3.2 Math Review - Probability Concepts

Let X and Y be random variables.

Marginal probability refers to the probability that Y will take on the value y.

$$P(Y = y)$$

$$\sum_{y_i} P(Y = y_i) = 1$$

Joint probability refers to the probability that variable X will take on the value x and variable Y will take on the value y.

$$P(X = x, Y = y)$$

Conditional probability refers to the probability that the variable Y will take on the value y, given that the variable X is observed to have the value x.

$$P(Y = y|X = x)$$

$$\sum_{y_i} P(Y = y_i|X = x) = 1$$

Sum Rule — The joint and marginal probabilities for X and Y are related.

$$P(X = x) = \sum_{y_i} P(X = x, Y = y_i)$$

$$P(X) = \sum_{Y} P(X, Y)$$

$$P(X = x) = \sum_{z_j} \sum_{y_i} P(X = x, Y = y_i, Z = z_j)$$

$$P(X) = \sum_{Z} \sum_{Y} P(X, Y, Z)$$

Product Rule — The joint and conditional probabilities for X and Y are related.

$$\begin{aligned} P(X = x, Y = y) &= P(Y = y|X = x) \times P(X = x) \\ &= P(X = x|Y = y) \times P(Y = y) \end{aligned}$$

$$P(X, Y) = P(Y|X) \times P(X) = P(X|Y) \times P(Y)$$

3.2.1 Decision with Priors

A decision rule prescribes what action to take based on observed input.

Predict $Y = y_i$ if:

$$P(Y = y_i) = \max_k P(Y = y_k)$$

3.3 Bayes Theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- $P(Y|X)$: Posterior Probability
- $P(X|Y)$: Likelihood
- $P(Y)$: Prior Probability
- $P(X)$: Evidence

3.3.1 General Bayesian Classifier

Predict $Y = y_i$ if:

$$P(Y = y_i|\boldsymbol{X} = \boldsymbol{x}) = \max_k P(Y = y_k|\boldsymbol{X} = \boldsymbol{x})$$

Applying Bayes Theorem:

$$P(Y = y_k|\boldsymbol{X} = \boldsymbol{x}) = \frac{P(\boldsymbol{X} = \boldsymbol{x}|Y = y_k)P(Y = y_k)}{P(\boldsymbol{X} = \boldsymbol{x})}$$

Since $P(\boldsymbol{X} = \boldsymbol{x})$ is constant w.r.t to different values of Y, predict $Y = y_i$ if:

$$P(\boldsymbol{X}|Y = y_i)P(Y = y_i) = \max_k P(\boldsymbol{X}|Y = y_k)P(Y = y_k)$$

3.4 Losses & Risks

So far, the decisions are assumed to be equally good or costly. However, the cost of misclassification for different classes may be different. For example, misdiagnosing a healthy patient as ill versus misdiagnosing an ill patient as healthy.

3.4.1 Formulae

- Let a_i be the action of predicting $Y = y_i$.
- Let λ_{ik} be the loss of a_i when the class value is actually y_k.

Expected Risk for Action a_i:

$$R(a_i|X) = \sum_{k=1}^{K} \lambda_{ik} P(Y = y_k|X)$$

Choose a_i if:

$$R(a_i|X) = \min_k R(a_k|X)$$

3.4.2 0/1 Loss

$$\lambda_{ik} = \begin{cases} 0 & i = k \\ 1 & i \neq k \end{cases}$$

If all correct decisions have no loss and all errors are equally costly:

$$\begin{aligned} R(a_i|X) &= \sum_{k=1}^{K} \lambda_{ik} P(Y = y_k|X) \\ &= \sum_{k \neq i} P(Y = y_k|X) \\ &= 1 - P(Y = y_i|X) \end{aligned}$$

3.4.3 Reject or Doubt

If the automatic system has low certainty of its decision and making a wrong decision may have very high cost, a manual decision is required.

The solution is to define an additional action for *reject* or *doubt*, a_{K+1}, with a_i, $i = 1, 2, \ldots K$, being the usual actions of predicting on classes y_i.

The revised 0/1 loss function is now:

$$\lambda_{ik} = \begin{cases} 0 & i = k \\ \theta & i = K + 1 \\ 1 & i \neq k \end{cases}$$

where $0 < \theta < 1$ is the loss incurred for choosing the reject action.

The expected risk of taking the reject action:

$$R(a_{K+1}|X) = \sum_{k=1}^{K} \theta P(Y = y_k|X) = \theta$$

3.4.4 Optimal Decision Rule

The following rules are meaningful if $0 < \theta < 1$. If $\theta = 0$, we will always reject and if $\theta \geq 1$, we will never reject.

3.4.4.1 Decision Rule 1

Take action a_i if the following conditions are true:

$$R(a_i|X) < R(a_k|X), \forall k|k \neq i$$
$$R(a_i|X) < R(a_{K+1}|X)$$

Choose reject if:

$$R(a_{K+1}|X) < R(a_i|X), i = 1, 2, \ldots K$$

3.4.4.2 Decision Rule 2

Predict $Y = y_i$ if the following conditions are true:

$$P(Y = y_i|X) > R(Y = y_k|X), \forall k|k \neq i$$
$$P(Y = y_i|X) > 1 - \theta$$

Choose reject otherwise.

Chapter 4

Naive Bayes Classifier

4.1 Bayesian Classifiers

Given a vector of features $\boldsymbol{X}$ and a class label $Y = y_k$, $\boldsymbol{X}$ and Y are treated as random variables and their relationship is captured using $P(Y|\boldsymbol{X})$ using training data.

A test record $\boldsymbol{X} = \boldsymbol{x^*}$ or $\boldsymbol{X^*}$ can be classified by finding the class y_i that maximizes the posterior $P(Y|\boldsymbol{X^*})$. Predict $Y = y_i$ if:

$$
\begin{aligned}
y_i &= \underset{k}{\operatorname{argmax}}\, P(Y = y_k|\boldsymbol{X^*}) \\
y_i &= \underset{k}{\operatorname{argmax}}\, P(\boldsymbol{X^*}|Y = y_k)P(Y = y_k)
\end{aligned}
$$

4.2 Math Review - Independence & Conditional Independence

Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two sets of random variables.

The variables in $\boldsymbol{X}$ are said to be **independent** of $\boldsymbol{Y}$ if the following condition holds:

$$P(\boldsymbol{X}, \boldsymbol{Y}) = P(\boldsymbol{X}|\boldsymbol{Y}) \times P(\boldsymbol{Y}) = P(\boldsymbol{X}) \times P(\boldsymbol{Y})$$

Let $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ be three sets of random variables.

The variables in $\boldsymbol{X}$ are said to be **conditionally independent** of $\boldsymbol{Y}$, given $\boldsymbol{Z}$, if the following condition holds:

$$P(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{Z}) = P(\boldsymbol{X}|\boldsymbol{Z})$$

The conditional independence between $\boldsymbol{X}$ and $\boldsymbol{Y}$ given $\boldsymbol{Z}$ can be used to derive the following:

$$
\begin{aligned}
P(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z}) &= \frac{P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})}{P(\boldsymbol{Z})} \\
&= \frac{P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})}{P(\boldsymbol{Y}, \boldsymbol{Z})} \times \frac{P(\boldsymbol{Y}, \boldsymbol{Z})}{P(\boldsymbol{Z})} \\
&= P(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{Z}) \times P(\boldsymbol{Y}|\boldsymbol{Z}) \\
&= P(\boldsymbol{X}|\boldsymbol{Z}) \times P(\boldsymbol{Y}|\boldsymbol{Z})
\end{aligned}
$$

4.3 Naive Bayes Classifier

Assuming that the features in $\boldsymbol{X}$ are conditionally independent given the class label:

$$P(\boldsymbol{X}|Y = y_k) = \prod_{i=1}^{d} P(X_i|Y = y_k)$$

$$P(X_1, X_2, \ldots X_d|Y = y_k) = \prod_{i=1}^{d} P(X_i|Y = y_k)$$

To classify a test record $\boldsymbol{X^*}$, the posteriors for each class y_k need to be computed using:

$$P(Y = y_k|\boldsymbol{X^*}) = \frac{P(Y = y_k) \prod_{i=1}^{d} P(X_i^*|Y = y_k)}{P(\boldsymbol{X^*})}$$

Since $P(\boldsymbol{X^*})$ is fixed for each y_k, it is sufficient to choose the class that maximizes the numerator:

$$\max_k P(Y = y_k) \prod_{i=1}^{d} P(\boldsymbol{X_i^*}|Y = y_k)$$

4.3.1 Estimate Priors

$$P(Y = y_k) = \frac{|Y = y_k|}{N}$$

where N is the total number of training examples.

4.3.2 Estimate Conditional Probabilities for Discrete Features

$$P(X_i = x_{ij}|Y = y_k) = \frac{|(X_i = x_{ij}) \cap (Y = y_k)|}{|Y = y_k|}$$

where x_{ij} is the jth distinct value of the feature X_i.

4.3.3 Estimate Conditional Probabilities for Continuous Features

- Assume the values of a feature given a specific y_k follows a Gaussian distribution i.e. $P(X_i|Y = y_k)$ is a Gaussian distribution.
- Use the training data to estimate the mean (μ) and variance (σ^2) of the distribution.

$$P(X_i|Y = y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$

4.4 Laplace Estimate & M-Estimate

If one of the conditional probabilities is zero, then the entire expression becomes zero. Therefore, an alternative probability estimation is needed.

Laplace Estimate:

$$P(X_i = x_{ij}|Y = y_k) = \frac{|(X_i = x_{ij}) \cap (Y = y_k)| + 1}{|Y = y_k| + n_i}$$

where n_i is the number of possible values for X_i.

M-Estimate:

$$P(X_i = x_{ij}|Y = y_k) = \frac{|(X_i = x_{ij}) \cap (Y = y_k)| + m \times p}{|Y = y_k| + m}$$

where m and p are user-specified parameters based on prior information of $P(X_i = x_{ij}|Y = y_k)$.

4.5 Notes

- Naive Bayes Classifier is computationally efficient.
- The independence assumption may not hold for some features.
- Correlated features can degrade the performance of Naive Bayes Classifiers.

Chapter 5

Bayesian Belief Networks

When two features are correlated, the assumption for Naive Bayes Classifier does not hold true.

$$P(X_1, X_2|Y) \neq P(X_1|Y) \times P(X_2|Y)$$

BBNs provide a more flexible approach for modeling the likelihood probabilities $P(\boldsymbol{X}|Y)$. A Bayesian network provides a graphical representation of the probabilistic relationships among a set of random variables.

There are two key elements:

1. A directed acyclic graph (DAG) encoding the dependence relationships among a set of variables.
2. A probability table associating each node to its immediate parent nodes.

5.1 BBN Directed Acyclic Graph

Considering three random variables A, B & C, where A and B are independent variables and each has a direct influence on C:

- There will be a directed arc from A to C and B to C in the DAG.
- A & B are parents of C.
- C is the child of A & B.

5.2 Conditional Independence

- A node in a Bayesian network is **conditionally independent** of its non-descendants, if its parents are known.
- A Naive Bayes Classifier can be represented using a Bayesian network where Y is the parent to all the features in $\boldsymbol{X}$.

5.3 BNN Representation

Besides the conditional independence conditions imposed by the network topology, each node is also associated with a probability table.

- If a node X does not have any parents, then the table contains only the prior probability $P(X)$.
- If a node X has only one parent Z, then the table contains the conditional probability $P(X|Z)$.
- If a node X has multiple parents $\{Z_1, Z_2, \dots Z_k\}$, then the table contains the conditional probability $P(X|Z_1, Z_2, \dots Z_k)$.

5.4 BNN Model Building

- Step 1: Create the structure of the network.
  - Network topology can be obtained by encoding the subjective knowledge of domain experts;
  - Or can be learned from data (structure learning).
- Step 2: Estimate the probability values in the table associated with each node.

5.5 Inference from BNN

Given a BBN and an inference problem:

1. Translate the problem into a probabilistic expression.
2. Based on the property on conditional independence, find all dependent variables to the probabilities being estimated.
3. If the probabilities to be estimated cannot be obtained directly from the probability tables of the BBN, apply the Sum Rule, Product Rule and/or Bayes Theorem to decompose the probabilities until all decomposed probabilities can be obtained from the tables.

5.6 Notes

- BBN provides an approach for capturing the prior knowledge of a specific domain using a directed graphical model.
  - Other Examples of Directed Graphical Models: Hidden Markov Models, Dynamic Bayesian Networks etc.
  - Other Examples of Undirected Graphical Models: Markov Random Fields, Condition Random Fields etc.

- Network construction, however, is time consuming as it requires domain knowledge or a system that can learn structure from data.

Chapter 6

Decision Tree

A decision tree is used to query particular features at particular points in the tree to come to a decision about the classification of the current $\boldsymbol{X}$. The tree has to be *induced* from the training data and can then be used to *deduce* classifications.

6.1 Induction Algorithms

A tree can be learned by splitting training data into subsets based on outcomes of a feature test. This process is recursively applied on each derived subset until the subset at a node has the same value for the target variable or there is no improvement for prediction.

6.2 Hunt's Algorithm

Hunt's algorithm grows a decision tree in a recursive fashion by partitioning the training records into successively purer subsets. A pure subset is one where the label (i.e. value of Y) is the same for all records.

Let D_t be the set of training records that reach a node t.

- If D_t contains records that belong to the same class y_t, then t is a leaf node labelled as y_t.
- Else, if D_t is an empty set, then t is a leaf node labelled by the default class y_d.
- Else, if D_t contains records that belong to more than one class of Y, then a feature is selected to conduct a conditional test to split the data into smaller subsets.
  - A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcome.
  - Recursively apply this procedure to each subset.

6.3 Specifying the Split

- Splitting based on binary features will split the D_t into two subsets.
- Splitting based on categorical features can be done in one of two ways:
  - A multi-way split uses as many partitions as distinct values.
  - A binary split divides D_t into two subsets for two distinct sets of possible values of the feature.
- Splitting based on continuous features can be done in one of two ways:
  - A binary split divides D_t into two subsets for two distinct ranges of possible values of the feature. $(X_j < v)$ or $(X_j \geq v)$.
  - A multi-way split divides D_t into multiple distinct ranges of possible values of the feature. May be very computationally intensive.

6.4 Determining the Best Split

Intuitive Idea: Nodes with **homogeneous** class distribution is preferred.

A split criterion can be defined in terms of the difference in degrees of node impurity before and after splitting. This impurity can be measured using the entropy at a given node t:

$$\text{Entropy}(t) = -\sum_{y_k} P(Y = y_k|t) \log_2 P(Y = y_k|t)$$

- Maximum: $\log_2 K$ where K is the total number of all possible values of Y i.e. records are equally distributed among all classes i.e. least information.
- Minimum: 0 i.e. when all records belong to one class i.e. most information.

6.4.1 Best Split on Information Gain

Suppose a parent node t is split into P partitions, the information gain:

$$\delta_{\text{info}} = \text{Entropy}(t) - \sum_{j=1}^{k} \frac{n_j}{n} \text{Entropy}(i)$$

where n_j represents the number of examples at child j and n represents the number of examples at node t.

The feature test that maximizes the information gain must be chosen as it minimizes the weighted average impurity measures of the child nodes.

However, a disadvantage is that splitting based on entropy tends to prefer splits that result in a large number of partitions, each being small but pure.

6.4.2 Gain Ratio

$$\delta_{\text{InfoR}} = \frac{\delta_{\text{info}}}{\text{SplitINFO}}$$

$$\text{SplitINFO} = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

where n is total number of records in the parent node t and n_i is the number of records in partition i.

By using the gain ratio to adjust information gain, higher entropy partitioning (i.e. large number of small partitions) is penalized.

6.5 Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class.
- Stop expanding a node when all the records have similar attribute values.
- Use early termination.

6.6 Notes

- Decision tree classifiers are easy to interpret and efficient in both, training & testing.
- They are used as a base classifier in many ensemble learning approaches.

Chapter 7

Generalization

7.1 Underfitting & Overfitting

- Underfitting occurs when the model is too simple, and hence, both training and test error rates are large.
- Overfitting occurs when the test error rate begins to increase even though the training error rate continues to decrease.
  - Overfitting results in decision trees that are more complex than necessary.
  - The training error no longer provides a good estimate of how well the tree will perform on previously unseen records. Therefore, there is a need for new ways to estimate the error rate.

7.2 Generalization Error

Generalization error is a measure of how accurately an algorithm is able to predict outcome values of previously unseen data. A model with the ideal complexity is the one that produces the lowest generalization error. Since there is no knowledge of the test data and how well the model will perform, the generalization error of the induced model needs to be estimated.

7.2.1 Occam's Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model.
- For complex models, there is a greater chance that it was fitted accidentally by errors in data. Therefore, one should include model complexity when evaluating a model.

7.2.2 Pessimistic Error Estimate

- Explicitly compute the generalization error as the sum of training error and a penalty term for model complexity.

$$e'(T) = e(T) + \Omega(T)$$

In a decision tree, we can define a penalty term of $k > 0$ on each leaf node:

$$e'(T) = e(T) + N \times k$$

where N is the total number of leaf nodes.

7.2.3 Using a Validation Set

- Divide the original training dataset into two subsets — one for training and the other (i.e. validation set) is for estimating the generalization error.
- The complexity of the best model can be estimated using the performance of the model on the validation set.

7.3 Addressing Overfitting

7.3.1 Pre-Pruning (Early Stopping Rule)

- Stop the algorithm before it becomes a fully-grown tree.
- Typical stopping conditions for a node:
  - Stop if all instances belong to the same class.
  - Stop if all the feature values are the same.
- More restrictive stopping conditions for a node:
  - Stop if the number of instances is less than some user-specified threshold.
  - Stop if the class distribution of the instances is independent of the available features.
  - Stop if expanding the current node does not improve impurity measures.

7.3.2 Post-Pruning

- Grow decision to its entirety and trim the nodes in a bottom-up fashion.
- If generalization error improves after trimming, replace sub-tree with a new leaf node. The class label of this leaf node is determined from the majority class of the instances in the sub-tree.

Chapter 8

Artificial Neural Networks

The human brain is a densely interconnected network of neurons, connected to others via axons. Axons are used to transmit nerve impulses from one neuron to another. The human brain learns by changing the strength of the synaptic connection between neurons. An Artificial Neural Network is composed of an interconnected assembly of nodes and directed links.

8.1 Perceptron

- Every input node is connected via a weighted link to the output node. Weights can be positive, negative or zero (no connection).
- The model is an assembly of interconnected nodes and weighted links.
- Output node first sums up each of its input value according to the weights of its links. The weighted sum is compared against some threshold θ and an output is produced based on the sign of the result.

$$\text{sign}(z) \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$y = \text{sign}\left(\sum_{i=1}^{d} w_i X_i - \theta \right)$$

Alternatively,

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x})$$

where:

$$\begin{aligned} \boldsymbol{w} &= (w_0, w_1, \dots w_d) \\ \boldsymbol{x} &= (X_0, X_1, \dots X_d) \\ w_0 &= -\theta \text{ and } X_0 = 1 \end{aligned}$$

8.1.1 Learning (Gradient Descent Approach)

During training, the weight parameters $\boldsymbol{w}$ are adjusted until the outputs of the perceptron become consistent with the true outputs of the training data. The weight parameters $\boldsymbol{w}$ are updated iteratively with every training example using the gradient descent approach.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^t - \lambda \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

where $\lambda \in (0, 1]$ is the learning rate and $E(\boldsymbol{w})$ is the error function to be minimized.

$$\boldsymbol{w}^* = \min_{\boldsymbol{w}} E(\boldsymbol{w})$$

Let h_i be the predicted output for $\boldsymbol{x}_i$ and consider the loss function for each training example as:

$$\begin{aligned} E &= \frac{1}{2}(y_i - h_i)^2 \\ &= \frac{1}{2}(y_i - \text{sign}(\boldsymbol{w}^{(t)} \cdot x_i))^2 \end{aligned}$$

The weight update rule can be rewritten as:

$$\begin{aligned} \boldsymbol{w}^{t+1} &= \boldsymbol{w}^t - \lambda \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} \\ &= \boldsymbol{w}^t - \lambda \frac{\partial E(h)}{\partial h} \frac{\partial h(z)}{\partial z} \frac{\partial z(\boldsymbol{w})}{\partial \boldsymbol{w}} \end{aligned}$$

where:

$$\begin{aligned} h &= \text{sign}(z) \\ z &= \boldsymbol{w} \cdot \boldsymbol{x} \end{aligned}$$

Thus, after solving:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \lambda(y_i - h_i)\boldsymbol{x}_i$$

8.1.1.1 Learning Model

- If the prediction is correct, $(y - h) = 0$, then the weight remains unchanged i.e. $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)}$.
- If $y = +1$ and $h = -1$, then $(y - h) = 2$.
  - The weights of all links with positive inputs need to be updated by increasing their values.
  - The weights of all links with negative inputs need to be updated by decreasing their weights.
- If $y = -1$ and $h = +1$, then $(y - h) = -2$.
  - The weights of all links with positive inputs need to be updated by decreasing their values.
  - The weights of all links with negative inputs need to be updated by increasing their weights.

8.1.2 Notes

- The decision boundary of a perceptron is a linear hyperplane.
- The perceptron learning algorithm is guaranteed to converge to an optimal solution for linear classification problems.
- However, if the problem is not linearly separable, the algorithm fails to converge.

8.2 Multilayer Neural Networks

- Multilayer ANNs are feed-forward neural networks i.e. the nodes in one layer are only connected to the nodes in the next layer.
- Each node takes in the inputs and passes it through an **integration function** followed by an **activation function**.

8.2.1 Integration Functions

Weighted Sum:

$$\sum_{i=1}^{d} w_i X_i - \theta$$

Quadratic Function:

$$\sum_{i=1}^{d} w_i X_i^2 - \theta$$

Spherical Function:

$$\sum_{i=1}^{d} (X_i - w_i)^2 - \theta$$

8.2.2 Activation Functions

Sign Function (Threshold Function):

$$a(z) = \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Unipolar Sigmoid Function:

$$a(z) = \frac{1}{1 + e^{-\lambda z}}$$

When $\lambda = 1$, it is called the sigmoid function.

8.2.3 General Algorithm for Learning

Just like the perceptron, the learning algorithm for multilayer ANNs starts with initializing the weights of the links and adjusting these weights such that the output of the ANN is consistent with the class labels of the training examples.

Loss function for each training instance:

$$E = \frac{1}{2}(y_i - h_i)^2$$

The objective is to find the weights w'_i that minimize the error function:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^t - \lambda \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

However, in multilayer ANNs, the errors cannot be computed directly for the hidden nodes.

8.2.4 Backpropagation Algorithm

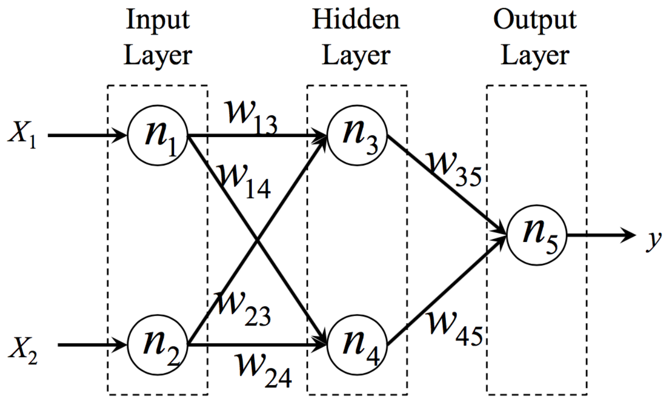


Figure 8.1: Multilayer ANN

After initializing the weights, the training examples are *forward passed* through the neural network to generate outputs.

Backpropagation involves starting with the output layer to propagate error back to the previous layer in order to update the weights between the two layers. Since it is not possible to calculate the error in the hidden layers, the error is decomposed using the weights and propagated back when a hidden layer is reached.

For example, in Figure 8.1, E_5 or the error in n_5 will be decomposed using the weights and propagated back. Thus, the value of E_3 will be $E_5 \times w_3 5$ and the value of E_4 will be $E_5 \times w_4 5$.

Revisiting the gradient descent rule:

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} + \lambda (y_i - h_i) \frac{\partial a(z)}{\partial z} \boldsymbol{x}_i \\ &= \boldsymbol{w}^{(t)} + \lambda E_i \frac{\partial a(z)}{\partial z} \boldsymbol{x}_i \\ &= \boldsymbol{w}^{(t)} + \lambda \Delta_i \boldsymbol{x}_i \end{aligned}$$

where $a(z)$ is the activation function, Δ_i is the modified error and $\boldsymbol{x}_i$ is the output of the previous layer.

For example, in Figure 8.1, assuming the activation function is the sign function:

$$\frac{\partial a(z)}{\partial z} = 1$$

Then:

$$\boldsymbol{w}_{35}^{(t+1)} = \boldsymbol{w}_{35}^{(t)} + \lambda E_5 o_3$$

$$\boldsymbol{w}_{45}^{(t+1)} = \boldsymbol{w}_{45}^{(t)} + \lambda E_5 o_4$$

$$\begin{aligned} \boldsymbol{w}_{13}^{(t+1)} &= \boldsymbol{w}_{13}^{(t)} + \lambda E_3 o_1 \\ &= \boldsymbol{w}_{13}^{(t)} + \lambda (\boldsymbol{w}_{35}^{(t+1)} \times E_5) o_1 \end{aligned}$$

$$\begin{aligned} \boldsymbol{w}_{14}^{(t+1)} &= \boldsymbol{w}_{14}^{(t)} + \lambda E_4 o_1 \\ &= \boldsymbol{w}_{14}^{(t)} + \lambda (\boldsymbol{w}_{45}^{(t+1)} \times E_5) o_1 \end{aligned}$$

$$\begin{aligned} \boldsymbol{w}_{24}^{(t+1)} &= \boldsymbol{w}_{24}^{(t)} + \lambda E_4 o_2 \\ &= \boldsymbol{w}_{24}^{(t)} + \lambda (\boldsymbol{w}_{45}^{(t+1)} \times E_5) o_2 \end{aligned}$$

where o_i is the output of n_i.

8.3 Design Issues for ANN

- The number of nodes in the input layer.
  - Assign an input node to each numerical or binary input variable.
- The number of nodes in the output layer.
  - Binary class problem → single node
  - k-class problem → k output nodes
- Training examples with missing values should be removed or replaced with most likely values.

Chapter 9

Support Vector Machines

SVMs are based on statistical learning theory that has shown promising empirical results in many practical applications, such as computer vision, sensor networks and text mining. The basic idea behind SVMs is the **maximum margin hyperplane**.

9.1 Maximum Margin

The goal is to learn a binary classifier by finding a hyperplane (decision boundary) such that all the squares reside on one side of the hyperplane and all the circles reside on the other side. Although there are many hyperplanes that can separate the training examples perfectly, their generalization errors may be different.

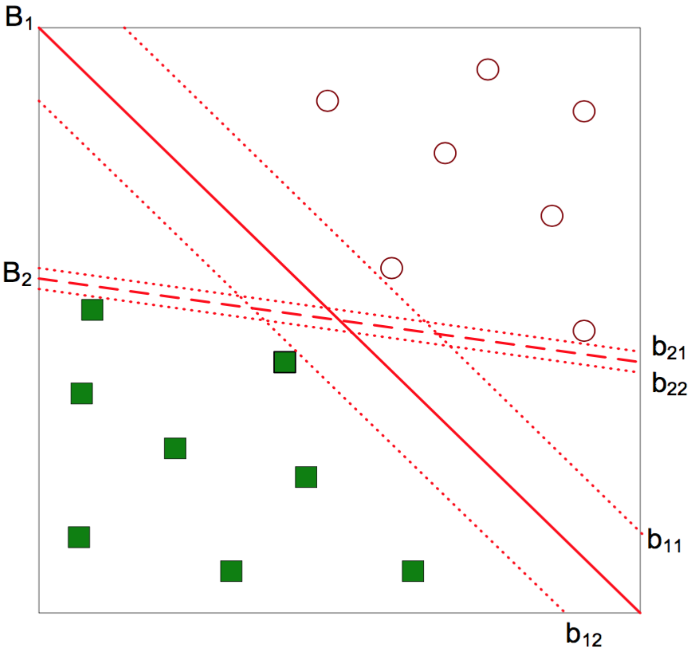


Figure 9.1: Maximum Margin Example (2)

Each decision boundary B_1 is associated with a pair of parallel hyperplanes — b_{i1} and b_{i2}. b_{i1} is obtained by moving the hyperplane until it touches the closest circle(s) and b_{i2} is obtained by moving a hyperplane away from the decision boundary until it touches the closest square(s).

The assumption is that larger margins imply better generalization errors. Since the margin of B_1 is larger than the margin of B_2, B_1 is better than B_2.

9.2 Math Review

9.2.1 Direction of Vectors

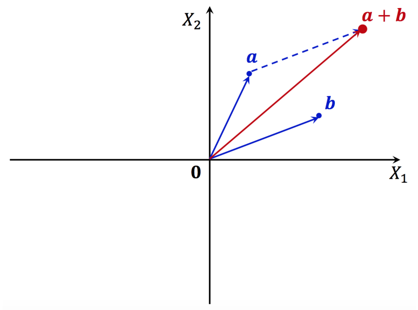


Figure 9.2: $\boldsymbol{a}+\boldsymbol{b}$

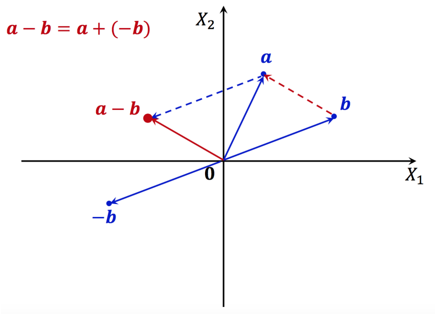

$\boldsymbol{a}-\boldsymbol{b}=\boldsymbol{a}+(-\boldsymbol{b})$
X_2
$\boldsymbol{a}$
$\boldsymbol{a}-\boldsymbol{b}$
$\boldsymbol{b}$
0
X_1
$-\boldsymbol{b}$

Figure 9.3: $\boldsymbol{a}-\boldsymbol{b}$

9.2.2 Inner Product

The inner product of two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$ of d dimensions is defined as:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{d} (u_i \times v_i)$$

From a geometry viewpoint:

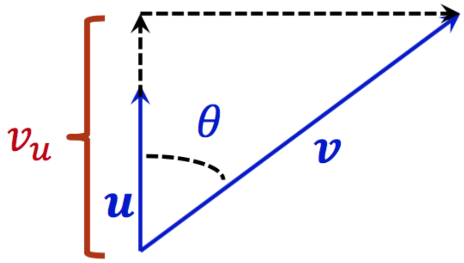


Figure 9.4: $\boldsymbol{u}$ & $\boldsymbol{v}$

$$\boldsymbol{u} \cdot \boldsymbol{v} = ||\boldsymbol{u}||_2 \times ||\boldsymbol{v}||_2 \times \cos(\theta)$$

$$||\boldsymbol{u}||_2^2 = \boldsymbol{u} \cdot \boldsymbol{u} = \sum_{i=1}^{d} (u_i \times u_i)$$

Thus, when $\boldsymbol{u} \cdot \boldsymbol{v} = 0$, $\theta = 90°$ i.e. $\boldsymbol{u}$ and $\boldsymbol{v}$ are orthogonal.

The value of v_u can be derived using the inner product:

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{v} &= ||\boldsymbol{u}||_2 \times ||\boldsymbol{v}||_2 \times \cos(\theta) \\ &= ||\boldsymbol{u}||_2 \times (||\boldsymbol{v}||_2 \times \cos(\theta)) \\ &= ||\boldsymbol{u}||_2 \times v_u \end{aligned}$$

where v_u represents the length of $\boldsymbol{v}$ in the direction of $\boldsymbol{u}$.

9.3 Linear SVM: Separable Case

Given a binary classification task, denoting $y_i = +1$ as the circle class and $y_i = -1$ as the square class, the decision boundary is defined as:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$$

For any test example $\boldsymbol{x}^*$:

$$\boldsymbol{x}^* : \begin{cases} f(\boldsymbol{x}^*) = +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ f(\boldsymbol{x}^*) = -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$

9.3.1 Direction of $\boldsymbol{w}$

Suppose $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ are two points on the decision boundary:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{x_a} + b = 0 \\ \boldsymbol{w} \cdot \boldsymbol{x_b} + b = 0 \end{aligned}$$

$$\boldsymbol{w} \cdot (\boldsymbol{x_a} - \boldsymbol{x_b}) = 0$$

Since $(\boldsymbol{x_a} - \boldsymbol{x_b})$ is a vector on the decision boundary, based on the definition of inner product, the direction of $\boldsymbol{w}$ is orthogonal to the decision boundary.

9.3.2 Equations for Parallel Hyperplanes (i.e. Support Vectors)

For any circle $\boldsymbol{x_c}$ located above the decision boundary:

$$\boldsymbol{w} \cdot \boldsymbol{x_c} + b = k, \text{ where } k > 0$$

For any square $\boldsymbol{x_s}$ located below the decision boundary:

$$\boldsymbol{w} \cdot \boldsymbol{x_s} + b = k', \text{ where } k' < 0$$

The two parallel hyperplanes passing the closest circle(s) and square(s) can be written as:

$$\begin{gathered} \boldsymbol{w} \cdot \boldsymbol{x_c} + b = \bar{k} \\ \boldsymbol{w} \cdot \boldsymbol{x_s} + b = -\bar{k} \\ \text{where } \bar{k} > 0 \end{gathered}$$

After rescaling $\boldsymbol{w}$ and b, the two parallel hyperplanes can be further rewritten as:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{x_c} + b = +1 \\ \boldsymbol{w} \cdot \boldsymbol{x_s} + b = -1 \end{aligned}$$

9.3.3 Deriving the Margin

Let $\boldsymbol{x_1}$ be a point on b_{11} and $\boldsymbol{x_2}$ be a point on b_{12}:

$$\begin{aligned} b_{11} : \boldsymbol{w} \cdot \boldsymbol{x_1} + b = +1 \\ b_{12} : \boldsymbol{w} \cdot \boldsymbol{x_2} + b = -1 \end{aligned}$$

$$\boldsymbol{w} \cdot (\boldsymbol{x_1} - \boldsymbol{x_2}) = 2$$

Based on the definition of inner product:

$$||\boldsymbol{w}||_2 \times d = 2$$

$$d = \frac{2}{||\boldsymbol{w}||_2}$$

where d represents the margin i.e. $(||\boldsymbol{x_1} - \boldsymbol{x_2}||_2 \times \cos(\theta))$.

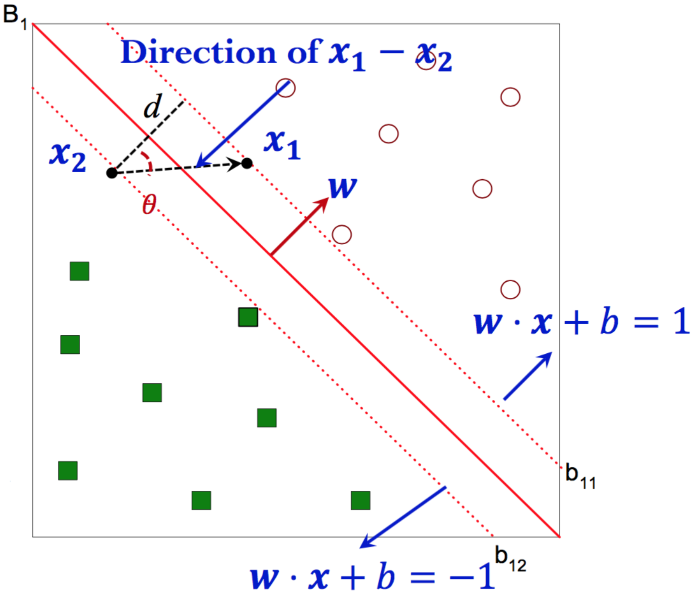


Figure 9.5: Deriving the Margin

9.3.4 Learning in Linear SVM

The goal is to maximize the margin d, or in other words, minimize the following:

$$\min_{\boldsymbol{w}} \frac{||\boldsymbol{w}||_2^2}{2}$$

Given the following constraints:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{x_i} + b &\geq +1, \text{ if } y_i = 1 \\ \boldsymbol{w} \cdot \boldsymbol{x_i} + b &\leq -1, \text{ if } y_i = -1 \end{aligned}$$

Or, in other words:

$$y_i \times (\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1$$

Thus, the optimization problem of linear SVM is:

$$\min_{\boldsymbol{w}} \frac{||\boldsymbol{w}||_2^2}{2}$$
$$\text{s.t. } y_i \times (\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1, \; i = 1, 2, \ldots N$$

The optimization is convex and there are many numerical approaches that can be applied to solve it.

9.4 Linear SVM: Nonseperable Case

In seperable cases, there is no training data within the margin. However, in nonseperable cases, there may be training data that lies within the margin. Thus, slack variables $\xi \geq 0$ need to be introduced to absorb errors.

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{x_i} + b &\geq +1 - \xi_i, \text{ if } y_i = 1 \\ \boldsymbol{w} \cdot \boldsymbol{x_i} + b &\leq -1 + \xi_i, \text{ if } y_i = -1 \end{aligned}$$

Or, in other words:

$$y_i \times (\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \xi_i$$

- If $\xi_i = 0$, there is no problem with $\boldsymbol{x_i}$.
- If $0 < \xi_i < 1$, $\boldsymbol{x_i}$ is correctly classified but in the margin.
- If $\xi_i = 1$, $\boldsymbol{x_i}$ is on the decision boundary (random guess).
- If $\xi_i > 1$, $\boldsymbol{x_i}$ is misclassified.

9.4.1 Soft Error

- Number of Misclassifications = $\#\{\xi_i > 1\}$
- Number of Nonseperable Points = $\#\{\xi_i > 0\}$

Soft Errors:

$$\sum_{i=1}^{N} \xi_i$$

Learning with soft errors:

$$\min_{w} \frac{||\boldsymbol{w}||_2^2}{2} + C\left(\sum_{i=1}^{N} \xi_i\right)$$
$$\text{s.t. } y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \xi_i, \; i = 1, 2, \ldots N, \xi_i \geq 0$$

where $C \geq 0$ is a parameter to tradeoff the impact of margin maximization and tolerable errors and a nonnegative ξ_i provides an estimate of the error of the decision boundary on the training example $\boldsymbol{x_i}$.

9.5 Non-linear SVM

To generalize linear decision boundary to become non-linear, $\boldsymbol{x_i}$ has to be transformed to a higher dimensional space using a function $\varphi(\boldsymbol{x_i})$. The original input space is mapped to a higher dimensional feature space where the training set is separable.

$$\varphi : \boldsymbol{x} \rightarrow \varphi(\boldsymbol{x})$$

The assumption is that in a higher dimensional space, it is easier to find a linear hyperplane to classify the data.

Optimization problem of non-linear SVM:

$$\min_{w} \frac{||\boldsymbol{w}||_2^2}{2}$$
$$\text{s.t. } y_i \times (\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}) + b) \geq 1, \, i = 1, 2, \ldots N$$

where $\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}) + b = 0$ is the hyperplane in feature space.

However, computation in the feature space can be costly because it is high dimensional since the feature space is typically very high dimensional.

9.5.1 Kernel Trick

Suppose $\varphi(\cdot)$ is given as follows, mapping an instance from two-dimensional space to six-dimensional space:

$$\varphi([X_1, X_2]) = [1, \sqrt{2}X_1, \sqrt{2}X_2, X_1^2, X_2^2, \sqrt{2}X_1X_2]$$

Given two data instances, $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$\begin{aligned}\varphi(\boldsymbol{a}) \cdot \varphi(\boldsymbol{b}) &= 1 + 2A_1B_1 + 2A_2B_2 + A_1^2B_1^2 + A_2^2B_2^2 + 2A_1A_2B_1B_2 \\ &= (1 + A_1B_1 + A_2B_2)^2\end{aligned}$$

So, if we define the kernel function as follows, there is no need to carry out $\varphi(\cdot)$ explicitly:

$$\begin{aligned}k(\boldsymbol{a}, \boldsymbol{b}) &= (1 + A_1B_1 + A_2B_2)^2 \\ &= (1 + \boldsymbol{a} \cdot \boldsymbol{b})^2\end{aligned}$$

The use of the kernel function to avoid carrying out $\varphi(\cdot)$ explicitly is known as the **kernel trick**.

Thus, if $\varphi(\cdot)$ satisfies some conditions, then we can find a function $k(\cdot, \cdot)$ such that:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$$

9.5.1.1 Well-Known Kernel Functions

Linear Kernel:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

Radial Basis Kernel Function w/ Width σ:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp\left(- \frac{||\boldsymbol{x_i} - \boldsymbol{x_j}||_2^2}{2\sigma^2} \right)$$

Polynomial Kernel Function w/ Degree d:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j} + 1)^d$$

9.5.2 Lagrange Multiplier Method

In its primal form, it is not possible to solve the optimization problem for non-linear SVMs using the kernel trick as instances in the feature space do not appear in the form of inner products. Thus, its dual form needs to be used.

Using the Lagrange Multiplier Method, we can transform the original optimization problem to its dual form where the mapping function $\varphi(\boldsymbol{x})$ does not need to computed explicitly.

Given an objective function to be minimized, with a set of inequality constraints:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w})$$
$$\text{s.t. } h_i(\boldsymbol{w}) \leq 0, \, i = 1, 2, \ldots q$$

The Langrangian for the optimization problem will be:

$$L(\boldsymbol{w}, \boldsymbol{\lambda}) = f(\boldsymbol{w}) + \sum_{i=1}^{q} \lambda_i h_i(\boldsymbol{w})$$

where $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots \lambda_q\}$ are the Langrange multipliers.

Thus,

$$\min_{w} \frac{||\boldsymbol{w}||_2^2}{2}$$
$$\text{s.t. } y_i \times (\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}) + b) \geq 1, \, i = 1, 2, \ldots N$$

can be transformed to its dual form:

$$\max_{\boldsymbol{\lambda}} L_D(\boldsymbol{\lambda}) = -\left(\frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \big(\varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})\big) - \sum_{i=1}^{N} \lambda_i \right)$$

9.5.2.1 Dual Optimization Problem

- The dual Langrangian involves only the Langrange multipliers and the training data.
- The negative sign in the dual Langrangian transforms a minimization problem of the primal form to a maximization problem of the dual form.
- The objective is to maximize $L_D(\boldsymbol{\lambda})$.
  - Can be solved using numerical techniques such as quadratic programming.

Once the λ_i values are found, we can obtain the feasible solutions for $\boldsymbol{w}$ and b from the two equations below:

$$\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \varphi(\boldsymbol{x_i})$$

$$\lambda_i(y_i(\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}) + b) - 1) = 0$$

The decision boundary can be expressed as:

$$\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}) + b = 0$$
$$\left(\sum_{i=1}^{N} \lambda_i y_i \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x}) \right) + b = 0$$

A test instance $\boldsymbol{x^*}$ can be classified using:

$$f(\boldsymbol{x^*}) = \operatorname{sign}\left(\sum_{i=1}^{N} \lambda_i y_i \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x^*}) + b \right)$$

If $\boldsymbol{x_i}$ is a support vector, then the corresponding $\lambda_i > 0$, otherwise, $\lambda_i = 0$.

9.5.3 Non-linear SVM via Kernel Trick

Training:

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \big(\varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})\big) \right)$$

Decision Boundary:

$$\sum_{i=1}^{N} \lambda_i y_i \big(\varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x^*})\big) + b$$

Thus, the data points only appear as inner product in feature space and can be replaced by the kernel function.

Training:

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) \right)$$

Decision Boundary:

$$\sum_{i=1}^{N} \lambda_i y_i k(\boldsymbol{x_i}, \boldsymbol{x^*}) + b$$

If $\boldsymbol{x_i}$ is a support vector, then the corresponding $\lambda_i > 0$, otherwise, $\lambda_i = 0$.

Chapter 10

Additional Notes for Multi-Class Classification Problems

Given a 3-class classification problem — C_1, C_2 & C_3, the following binary classifications are performed:

1. Binary Classification 1: Positive (C_1) v.s. Negative (C_2 & C_3)
2. Binary Classification 2: Positive (C_2) v.s. Negative (C_1 & C_3)
3. Binary Classification 3: Positive (C_3) v.s. Negative (C_1 & C_2)

The results of each binary classification need to be combined for a test instance $\boldsymbol{x^*}$ to make a final prediction.

- Case 1: Suppose f_i is a probabilistic model that outputs how likely an instance belongs to class C_i.
  - The final prediction will be based on which f_i had the highest probability output.
- Case 2: Suppose f_i only generates 0/1.
  - A voting table needs to be generated from the results of each f_i and the class with the most votes is chosen.
  - Each f_i will either generate a vote for C_i or a vote for every class except C_i.

Chapter 11

k-Nearest Neighbor Classifier

NN Classifier uses k *closest* points (nearest neighbors) for performing classification. It requires three things:

1. Set of stored labeled instances.
2. Distance metric to compute the distance between instances.
3. The value of k, the number of nearest neighbors to retrieve.

To classify an unknown instance:

1. Compute the distance to other training instances.
2. Identify the k nearest neighbors.
3. Use class labels of nearest neighbors to determine the class label of unknown instances.

11.1 Distance Metric

One way to compute the distance between two points is by computing the Euclidean distance:

$$d(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}$$

To determine the class from the nearest neighbor list, a majority vote of class labels among the k-nearest neighbors can be used.

$$y^* = \underset{k}{\operatorname{argmax}} \sum_{(\boldsymbol{x_i}, y_i) \in N_{\boldsymbol{x^*}}} I(y = y_i)$$

11.2 Value of k

- If k is too small, sensitive to noise points.
- If k is too large, neighborhood may include points from other classes.

11.3 Voting Approaches

In simple majority voting, every neighbor has the same impact on the classification which makes the algorithm sensitive to the choice of k. The solution is to implement **distance-weight voting**:

$$y^* = \underset{k}{\operatorname{argmax}} \sum_{(\boldsymbol{x_i}, y_i) \in N_{\boldsymbol{x^*}}} w_i \times I(y = y_i)$$

where w_i is the weight according to the distance of the nearest neighbor:

$$w_i = \frac{1}{d(\boldsymbol{x^*}, \boldsymbol{x_1})^2}$$

11.4 Normalization

Another issue is that the feature may need to be scaled to prevent the distance from being dominated by some features. For example, the height of a person may vary from 1.5m to 1.8m but the income may vary from \$10K to \$10M. The solution is to normalize the features on different scales.

11.4.1 Min-Max Normalization

$$v_{new} = \frac{v_{old} - min_{old}}{max_{old} - min_{old}} (max_{new} - min_{new}) + min_{new}$$

where *old* represents the original scale and *new* represents the new scale to be normalized to.

11.4.2 Standardization

$$v_{new} = \frac{v_{old} - \mu_{old}}{\sigma_{old}}$$

The above results in $\mu_{new} = 0$ and $\sigma_{new} = 1$.

11.5 Notes

- k-NN Classifiers are lazy learners as it does not build models explicitly.

- *Training* is very efficient but classifying unknown test instances is relatively expensive.

Chapter 12

Model Evaluation

The most widely-used evaluation metric is:

$$\begin{aligned}\text{Accuracy} &= \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \\ &= \frac{\text{TP} \ + \ \text{TN}}{\text{TP} \ + \ \text{FN} \ + \ \text{FP} \ + \ \text{TN}}\end{aligned}$$

$$\begin{aligned}\text{Error Rate} &= \frac{\text{Number of Wrong Predictions}}{\text{Total Number of Predictions}} \\ &= \frac{\text{FP} \ + \ \text{FN}}{\text{TP} \ + \ \text{FN} \ + \ \text{FP} \ + \ \text{TN}}\end{aligned}$$

However, this metric has limitations. For example, if the training set has 99 examples of Class 0 and 1 example of Class 1, even if the model predicts everything as Class 0, it has an accuracy of 99%, which is misleading.

In many real-world applications, datasets may have imbalanced class distributions. Since accuracy measure treats every class as equally important, it may not be suitable for analyzing imbalanced datasets, where the rare class is considered more interesting than the majority class.

For binary classification, the rare class is often denoted as the positive class while the majority class is denoted as the negative class.

Precision and **Recall** are two widely used metrics in applications where successful detection of one of the classes is considered more significant than detection of the other classes. A good model should have both high precision and recall.

12.1 Precision

Precision is to measure among all the predicted positive instances, how many are true positive.

$$p = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

12.2 Recall

Recall is to measure among all the true positive instances, how many are predicted correctly by the classifier.

$$r = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

12.3 F_1-measure

Recall and Precision can be summarized into another metric known as F_1-measure:

$$F_1 = \frac{2rp}{r + p}$$

A high value of F_1-measure ensures that precision and recall are reasonably high.

12.4 Receiver Operating Characteristic (ROC)

ROC is a graphical approach to displaying the tradeoff between true positive rate and false positive rate of a classifier.

$$\text{True Positive Rate: TPR } = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{False Positive Rate: FPR } = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

The ROC curve graphs (FPR, TPR).

- A value of (0, 0) declares everything to be negative class.
- A value of (1, 1) declares everything to be positive class.
- A value of (0, 1) is ideal.
- A value on the diagonal is random guessing.
- A value below the diagonal means the prediction is the opposite of the true class.

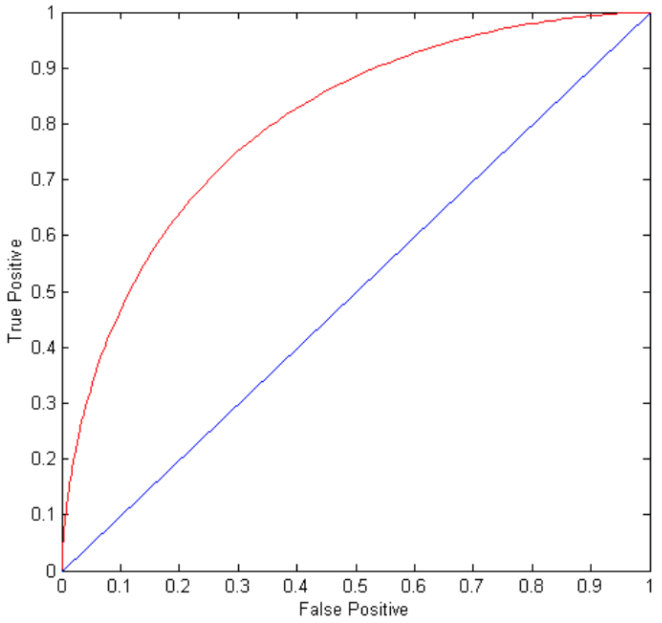


Figure 12.1: ROC Curve

To compare two classifiers using the ROC, the area under the ROC curve must be compared as no classifier consistently outperforms the other. The area under the ROC curve should ideally be 1.

Chapter 13

Regression

13.1 Linear Regression Model

In a special case where an instance is represented by one input feature, the goal is to learn a linear function $f(x)$ (where x is a scalar) in terms of w (also a scalar) from $\{x_i, y_i\}$ such that the difference (i.e. error) between the predicted values of $f(x_i)^{'}$s and the ground truth values $y_i^{'}$s is as small as possible.

$$f(x) = x \times w$$

Suppose sum-of-squares error is used:

$$\begin{aligned} E(w) &= \frac{1}{2}\sum_{i=1}^{N}(f(x_i) - y_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{N}(w \times x_i - y_i)^2 \end{aligned}$$

The linear model is learnt in terms of w by minimizing the error:

$$w^* = \underset{w}{\operatorname{argmin}}\, E(w)$$

13.1.1 Linear Regression Model Learning

To solve the unconstrained minimization problem, the derivate of $E(w)$ with respect to w can be set to zero.

$$\begin{aligned} \frac{\partial E(w)}{\partial w} &= 0 \\ \frac{\partial\left(\frac{1}{2}\sum_{i=1}^{N}(w \times x_i - y_i)^2\right)}{\partial w} &= 0 \\ \sum_{i=1}^{N}(w \times x_i - y_i) \times x_i &= 0 \\ w\sum_{i=1}^{N}x_i^2 \sum_{i=1}^{N} y_i \times x_i &= 0 \end{aligned}$$

$$w = \frac{\sum_{i=1}^{N} y_i \times x_i}{\sum_{i=1}^{N} x_i^2}$$

13.2 Math Review

13.2.1 Matrix Transformation

If a matrix/vector $\boldsymbol{X}$ is transposed to $\boldsymbol{X}^T$, each element in the matrix/vector switches from e_{ij} to e_{ji} where i is the original row number and j is the original column number of the element.

- If $\boldsymbol{X}$ is a **square matrix**, its number of rows and columns is the same.
- If $\boldsymbol{X}$ is a **symmetric matrix**, it is a square matrix and $\boldsymbol{X}^T = \boldsymbol{X}$.

$$\begin{aligned} \boldsymbol{X}\boldsymbol{Y}^T &= \boldsymbol{Y}^T\boldsymbol{X}^T \\ \boldsymbol{X}\boldsymbol{w}^T &= \boldsymbol{w}^T\boldsymbol{X}^T \\ \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{w}^T &= \boldsymbol{w}^T\boldsymbol{x} \\ x^T &= x \end{aligned}$$

13.2.2 Identity Matrix

- For a square matrix, if $\boldsymbol{X}$ is invertible, then $\boldsymbol{X}\boldsymbol{X}^{-1} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix.
- Any vector (or matrix) $\boldsymbol{x}$ (or $\boldsymbol{X}$) times the identity matrix $\boldsymbol{I}$ equals to the vector (or matrix) itself.

13.3 Linear Regression Model (cont.)

To learn a linear function $f(\boldsymbol{x})$ in a more general form in terms of $\boldsymbol{w}$ and b:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

By defining $w_0 = b$ and $X_0 = 1$, $\boldsymbol{w}$ and $\boldsymbol{x}$ are of $d+1$ dimensions:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

Suppose sum-of-squares error is used:

$$\begin{aligned} E(\boldsymbol{w}) &= \frac{1}{2}\sum_{i=1}^{N}(f(\boldsymbol{x_i}) - y_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x_i} - y_i)^2 \end{aligned}$$

Learn the linear model using in terms of $\boldsymbol{w}$ by minimizing the error:

$$\boldsymbol{w^*} = \underset{\boldsymbol{w}}{\operatorname{argmin}} E(\boldsymbol{w}) + \frac{\lambda}{2}||\boldsymbol{w}||_2^2$$

where λ is a positive tradeoff parameter and $||\boldsymbol{w}||_2^2$ is a regularization term to control the complexity of the model.

13.3.1 Linear Regression Model Learning

To solve the unconstrained minimization problem, the derivate of $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ can be set to zero.

$$\begin{aligned} \frac{\partial\left(E(\boldsymbol{w}) + \frac{\lambda}{2}||\boldsymbol{w}||_2^2\right)}{\partial \boldsymbol{w}} &= 0 \\ \frac{\partial\left(\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x_i} - y_i)^2 + \frac{\lambda}{2}\boldsymbol{w} \cdot \boldsymbol{w}\right)}{\partial \boldsymbol{w}} &= 0 \end{aligned}$$

A closed-form solution for the above can be obtained.

Denoting $\boldsymbol{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots \boldsymbol{x_N})^T$:

$$\boldsymbol{X} = \begin{pmatrix} x_{10} & \dots & x_{N0} \\ \vdots & \ddots & \vdots \\ x_{1d} & \dots & x_{Nd} \end{pmatrix}^T = \begin{pmatrix} x_{10} & \dots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{N0} & \dots & x_{Nd} \end{pmatrix}$$

And $\boldsymbol{y}$:

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

Solving the minimization problem:

$$\begin{aligned} \frac{\partial\left(\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x_i} - y_i)^2 + \frac{\lambda}{2}\boldsymbol{w} \cdot \boldsymbol{w}\right)}{\partial \boldsymbol{w}} &= 0 \\ \sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x_i} - y_i)\boldsymbol{x_i} + \lambda \boldsymbol{w} &= 0 \\ \sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x_i})\boldsymbol{x_i} - \sum_{i=1}^{N} y_i \boldsymbol{x_i} + \lambda \boldsymbol{w} &= 0 \\ (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y} + \lambda \boldsymbol{I}\boldsymbol{w} &= 0 \\ (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{y} \end{aligned}$$

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

As long as λ is positive, $(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})$ is always invertible.

13.4 Linear Basis Function Models

Linear basis function models can be used for non-linear fitting. The linear function $f(\boldsymbol{x})$ in terms of a set of basis functions is written as:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})$$

where $\boldsymbol{\phi}(\boldsymbol{x}) = (\boldsymbol{\phi}_0(\boldsymbol{x}), \boldsymbol{\phi}_1(\boldsymbol{x}), \dots \boldsymbol{\phi}_m(\boldsymbol{x}))$ and each $\boldsymbol{\phi}_i(\boldsymbol{x})$ maps the instance $\boldsymbol{x}$ to a scalar.

Typically, $\boldsymbol{\phi}_0(\boldsymbol{x}) = 1$, then w_0 acts as a bias.

In the simplest case (if $d = m$ and $\boldsymbol{\phi}_i(\boldsymbol{x}) = x_i$), it is reduced to a standard linear model.

13.4.1 Examples of Basis Functions

Polynomial Basis Functions:

$$\phi_j(\boldsymbol{x}) = ||\boldsymbol{x}||_2^j$$

In a special case x with one input feature:

$$\phi_j(x) = x^j$$

Then, the polynomial curve will be:

$$f(\boldsymbol{x}) = \sum_{i=0}^{m} w_i x^i$$

Gaussian Basis Functions:

$$\phi_j(\boldsymbol{x}) = \exp\Big(-\frac{||\boldsymbol{x}-\boldsymbol{u}_j||_2^2}{2\sigma^2}\Big)$$

Sigmoid Basis Functions:

$$\phi_j(\boldsymbol{x}) = g\Big(\frac{||\boldsymbol{x}-\boldsymbol{u}_j||^2}{\sigma}\Big)$$

where:

$$g(a) = \frac{1}{1-\exp(-a)}$$

where $\boldsymbol{u}_j$ and σ control location and scale (width).

13.4.2 Linear Basis Function Learning

The goal is to minimize the following error:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{w}\cdot\phi(\boldsymbol{x_i}) - y_i)^2 + \frac{\lambda}{2}||\boldsymbol{w}||_2^2$$

A closed-form solution for $\boldsymbol{w}$ can be obtained:

Denoting $\boldsymbol{\Phi} = (\boldsymbol{\phi}(\boldsymbol{x_1}), \boldsymbol{\phi}(\boldsymbol{x_1}), \ldots \boldsymbol{\phi}(\boldsymbol{x_N}))^T$:

$$\begin{aligned}\boldsymbol{\Phi} &= \begin{pmatrix}\phi_0(\boldsymbol{x_1}) & \ldots & \phi_0(\boldsymbol{x_N})\\ \vdots & \ddots & \vdots\\ \phi_m(\boldsymbol{x_1}) & \ldots & \phi_m(\boldsymbol{x_N})\end{pmatrix}^T\\ &= \begin{pmatrix}\phi_0(\boldsymbol{x_1}) & \ldots & \phi_m(\boldsymbol{x_1})\\ \vdots & \ddots & \vdots\\ \phi_0(\boldsymbol{x_N}) & \ldots & \phi_m(\boldsymbol{x_N})\end{pmatrix}\end{aligned}$$

The closed-form solution for $\boldsymbol{w}$ can be written as:

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$$

where:

$$\boldsymbol{y} = \begin{pmatrix}y_1\\ \vdots\\ y_N\end{pmatrix}$$

13.4.3 Limitations

- In practice, to learn a precise regression model, we may need a large number of basis functions.
- The computational cost is expensive.

13.4.4 Kernelized Regression

By using the kernel trick $k(x_i, x_j) = \phi(x_i)\cdot\phi(x_j)$:

$$\begin{aligned}f(\boldsymbol{x}) &= \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x})\\ f(\boldsymbol{x}) &= \boldsymbol{k}(\boldsymbol{x})(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}\end{aligned}$$

where:

$$\begin{aligned}\boldsymbol{K} &= \begin{pmatrix}k(\boldsymbol{x_1},\boldsymbol{x_1}) & \ldots & k(\boldsymbol{x_1},\boldsymbol{x_N})\\ \vdots & \ddots & \vdots\\ k(\boldsymbol{x_N},\boldsymbol{x_1}) & \ldots & k(\boldsymbol{x_N},\boldsymbol{x_N})\end{pmatrix}\\ &= \begin{pmatrix}\phi(\boldsymbol{x_1})\cdot\phi(\boldsymbol{x_1}) & \ldots & \phi(\boldsymbol{x_1})\cdot\phi(\boldsymbol{x_N})\\ \vdots & \ddots & \vdots\\ \phi(\boldsymbol{x_N})\cdot\phi(\boldsymbol{x_1}) & \ldots & \phi(\boldsymbol{x_N})\cdot\phi(\boldsymbol{x_N})\end{pmatrix}\end{aligned}$$

13.5 Evaluation

Root Mean Square Error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x_i}) - y_i)^2}$$

Mean Absolute Error (MAE):

$$\frac{1}{N}\sum_{i=1}^{N}|f(\boldsymbol{x_i}) - y_i|$$

Chapter 14

Ensemble Learning

What?

Ensemble learning refers to a collection of methods that learn a target function by training a number of individual learners and combining their predictions.

Why?

- Accuracy: a more reliable mapping can be obtained by combining the output of multiple *experts* (i.e. classifiers).
- Efficiency: a complex problem can be decomposed into multiple sub-problems that are easier to understand and solve (divide-and-conquer approach).
- Multiple representations and multiple models: There is not a single model that works for all pattern recognition problems!

When?

When you can build component classifiers that are more accurate than chance and, more importantly, that are independent from each other.

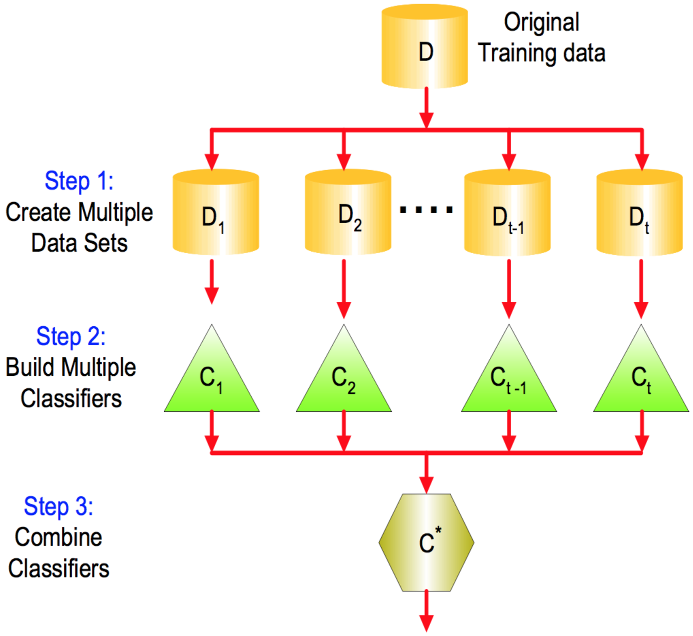


Figure 14.1: Ensemble Learning

14.1 Why do ensembles work?

14.1.1 Increased Accuracy

If the error of individual classifiers is independent, we can eliminate the error of combined classifier. For example:

Assume a binary classification problem for which you can train individual classifiers with an error rate of 0.3. Assume that you build an ensemble by combining the prediction of 21 such classifiers with a majority vote. If there are i classifiers out of 21 making an error in prediction, the error of classification is:

$$\binom{21}{i} 0.3^i (1-0.3)^{21-i}$$

where:

$$\binom{21}{i} = \frac{21!}{(21-i)!i!}$$

In order for the ensemble to misclassify an example, 11 or more classifiers have to be in error, or a probability of 0.026.

$$\sum_{i=11}^{2} 1 \binom{21}{i} 0.3^i (1-0.3)^{21-i} = 0.026$$

14.1.2 Approximation by Ensemble Averaging

The desired target function may not be implementable with individual classifiers, but may be approximated by ensemble averaging. For example:

Assume that we use a single discriminant function as the classifier, whose decision boundary is linear. However, the true boundary may be non-linear. If we combine many linear discrimination functions, we can approximate the non-linear boundary.

14.2 Methods for Constructing Ensembles

1. Subsampling the training examples
   a. Multiple hypotheses are generated by training individual classifiers on different datasets obtained by resampling a common training set (Bagging, Boosting).
2. Manipulating the input features
   a. Multiple hypotheses are generated by training individual classifiers on different representations or different subsets of a common feature vector.
3. Manipulating the output targets
   a. The output targets for C classes are encoded with an L-bit codeword and an individual classifier is built to predict each one of the bits in the codeword.
   b. Additional "auxiliary" targets may be used to differentiate classifiers.
4. Modifying the learning parameters of the classifier
   a. A number of classifiers are built with different learning parameters, such as number of neighbors in a *k*-Nearest Neighbor rule, initial weights in a Multi-Layer Perceptron etc.

14.3 Structure of Ensemble Classifiers

14.3.1 Parallel

All the individual classifiers are invoked independently and their results are fused with a combination rule (e.g. average, weighted voting) or a meta-classifier (e.g. stacked generalization).

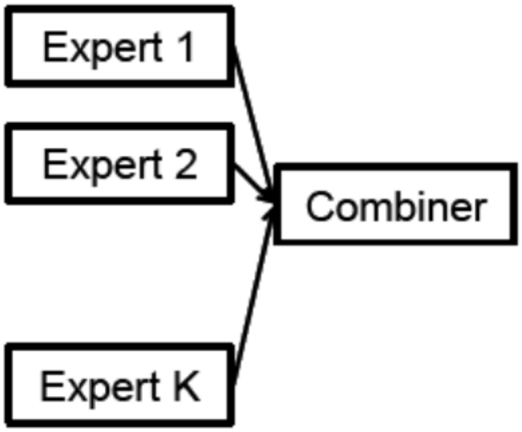


Figure 14.2: Parallel Ensemble Learning

14.3.2 Cascading or Hierarchical

Classifiers are invoked in a sequential or tree-structured fashion. For the purpose of efficiency, inaccurate but fast methods are invoked first (maybe using a small subset of the features) and computationally more intensive but accurate methods are left for the latter stages.

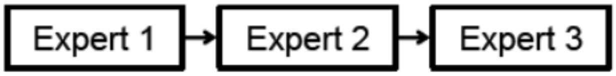


Figure 14.3: Cascading Ensemble Learning

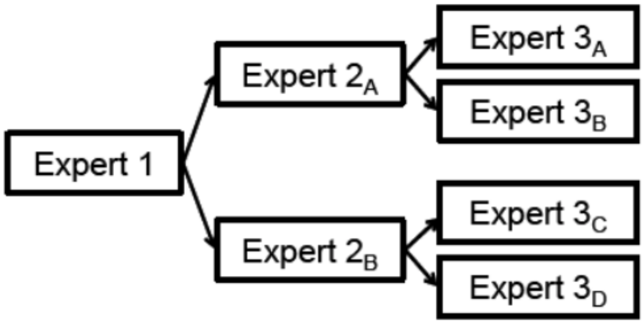


Figure 14.4: Hierarchical Ensemble Learning

14.4 Combination Strategies

14.4.1 Static Combiners

The combiner decision rule is independent of the feature vector. Static approaches can be broadly divided into non-trainable and trainable.

- Non-trainable: The voting is performed independent of the performance of each individual classifier.
  - Voting: used when each classifier produces a single class label. In this case, each classifier "votes" for a particular class and the class with the majority vote on the ensemble wins.
  - Averaging: used when each classifier produces a confidence estimate (e.g. a posterior). In this case, the winner is the class with the highest average posterior across the ensemble.
- Trainable: The combiner undergoes a separate training phase to improve the performance of the ensemble machine. Two noteworthy approaches are:
  - Weighted Averaging: the output of each classifier is weighted by a measure of its own performance, for example, prediction accuracy on a separate validation set.
  - Stacked Generalization: the output of the ensemble serves as a feature vector to a meta-classifier.

14.4.1.1 Stacked Generalization

In stacked generalization, the output pattern of an ensemble of trained experts serves as an input to a second-level expert.

Training of this modular ensemble can be performed as follows:

1. From a dataset X with N examples, leave out one test example and train each of the Level-0 experts on the remaining $(N-1)$ examples.
2. Generate a prediction for the test example. The output pattern $y = [y_1, y_2, \ldots y_K]$ across the Level-0 experts, along with the target t for the test example, becomes a training example for the Level-1 expert.
3. Repeat this process in a leave-one-out fashion. This yields a training set Y with N examples, which is used to train the Level-1 expert separately.
4. To make full use of the training data, re-train all the Level-0 experts one more time using all N examples in X.

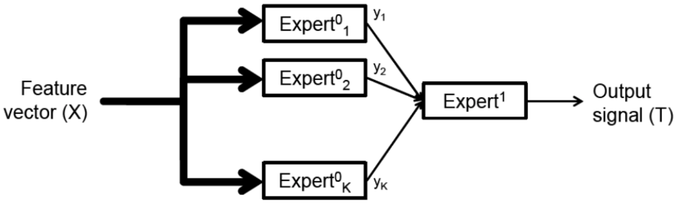


Figure 14.5: Stacked Generalization

14.4.2 Adaptive Combiners

The combiner is a function that depends on the input feature vector. Thus, the ensemble implements a function that is local to each region in the feature space.

This divide-and-conquer approach leads to modular ensembles where relatively simple classifiers specialize in different parts of the input-output space. In contrast with static-combiner ensembles, the individual experts here do not need to perform well for all inputs, only in their region of expertise.

Representative examples of this approach are Mixture of Experts (ME) and Hierarchical ME.

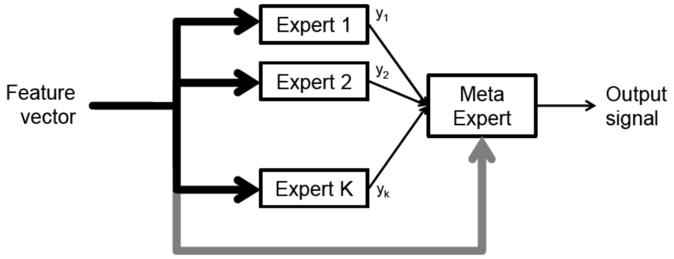


Figure 14.6: Adaptive Learning

14.4.2.1 Mixture of Experts

- A gating network is used to partition the feature space into different regions, with one expert in the ensemble being responsible for generating the correct output within one region.
- The experts in the ensemble and the gating network are trained simultaneously, which can be efficiently performed with the expectation-maximization algorithm.
- ME can be extended to a multi-level hierarchical structure, where each component is itself a ME. In this case, a linear network can be used for the terminal classifiers without compromising the modeling capabilities of the machine.

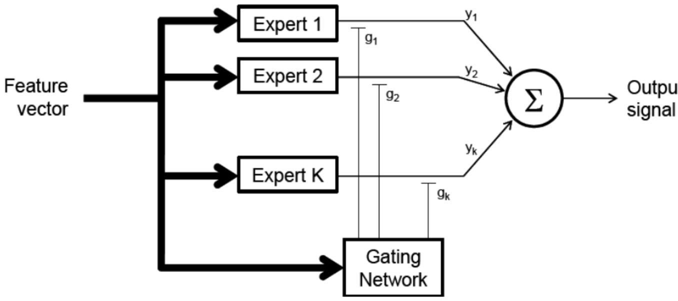


Figure 14.7: Mixture of Experts

14.5 Bagging

This method is also known as "arcing" (adaptive re-weigthing and combining). One of the simplest is "bagging" (bootstrap aggregation).

Algorithm:

1. Given N labeled data points $\{x_i\}$, use random sampling (with replacement) to create K new data sets, each with $N' < N$.
2. Learn a classifier (of whatever type) based on each of the K data sets giving K classifiers.
3. Classify a new point x based on a majority vote among the K classifiers.

- The perturbation in the training set due to the bootstrap resampling causes different hypotheses to be built, particularly if the classifier is unstable.
  - A classifier is said to be unstable if a small change in the training data (e.g. order of presentation of examples) can lead to a radically different hypothesis. This is the case of decision trees and, arguably, neural networks.
- Bagging can be expected to improve accuracy if the induced classifiers are uncorrelated.
  - In some cases, such as k-Nearest Neighbors, bagging has been shown to degrade performance as compared to individual classifiers as a result of an effectively smaller training set.

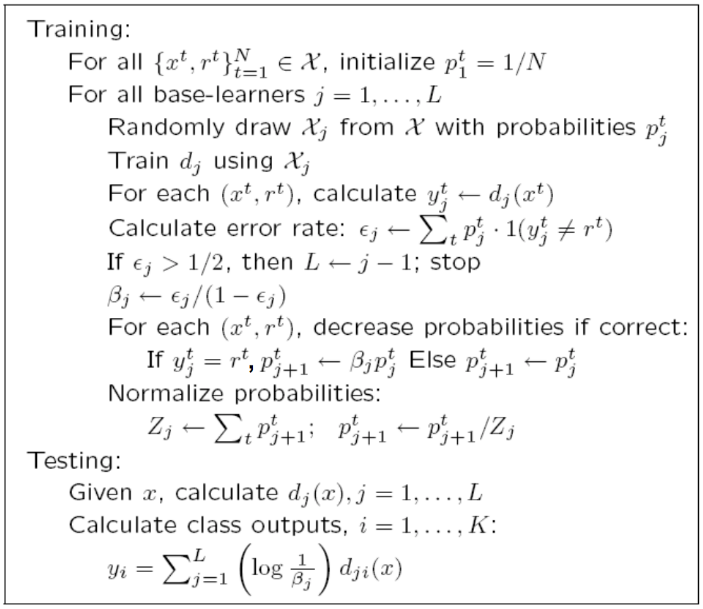
14.6 Boosting

Boosting takes a different resampling approach than bagging, which maintains a constant probability of $1/N$ for selecting each individual example. In boosting, this probability is adapted over time based on performance. The component classifiers are built sequentially and examples that are mislabeled by previous components are chosen more often than those that are correctly classified.

Boosting is based on the concept of a “weak learner” (i.e. an algorithm that performs slightly better than chance). An example is a binary classifier with a 50% classification rate. A weak learner can be converted into a strong learner by changing the distribution of the training examples. Boosting can also be used with classifiers that are highly accurate but the benefits in this case will be very small.

A popular variant of boosting is AdaBoost (Adaptive Boosting), which allows the designer to continue adding components until an arbitrarily small error rate is obtained on the training set.

14.6.1 AdaBoost



```
Training:
    For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$
    For all base-learners $j = 1, \ldots, L$
        Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities $p_j^t$
        Train $d_j$ using $\mathcal{X}_j$
        For each $(x^t, r^t)$, calculate $y_j^t \leftarrow d_j(x^t)$
        Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
        If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop
        $\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$
        For each $(x^t, r^t)$, decrease probabilities if correct:
            If $y_j^t = r^t$, $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
        Normalize probabilities:
            $Z_j \leftarrow \sum_t p_{j+1}^t$;   $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$
Testing:
    Given $x$, calculate $d_j(x), j = 1, \ldots, L$
    Calculate class outputs, $i = 1, \ldots, K$:
        $y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$
```

Figure 14.8: AdaBoost Algorithm

Chapter 15

Cluster Analysis

15.1 What is Cluster Analysis?

Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in the other groups. Intra-cluster distances are minimized whereas inter-cluster distances are maximized.

15.1.1 Measurements & Distances

Any technique for clustering needs a measure of distance between items. The distance d_i between two objects x_i and x_j can be calculated using the Euclidean distance between two points of p dimensions.

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$$

The Euclidean distance is an example of a dissimilarity measure. However, it may be more convenient to work with a similarity measure, such as the square of the correlation coefficient:

$$r_{ij}^2 = \frac{\sum\limits_{m=1}^{p} (x_{im} - \bar{x}_m)(x_{jm} - \bar{x}_m)}{\sqrt{\sum\limits_{m=1}^{p} (x_{im} - \bar{x}_m)^2 \sum\limits_{m=1}^{p} (x_{jm} - \bar{x}_m)}}$$

where $\bar{x}_m$ is the mean of the m dimension between the two points.

15.1.2 Application of Clustering in Image Segmentation

- There is a need for data reduction i.e. obtaining a compact representation for *interesting* image data in terms of a set of components.
- The aim is to find components that belong together (form clusters).

15.1.3 Types of Clusterings

A **clustering** is a set of clusters. There are two main types of clusterings:

1. Partitional Clustering - non-overlapping subsets, each data object is in exactly one subset.
2. Hierarchical Clustering - nested clusters, organized as a hierarchical tree.

15.2 K-means Clustering

K-means clustering is a partitional clustering approach, where each cluster is associated with a centroid (i.e. center point). Each point is assigned to the cluster with the closest centroid and the number of clusters (i.e. K) must be specified.

Algorithm:

1. Select K points as the initial centroids.
2. Repeat:
    a. Form K clusters by assigning all points to the closest centroid.
    b. Recompute the centroid of each cluster.
3. Stop when the centroids no longer change.

15.2.1 Further Details

- The initial centroids are often chosen randomly and the clusters produced vary from one run to another.
- The centroid is (usually) the mean of the points in the cluster.
- 'Closeness' is measured by Euclidean distance, cosine similarity, correlation etc.
- K-means will converge for common similarity measures and most of the convergence happens in the first few iterations. Often, the stopping condition may be changed to "Stop when relatively few points change clusters".
- Complexity is $O(n \times K \times I \times d)$, where n = number of points, K = number of clusters, I = number of iterations, d = number of attributes.

15.2.2 Problems with Selecting Initial Points

If there are K ‘real’ clusters, then the chance of selecting one centroid from each cluster is small. The chance is relatively small when K is large and it is not guaranteed that the centroids will readjust themselves in the right way. To solve this, the following methods can be used:

1. Multiple runs.
2. Use hierarchical clustering to determine initial centroids.
3. Select more than K initial centroids (that are widely separated) and then select among these initial centroids.

15.2.3 Evaluating K-means Clusters

The most common measure is Sum of Squared Error (SSE), where the error for each point is the distance to the nearest cluster.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} d(m_i, x)^2$$

where x is a data point in cluster C_i and m_i is the representative point for cluster C_i.

15.2.4 Limitations of K-means

K-means has problems when:

1. Clusters are of different sizes or densities.
2. Data contains outliers.
3. The number of clusters (i.e. K) is difficult to determine.

15.3 Hierarchical Clustering

Hierarchical clustering produces a set of nested clusters organized as a hierarchical tree and can be visualized as a dendogram.

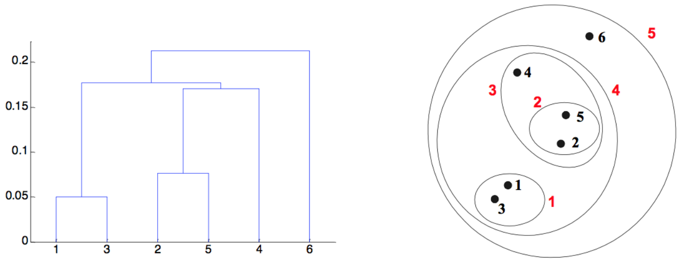


Figure 15.1: Hierarchical Clustering

15.3.1 Strengths of Hierarchical Clustering

- There is no assumption about the number of clusters and any desired number of clusters can be obtained by *cutting* the dendogram at the appropriate level.
- The clustering may correspond to meaningful taxonomies (e.g. animal kingdom).

15.3.2 Types of Hierarchical Clustering

1. Agglomerative: Start with the points as individual clusters and at each step, the closest pair of clusters is merged until one cluster (or k clusters) are left.
2. Divisive: Start with one, all-inclusive cluster and at each step, split a cluster until each cluster contains one point (or there are k clusters).

15.3.3 Agglomerative Hierarchical Clustering

Algorithm:

1. Compute the proximity matrix.
2. Let each data point be a cluster.
3. Repeat:
   a. Merge the two closest clusters.
   b. Update the proximity matrix.
4. Stop when only one cluster remains.

The key operation is the computation of proximity between two clusters. The different approaches to calculate this proximity distinguish the different algorithms.

15.3.3.1 Inter-Cluster Similarity

1. MIN (Single Linkage): Similarity of two clusters is based on the two most similar (i.e. closest) points in the different clusters. Hence, this inter-cluster similarity is only determined by one pair of points i.e. by one link in the proximity graph.

$$d(A, B) = \min_{a \in A, b \in B} d(a, b)$$

2. MAX (Complete Linkage): Similarity of two clusters is based on the two least similar (i.e. farthest) points in the different clusters. Hence, this inter-cluster similarity is determined by all pairs of points in the two clusters and a cluster is only formed when all points in the cluster are completely linked.

$$d(A, B) = \max_{a \in A, b \in B} d(a, b)$$

3. Group Average (Average Linkage): Proximity of two clusters is the average of pairwise proximity between points in the two clusters. There is a need to use average connectivity for scalability since total proximity favors large clusters. This method is less susceptible to noise and outliers but is biased towards spherical clusters.

$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A, b \in B} d(a,b)$$

4. Distance Between Centroids:

$$d(A,B) = d(c_A, c_B)$$

15.3.3.2 Time & Space Complexity

- $O(N^2)$ space complexity due to the proximity matrix, where N is the number of points.
- $O(N^3)$ time complexity as there are N steps and at each step, N^2 time is needed to search and update the proximity matrix. In some approaches, the time complexity can be reduced to $O(N^2 log(N))$.

15.4 Measures of Cluster Validity

Numerical measures used to judge various aspects of cluster validity are classified into three types:

1. External Index: Used to measure the extent to which cluster labels match externally supplied cluster labels (Entropy, Purity).
2. Internal Index: Used to measure the goodness of a clustering structure without respect to external information (Sum of Squared Error).
3. Relative Index: Used to compare two different clusterings or clusters. Often an external or internal index is used for this function.

15.4.1 Internal Measures

15.4.1.1 Sum of Squared Error (SSE)

Clusters with complicated distributions aren't well separated. SSE is good for comparing two clusterings or two clusters (average SSE) and can also be used to estimate the number of clusters.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} d(m_i, x)^2$$

15.4.1.2 Cohesion & Separation

Cluster Cohesion measures how closely related objects are within a cluster. It can be measured by the within cluster sum of squares:

$$WSS = \sum_i \sum_{x \in C_i} d(x, m_i)^2$$

Cluster Separation measures how distinct or well-separated a cluster is from other clusters. It can be measured by the between cluster sum of squares:

$$BSS = \sum_i |C_i|(m - m_i)^2$$

The sum of WSS and BSS will always be a constant.

15.4.1.3 Proximity Graph Approach for Cohesion & Separation

- Cluster Cohesion is the sum of the weight of all links within a cluster.
- Cluster Separation is the sum of the weights between nodes in the cluster and nodes outside the cluster.

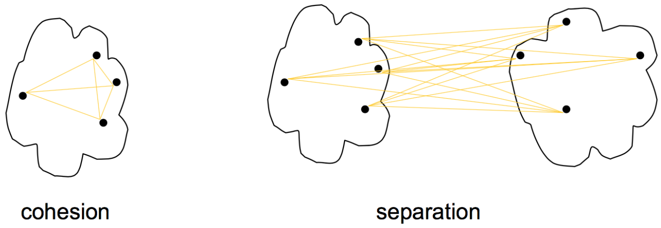


Figure 15.2: Cohesion & Separation (Proximity Graph)

15.4.1.4 Silhouette Coefficient

Silhouette Coefficient combines the ideas of both cohesion and separation, but for individual points as well as clusters and clusterings.

For an individual point i:

1. Calculate a = average distance of i to the points in its cluster.
2. Calculate b = min(average distance of i to points in another cluster).
3. Silhouette Coefficient $s = 1 - a/b$ if $a < b$ (or $s = b/a - 1$ if $a > b$).
    a. Typically between 0 and 1. The closer to 1 the better.

- Average silhouette width for a cluster can be calculated by averaging the silhouette coefficients of all the points in the cluster.

- Average silhouette width for a clustering can be calculated by averaging the average silhouette widths of all the clusters.

15.4.2 External Measures

15.4.2.1 Entropy & Purity

- Probability that a member of cluster j belongs to class i:

$$p_{ij} = \frac{m_{ij}}{m_j}$$

where m_j is the number of values in cluster j and m_{ij} is the number of values of class i in cluster j.

- Entropy of cluster j:

$$e_j = -\sum_{i=1}^{L} p_{ij} \log_2 p_{ij}$$

where L is the number of classes.

- Entropy of clustering:

$$\boldsymbol{e} = \sum_{j=1}^{K} \frac{\boldsymbol{m_j}}{\boldsymbol{m}} \boldsymbol{e_j}$$

where m_j is the size of the cluster j, K is the number of clusters and m is the total number of data points.

- Purity of cluster j:

$$\text{purity}_j = \max_i p_{ij}$$

- Purity of clustering:

$$\text{purity} = \sum_j \frac{m_j}{m} \text{purity}_j$$

15.5 Visual Object Classes Challenge

Cluster analysis can be applied in the field of object recognition from images.

1. Every image is converted into a bag of ‘words’ i.e. image blocks, where each image block represents an independent feature of the object in the image.
2. The different image blocks are clustered to generate a dictionary of ‘codewords’, where a codeword is the centroid of a cluster.
3. Now, each image can be represented using a bag of these codewords by finding the centroid of the cluster each image block belongs to.
4. Classifiers can use the information about the frequency of codewords in an image to recognize objects.

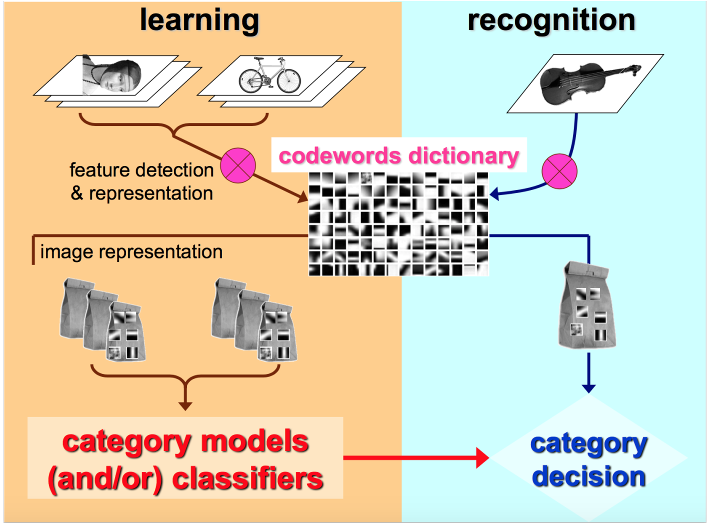


Figure 15.3: Visual Object Classes Challenge

The relationship between each image block / codeword is not needed when using this method.

Chapter 16

Dimensionality Reduction

Why?

- Reduces time complexity (less computation)
- Reduces space complexity (fewer parameters)
- Saves the cost of observing the feature.
- Simpler models are more robust on small datasets.
- More interpretable (simpler explanation).
- Data visualization is easier if plotted in 2 or 3 dimensions.

Feature Selection vs Extraction

Feature selection involves choosing $k < d$ important features, ignoring the remaining $d - k$ features. E.g. Subset selection algorithms.

Feature extraction involves projecting the original d dimensions to new $k < d$ dimensions.

16.1 Subset Selection

There are 2^d subsets of d features.

Forward Search: Add the best feature at each step.

- Set of features F initially $\varnothing$.
- At each iteration, find the best new feature where the error E after inclusion is minimum. $j = \text{argmin}_i E(F \cup x_i)$.
- Add x_j to F if $E(F \cup x_j) < E(F)$.

Backward Search: Start with all the features and remove one at a time, if possible.

Floating Search: Add k features, remove l features at every iteration.

16.2 Math Review - Eigenvalues & Eigenvectors

Let $\boldsymbol{A}$ be a square matrix and vector $\boldsymbol{x}$ is said to be an eigenvector of the matrix if:

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$$

where the constant λ is called the eigenvalue.

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\boldsymbol{x} = 0$$

The above equation has a non-zero solution for $\boldsymbol{x}$ if and only if:

$$|\boldsymbol{A} - \lambda \boldsymbol{I}| = 0$$

$$|\boldsymbol{A}| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$|\boldsymbol{A}| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

The eigenvalue is obtained by solving the characteristic equation above.

16.3 Principal Component Analysis (PCA)

The aim of PCA is to find a lower dimensional space such that when the signal $\boldsymbol{x}$ is projected there, the information loss is minimized. The information in the signal is represented by its variance. PCA finds a set of projection vectors that maximize the variance of the signal in the projected space.

A projection of $\boldsymbol{x}$ in the direction of $\boldsymbol{w}$ is given by:

$$\boldsymbol{z} = \boldsymbol{w}^T \boldsymbol{x} = |\boldsymbol{w}||\boldsymbol{x}| \cos(\theta)$$
$$|\boldsymbol{w}| = 1$$

Given a set of patterns $\boldsymbol{x}_p$ where $p = 1, 2, \dots P$, the goal is to find the unit vector $\boldsymbol{w}$ that maximizes the covariance $\text{cov}(z)$.

$$
\begin{aligned}
\operatorname{cov}(z) &= \frac{1}{P-1}\sum_{p=1}^{P}(z_p-\mu_z)^2 \\
&= \frac{1}{P-1}\sum_{p=1}^{P}(z_p-\mu_z)(z_p-\mu_z)^T \\
&= \frac{1}{P-1}\sum_{p=1}^{P}(\boldsymbol{w}^T\boldsymbol{x}_p-\boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x}_p-\boldsymbol{w}^T\boldsymbol{\mu})^T \\
&= \boldsymbol{w}^T\frac{1}{P-1}\sum_{p=1}^{P}(\boldsymbol{x}_p-\boldsymbol{\mu})(\boldsymbol{x}_p^T-\boldsymbol{\mu}^T)\boldsymbol{w} \\
&= \boldsymbol{w}^T\frac{1}{P-1}\sum_{p=1}^{P}(\boldsymbol{x}_p-\boldsymbol{\mu})(\boldsymbol{x}_p-\boldsymbol{\mu})^T\boldsymbol{w} \\
&= \boldsymbol{w}^T\boldsymbol{C_{xx}}\boldsymbol{w}
\end{aligned}
$$

$\boldsymbol{C_{xx}}$ represents the covariance matrix and the aim of PCA is to find the $\boldsymbol{w}$ that satisfies:

$$\max_{\boldsymbol{w}} \boldsymbol{w}^T\boldsymbol{C_{xx}}\boldsymbol{w} \text{ such that } ||\boldsymbol{w}|| = 1$$

This can be done by minimizing the cost:

$$J(\boldsymbol{w}) = -\boldsymbol{w}^T\boldsymbol{C_{xx}}\boldsymbol{w} + \lambda(\boldsymbol{w}^T\boldsymbol{w}-1)$$

After solving for $\frac{\delta J(\boldsymbol{w})}{\delta \boldsymbol{w}} = 0$:

$$
\begin{aligned}
-2\boldsymbol{C_{xx}}\boldsymbol{w} + 2\lambda\boldsymbol{w} &= 0 \\
\boldsymbol{C_{xx}}\boldsymbol{w} &= \lambda\boldsymbol{w}
\end{aligned}
$$

Thus, the projection vectors are given by the eigenvectors of the covariance matrix $\boldsymbol{C_{xx}}$. The top principal eigenvector (that corresponds to the largest eigenvalue) gives the direction of maximum variance. The second principal eigenvector gives the direction of the next largest variance and so on.

Given a matrix of size $n \times n$, there are n eigenvectors and n corresponding eigenvalues. Let $\boldsymbol{u}_1, \boldsymbol{u}_2 \dots \boldsymbol{u}_n$ be the eigenvectors and $\lambda_1, \lambda_2 \dots \lambda_n$ be the corresponding eigenvalues in decreasing order. Thus, $\boldsymbol{u}_1$ is the first principal eigenvector and the second principal eigenvector is $\boldsymbol{u}_2$ and so on.

Eigenmatrix $\boldsymbol{U}$ contains the eigenvectors as columns with the principal eigenvector first, the second principal eigenvector next and so on:

$$\boldsymbol{U} = \begin{pmatrix}\boldsymbol{u}_1 & \boldsymbol{u}_2 & \dots & \boldsymbol{u}_n\end{pmatrix}$$

16.3.1 Projecting the Data

The data can now be represented in eigenspace by rotating the mean-corrected data by eigenvectors. A given data point $\boldsymbol{x}$ will be represented in eigenspace by $\boldsymbol{z}$:

$$\boldsymbol{z} = \boldsymbol{U}^T(\boldsymbol{x}-\boldsymbol{\mu})$$

where $\boldsymbol{z} = (z_1, z_2, \dots z_n)^T$. $z_1 = \boldsymbol{u_1}^T(\boldsymbol{x}-\boldsymbol{\mu})$ is known as the first principal component of data, z_2 is known as the second principal component and so on.

Since the eigenmatrix is orthogonal (i.e. $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$), the data in eigenspace can be rotated back to original space:

$$
\begin{aligned}
\boldsymbol{U}\boldsymbol{z} &= \boldsymbol{U}\boldsymbol{U}^T(\boldsymbol{x}-\boldsymbol{\mu}) \\
&= \boldsymbol{x}-\boldsymbol{\mu} \\
\boldsymbol{x} &= \boldsymbol{U}\boldsymbol{z}+\boldsymbol{\mu}
\end{aligned}
$$

16.3.2 Dimensionality Reduction

The first principal component of the data carries most of the variance (information) of the data, the second principal component carries the next and so on. We can represent data in a lower k dimensional space loosing the least amount of information by using the top k principal components of the data.

Let $\tilde{\boldsymbol{z}} = (z_1, z_2, \dots z_k)^T$ be the data in the $k < n$ dimensional space and $\boldsymbol{U_k}$ be the eigenmatrix with the top k eigenvectors:

$$\tilde{\boldsymbol{z}} = \boldsymbol{U_k}^T(\boldsymbol{x}-\boldsymbol{\mu})$$

By transforming data into a lower dimensional space, we inevitably loose information. The reduced dimensional data can be projected back to the original space:

$$\hat{\boldsymbol{x}} = \boldsymbol{U_k}\tilde{\boldsymbol{z}}+\boldsymbol{\mu}$$

16.3.3 Choosing k

Proportion of Variance (PoV) can be used:

$$\operatorname{PoV}(k) = \frac{\lambda_1+\lambda_2+\ldots+\lambda_k}{\lambda_1+\lambda_2+\ldots+\lambda_k+\ldots+\lambda_n}$$

A PoV value greater than 0.9 is usually used.

16.4 Math Review - Spectral Decomposition

Let $\boldsymbol{X}$ be a data matrix, the spectral decomposition states:

$$\boldsymbol{S} = \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$$

where $\boldsymbol{U}$ contains eigenvectors of $\boldsymbol{S}$ as columns and $\boldsymbol{D}$ has the eigenvalues as diagonal elements. The data matrix $\boldsymbol{X}$ has data points as its rows.

When $\boldsymbol{X}$ is the mean-corrected data matrix, $\boldsymbol{S}$ is proportional to the covariance matrix of the data. The spectral decomposition (AKA eigenvalue decomposition) is true for any real symmetric matrix $\boldsymbol{S}$.

16.5 Feature Embedding

When $\boldsymbol{X}$ is the $N \times d$ data matrix:

- $\boldsymbol{X}^T\boldsymbol{X}$ is the $d \times d$ matrix (covariance of features if mean-centered)
- $\boldsymbol{X}\boldsymbol{X}^T$ is the $N \times N$ matrix (pairwise similarities between instances)

PCA uses eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ which are d-dimensional and can be used for projection. Feature embedding uses the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$ which are N-dimensional and which give the coordinates after projection directly (without a projection vector).

Sometimes, it is possible to define pairwise similarities (or distances) between instances and use feature embedding without needing to represent the instances as vectors.

By spectral decomposition, $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{V}\boldsymbol{E}\boldsymbol{V}^T$, where $\boldsymbol{V}$ has eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$ as columns.

Feature embedding uses the significant eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$ as features — $k < N$. Feature embedding is preferred when $N < d$.

16.6 Math Review - Singular Value Decomposition

$$\boldsymbol{X} = \boldsymbol{V}\boldsymbol{A}\boldsymbol{U}^T$$

where $\boldsymbol{V}$ is $N \times N$ and contains the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{U}$ is $d \times d$ and contains the eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{A}$ is $N \times d$ and contains singular values on its first k diagonal. Singular values are square roots of eigenvalues.

$$\boldsymbol{X} = \boldsymbol{v_1} a_1 \boldsymbol{u_1}^T + \ldots + \boldsymbol{v_k} a_k \boldsymbol{u_k}^T$$

where k is the rank of $\boldsymbol{X}$.

By taking a reduced number of eigenvectors, $\boldsymbol{X} \approx \boldsymbol{V_r}\boldsymbol{A_r}\boldsymbol{U_r}^T$. That is, SVD can be used for matrix factorization.

16.7 Multidimensional Scaling

Given pairwise distances between N points, d_{ij} where $i, j = 1, \ldots N$, they can be placed on a low-dimensional map such that the distances are preserved (by feature embedding).

1. Let distance matrix be $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{X}^T$.
2. From spectral decomposition, $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$.
3. Thus, $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}^{0.5}$ is an approximation considering only k ($<< d$ and N) eigenvectors.

MDS preserves the original distances between data points in the lower dimensional space.

16.8 Linear Discriminant Analysis

The goal is to find a low-dimensional space such that when $\boldsymbol{x}$ is projected, classes are well-separated.

16.8.1 Fisher's Linear Discriminant

16.8.1.1 $K = 2$ Classes

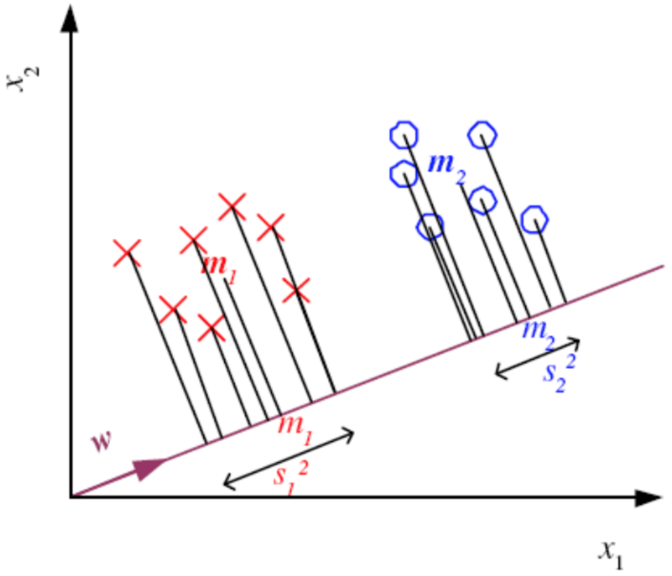


Figure 16.1: Linear Discriminant Analysis

Find $\boldsymbol{w}$ that maximizes the objective function (Fisher Ratio):

$$J(\boldsymbol{w}) = \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2}$$

where mean m_1 and scatter s_1^2 correspond to Class 1.

$$m_1 = \frac{1}{|C_1|} \sum_{x \in C_1} \boldsymbol{w}^T \boldsymbol{x}$$

$$s_1^2 = \sum_{x \in C_1} (\boldsymbol{w}^T \boldsymbol{x} - m_1)^2$$

The between-class scatter $\boldsymbol{S}_B$:

$$\boldsymbol{S}_B = (\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^T$$

The within-class scatter $\boldsymbol{S}_W$:

$$\boldsymbol{S}_W = \sum_{x \in C_1} (\boldsymbol{x} - \boldsymbol{m_1})(\boldsymbol{x} - \boldsymbol{m_1})^T + \sum_{x \in C_2} (\boldsymbol{x} - \boldsymbol{m_2})(\boldsymbol{x} - \boldsymbol{m_2})^T$$

Thus, find $\boldsymbol{w}$ that maximizes:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}} = \frac{|\boldsymbol{w}^T(\boldsymbol{m_1} - \boldsymbol{m_2})|^2}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$$

Thus, the LDA solution, $\boldsymbol{w}$ (i.e. Fisher's linear discriminant line), is given by:

$$\boldsymbol{w} = c \cdot \boldsymbol{S}_W^{-1}(\boldsymbol{m_1} - \boldsymbol{m_2})$$

where $c = 1.0$ is a constant.

16.8.1.2 $K > 2$ Classes

The between-class scatter $\boldsymbol{S}_B$:

$$\boldsymbol{S}_B = \sum_{k=1}^{K} |C_k| (\boldsymbol{m_k} - \boldsymbol{m})(\boldsymbol{m_k} - \boldsymbol{m})^T$$

where $\boldsymbol{m} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{m_k}$.

The within-class scatter $\boldsymbol{S}_W$:

$$\boldsymbol{S}_W = \sum_{k=1}^{K} \sum_{x \in C_k} (\boldsymbol{x} - \boldsymbol{m_k})(\boldsymbol{x} - \boldsymbol{m_k})^T$$

Find $\boldsymbol{W}$ that maximizes:

$$J(\boldsymbol{W}) = \frac{|\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}|}{|\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}|}$$

The largest eigenvectors $\boldsymbol{S}_W^{-1} \boldsymbol{S}_B$ gives the solution.

16.9 Notes

- PCA preserves the variance (information).
- LDA aims to separate classes.

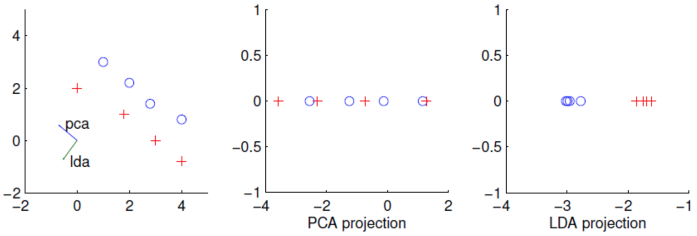


Figure 16.2: PCA vs. LDA

Chapter 17

Non-Parametric Density Estimation

In Machine Learning, we try to estimate $p(x|\omega_i, D_i)$. This approach requires some knowledge about $p(x|\omega_i, D_i)$ (e.g. the form of the distribution) and only the parameters need to be estimated. However, if we do not have any knowledge about the distribution, we need to use non-parametric density estimation.

17.1 Histograms

A histogram divides the sample space into a number of bins and approximates the density at the center of each bin by the fraction of points in the training data that falls into the corresponding bin.

Histogram estimator:

$$P_H(x) = \frac{1}{N} \frac{[\text{number of } x^k \text{ in the same bin as } x]}{[\text{width of bin containing } x]}$$

where x^k is one training sample and N is the total number of training samples.

Histograms require two parameters, bin width and starting position of the first bin.

$$\int P_H dx = \sum P_H(x_c) \times P_W = 1$$

where $P_H(x_c)$ is the height of each bin and P_W is the width.

17.1.1 Pros/Cons

Pros:

- Easy to understand.
- Easy to implement.

Cons:

- The estimated density depends on the starting position of the bins and bin width.
- The discontinuities of the estimate may not be due to the underlying density. They are an artifact of the chosen bin locations.
- These discontinuities make it very difficult to grasp the structure of the data.
- The most serious problem is the curse of dimensionality, since the number of bins grows exponentially with the number of dimensions.
    - For example, a five-dimension problem where each dimension uses 10 bins would result in 10^5 hypercubes.

17.2 General Theory for Non-Parametric Density Estimation

The probability that a vector x, drawn from a distribution $p(x)$, will fall in a given region R of the same space is:

$$P = \int_{R_{\text{lower}}}^{R_{\text{upper}}} p(u)du$$

Assume that we have N sample points $x_1, x_2, \ldots x_N$ from the distribution that we want to estimate. The probability that k of these N sample points fall in R is given by the binomial distribution:

$$P(k) = \binom{N}{k} P^k (1-P)^{N-k}$$

where:

$$\binom{N}{k} = \frac{N!}{(N-k)!k!}$$

The mean of the binomial distribution is $\mathrm{E}(k) = NP$ and variance is $\mathrm{Var}(k) = NP(1-P)$. Therefore, it can be proved that:

$$\mathrm{E}(k/N) = P$$
$$\mathrm{Var}(k/N) = P(1-P)/N$$

Therefore, when $N \to \infty$, $P(1-P)/N \approx 0$. This means that the distribution of k/N becomes very sharp at $k/N = P$. Therefore, we can use k/N to approximate P.

If we assume R is very small and $p(x)$ does not change too much in R:

$$P = \int_R p(x')dx' \approx p(x)V$$

where V is the volume enclosed by region R.

Combining the results,

$$P = \frac{k}{N} \approx p(x)V$$
$$p(x) \approx \frac{k}{NV}$$

where p is the probability density that we want to estimate, V is the volume surrounding x, N is the total number of examples and k is the number of examples inside V.

Comparing to the histogram estimator equation for $P_H(x)$, $k =$ [number of x^k in the same bin as x] and $V =$ [width of bin containing x].

We have two approaches to apply this result:

1. We can select a fixed value of the volume V and determine k from the data i.e. Kernel Density Estimation (KDE).
2. We can select a fixed value of k and determine the corresponding volume V from the data i.e. k Nearest Neighbor (kNN) Approach.

17.3 Parzen Windows

Let R be a hypercube with sides of length h centered at the estimation point x. Therefore, the volume is:

$$V = h^d$$

where d is the number of dimensions.

To count the number of sample points that fall inside the hypercube, we define a kernel function $K(u)$ where $u = (u_1, u_2, \dots u_d)$:

$$K(u) = \begin{cases} 1 & |u_j| < \frac{1}{2} \ \forall j = 1, 2, \dots d \\ 0 & \text{otherwise} \end{cases}$$

The kernel, which corresponds to a unit hypercube centered at the origin is knows as Parzen window or the naive estimator.

Then, $K((x - x^i)/h)$ is equal to one if $|(x - x^i)| < h/2$. In other words, the point x^i is inside a hypercube of side h centered on x.

The total number of points at x inside the hypercube is:

$$k = \sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right)$$

Now that we know k, V & N:

$$p_{\text{KDE}}(x) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right)$$

Notice that the Parzen window density estimate resembles the histogram, with the exception that the bin locations are determined by the data points.

We can calculate the probability of the original distribution at x by:

$$p(x) = p_{\text{KDE}}(x) \times h^d$$

17.3.1 Drawbacks

- The estimated density has discontinuities.
- The weights of x_i are equal, regardless of their distance to the estimation point x.

17.4 Smooth Kernels

To overcome the drawbacks of Parzen window, the hypercube in the original Parzen window can be replaced with other smooth kernel functions $K(u)$ such that:

$$\int_R K(x)dx = 1$$

Usually, the selected kernel function fulfills the following conditions:

- Smooth (also means differentiable)
- Symmetric
- Unimodal

Therefore, the multivariate Gaussian density function can be used:

$$K(x) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}x^T x\right)$$

The Parzen window approach can be considered as a sum of boxes centered at the observations while the smooth kernel approach can be considered as a sum of *bumps* placed at the data points. The shape of the bumps is determined by the kernel function. The parameter h, also called the smoothing parameter or bandwidth determines the width.

17.5 Bandwidth Selection

The problem of choosing the bandwidth is crucial in the density estimation, for smooth kernel functions and histograms alike.

- A large bandwidth will over-smooth the density and mask the structure in the data.
- A small bandwidth will yield a density estimate that is spiky and very hard to interpret.

We would like to find a value of the smoothing parameter that minimizes the error between the estimated density ($p_{\text{KDE}}(x)$) and the true density. A commonly used error metric is the mean square error at the estimation point x:

$$\begin{aligned}\text{MSE}(p_{\text{KDE}}(x)) &= E[(p_{\text{KDE}}(x) - p(x))^2] \\ &= (E(p_{\text{KDE}}(x) - p(x)))^2 + \text{Var}(p_{\text{KDE}}(x))\end{aligned}$$

where $p_{\text{KDE}}(x)$ is a random variable, however, $p(x)$ is a constant.

This expression is an example of the bias-variance tradeoff. Reducing the bias would increase variance and vice-versa.

- Bias shows the expected difference of the estimation.
- Variance shows the variance of the estimation.

The bias-variance dilemma applied to bandwidth selection means that:

- A large bandwidth will reduce the differences among the estimates of $p_{\text{KDE}}(x)$ for different datasets (i.e. the variance) but it will increase the bias of $p_{\text{KDE}}(x)$ with respect to the true density $p(x)$.
- A small bandwidth will reduce the bias of $p_{\text{KDE}}(x)$ at the expense of a larger variance in the estimates of $p_{\text{KDE}}(x)$.

17.5.1 Subjective Choice

The natural way for choosing the smoothing parameter is to plot out several curves and choose the estimate that is most in accordance with one's prior (subjective) ideas. However, this method is not practical in pattern recognition since we typically have high dimensional data.

17.5.2 Reference to Standard Distribution

Assume a standard density function and find the value of the bandwidth that minimizes the integral of the square error (MISE):

$$\begin{aligned}h_{\text{opt}} &= \underset{h}{\text{argmin}}\{\text{MISE}(p_{\text{KDE}}(x))\} \\ &= \underset{h}{\text{argmin}}\left\{E\left[\int (p_{\text{KDE}}(x) - p(x))^2 dx\right]\right\}\end{aligned}$$

If we assume that the true distribution is a Gaussian density and we use a Gaussian kernel, it can be shown that the optimal value of the bandwidth becomes $h_{\text{opt}} = 1.06\sigma N^{-1/5}$, where σ is the sample variance and N is the number of training examples.

17.5.3 Likelihood Cross-Validation

If we consider the estimation of the bandwidth, h, to be a parametric estimation problem, we can use an approach based on cross-validation. The final h is defined as:

$$h_{\text{MLCV}} = \underset{h}{\text{argmax}}\left\{\frac{1}{N}\sum_{i=1}^{N}\log(p_{-i}(x^i))\right\}$$

where p_{-n} is computed from leave-one-out cross-validation:

$$p_{-n}(x) = \frac{1}{(N-1)h}\sum_{i=1, i\neq n}^{N} K\left(\frac{x^i - x}{h}\right)$$

17.6 Multivariate Density Estimation

Till now, for one dimension or high dimensional problems, we use the same equation for $p_{\text{KDE}}(x)$. Thus, it should be noted that the bandwidth h is the same for all axes. Hence, this density estimate will weigh all the axes equally.

$$p_{\text{KDE}}(x) = \frac{1}{Nh^d}\sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right)$$

However, if the spread of the data is much greater in one of the coordinate directions than the others, we should use a wider bandwidth for that direction. Thus, the above equation is modified to:

$$p_{\text{KDE}}(x) = \frac{1}{N}\sum_{i=1}^{N} K(x, x^i, h_1, h_2, \dots h_d)$$

$$K(x, x^i, h_1, h_2, \dots h_d) = \frac{1}{h_1 \dots h_D}\prod_{r=1}^{d} K_1\left(\frac{x(r) - x^i(r)}{h_r}\right)$$

where K_1 is a one-dimensional kernel.

17.7 k Nearest Neighbor

In the kNN method, we grow the volume surrounding the estimation point x until it encloses a total of k data points. The density estimate for kNN becomes:

$$p(x) \approx \frac{k}{NV} = \frac{k}{N \times c_d \times R_k^d(x)}$$

where $R_k^d(x)$ is the distance between the estimation point x and its k-th closest neighbor in d dimensions, c_d is the volume of the unit sphere in d dimensions.

$$c_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

$$c_d = \begin{cases} 2 & d=1 \\ \pi & d=2 \\ \frac{4\pi}{3} & d=3 \end{cases}$$

where $\Gamma(d/2+1)$ is called the gamma function defined as:

$$\Gamma(m) = 2\int_0^\infty e^{-r^2} r^{2m-1} dr$$

17.7.1 Disadvantages

In general, the estimates obtained with the kNN method are not very satisfactory.

- The estimates are prone to local noise.
- The method produces estimates with very heavy tails.
- Since the function $R_k(x)$ is not differentiable, the density estimate will have discontinuities.
- The resulting density is not a true probability density since its integral over all the sample space diverges.

17.7.2 Approximation to Bayes Classifier

The main advantage of the kNN method is that it leads to a very simple approximation of the (optimal) Bayes classifier.

Assume a dataset of N examples, N_i from class ω_i, and an unknown sample x_u that needs to be classified. By drawing a hypersphere of volume V around x_u that contains a total of k examples, k_i from class ω_i, it is possible to approximate the likelihood functions using the kNN method by:

$$p(x|\omega_i) = \frac{k_i}{N_i V}$$

Similarly, the unconditional density is estimated by:

$$p(x) = \frac{k}{NV}$$

The priors are approximated by:

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

Therefore,

$$\begin{aligned} p(\omega_i|x) &= \frac{p(x|\omega_i)p(\omega_i)}{p(x)} \\ &= \frac{\frac{k_i}{N_i V}\frac{N_i}{N}}{\frac{k}{NV}} \\ &= \frac{k_i}{k} \end{aligned}$$

17.8 Curse of Dimensionality

For high dimensional problems, it is not possible to directly use the non-parametric approach to estimate $p(x|\omega_i)$.

17.9 Naive Bayes Classifier

To simplify the estimation for $p(x|\omega_i)$, Naive Bayes Classifier was developed. It employs the assumption that the features are class-conditionally independent across dimensions i.e.:

$$p(x|\omega_i) = \prod_{d=1}^{D} p(x(d)|\omega_i)$$

Note that this assumption is different from independent features:

$$p(x) = \prod_{d=1}^{D} p(x(d))$$

Therefore, the Bayes decision rule $j = \underset{i}{\operatorname{argmax}}(p(x|\omega_i)p(\omega_i))$ can be rewritten as:

$$j = \underset{i}{\operatorname{argmax}}\left(\prod_{d=1}^{D} p(x(d)|\omega_i)p(\omega_i)\right)$$

This is the Naive Bayes Classifier.

Chapter 18

Graphical Models

Graphical models are a combination of probability theory and graph theory.

Advantages:

- Framework for designing and analyzing probabilistic models (causal/diagnostic inference)
- Perform learning tasks (parameter/structure estimation from training set)
- Efficient software implementation (conditional independence properties)

A probabilistic problem can be visualized as a directed acyclic graph where **nodes** are the random variables and **edges** represent the conditional probabilities.

18.1 Factorization

Any joint probability of a directed acyclic graph (i.e. Bayesian network) can be factorized as a product of conditional distributions:

$$P(X_1, \dots X_n) = \prod_i P(X_i | pa(X_i))$$

where $pa(X_i)$ are the parents of X_i on the Bayesian network.

18.2 Independence

X and Y are independent if:

$$P(X, Y) = P(X)P(Y)$$

X and Y are conditionally independent given Z if:

$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$
$$P(X|Y, Z) = P(X|Z)$$

There are two consequences of conditional independence:

1. Factorization implies that each variable X_i is conditionally independent of all its non-descendants given its parents.
2. Conditional independence tells us when two variables in a BN are conditionally independent given another variable. There are three possible cases: tail-to-tail, head-to-tail, head-to-head.

18.2.1 Tail-to-Tail Node (Common Cause)

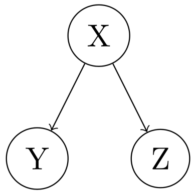


Figure 18.1: Tail-to-Tail

If X is unobserved:

$$\begin{aligned} P(Y, Z) &= \sum_X P(X, Y, Z) \\ &= \sum_X P(X)P(Y|X)P(Z|X) \\ &\neq P(Y)P(Z) \end{aligned}$$

If X is observed:

$$P(Y, Z|X) = P(Y|X)P(Z|X)$$

A tail-to-tail node X blocks the path from Y to Z if it is observed i.e. Y, Z become independent.

18.2.2 Head-to-Tail Node (Causal Effect)

If Y is unobserved:

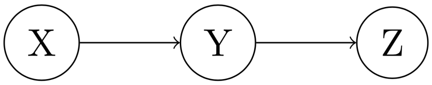


Figure 18.2: Head-to-Tail

$$
\begin{aligned}
P(X,Z) &= \sum_{Y} P(X,Y,Z) \\
&= \sum_{Y} P(X)P(Y|X)P(Z|Y) \\
&\neq P(X)P(Z)
\end{aligned}
$$

If Y is observed:

$$P(X,Z|Y) = P(X|Y)P(Z|Y)$$

A head-to-tail node Y blocks the path from X to Z if it is observed i.e. X, Z become independent.

18.2.3 Head-to-Head Node (Common Effect)

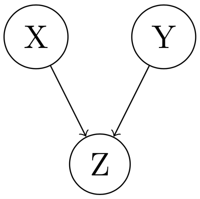


Figure 18.3: Head-to-Head

If Z is unobserved:

$$
\begin{aligned}
P(X,Y) &= \sum_{Z} P(X,Y,Z) \\
&= \sum_{Z} P(X)P(Y)P(Z|X,Y) \\
&= P(X)P(Y)\sum_{Z} P(Z|X,Y) \\
&= P(X)P(Y)
\end{aligned}
$$

If Z is observed:

$$P(X,Y|Z) \neq P(X|Z)P(Y|Z)$$

A head-to-head node Z unblocks the path from X to Y if it is observed i.e. X, Y become dependent.

18.3 Inference in Bayesian Networks

- Inference = answering a query in a Bayesian network
  - Computation of conditional distributions $P(Q|e)$
  - Q = set of query variables
  - e = evidence (instantiation of a random variable E)
- Causal Inference: Reasoning in the direction of the Bayesian network.
- Diagnostic Inference: Reasoning against the direction of the Bayesian network.

The structure of the Bayesian network is used to answer queries efficiently using factorization and the rules of probability.

18.3.1 Computational Complexity

Consider n binary variables $(X_1, \dots X_n)$.

- Full unconstrained joint distribution $P(X_1, \dots X_n)$ requires $O(2^n)$ probabilities. Thus, inference is usually intractable in the general case.
- If we have a Bayesian network, with a maximum of k parents for any node, then we need $O(n\,2^k)$ probabilities to estimate $P(X_1, \dots X_n)$.

18.4 d-Separation

Given an arbitrary subset of nodes, A, B and C, a path from nodes in A to nodes in B is blocked (d-separated) by nodes in C if one of the following conditions is satisfied:

1. The directions of edges meet head-to-tail and the node is in C.
2. The directions of edges meet tail-to-tail and the node is in C.
3. The directions of edges meet head-to-head and neither that node or any of its descendants is in C.

If all paths are blocked, A and B are d-separated (i.e. conditionally independent) given C.