Data mining

Giacomo Fantoni
Telegram: @GiacomoFantoni
Github: https://github.com/giacThePhantom/DataMining

January 6, 2022

Contents

1 Introduction **10**
- 1.1 Formalization of a machine learning problem . . . 10
  - 1.1.1 Components of a machine learning problem . . . 10
  - 1.1.2 Designing a machine learning system . . . 10
- 1.2 Learning settings . . . 11
  - 1.2.1 Supervised learning . . . 11
  - 1.2.2 Unsupervised learning . . . 12
  - 1.2.3 Semi-supervised learning . . . 12
  - 1.2.4 Reinforcement learning . . . 12
- 1.3 Probabilistic reasoning . . . 13
- 1.4 Choice of learning algorithms . . . 13
  - 1.4.1 Based on information available . . . 13

2 Decision trees learning **14**
- 2.1 Introduction . . . 14
  - 2.1.1 Appropriate problems for decision trees . . . 14
- 2.2 Learning decision trees . . . 14
  - 2.2.1 Greedy top-down strategy . . . 14
  - 2.2.2 Choosing the best attribute . . . 15
- 2.3 Issues in decision tree learning . . . 15
  - 2.3.1 Overfitting avoidance . . . 15
  - 2.3.2 Post-pruning . . . 15
  - 2.3.3 Dealing with continues valued attributes . . . 16
  - 2.3.4 Alternative attribute test measures . . . 16
  - 2.3.5 Handling attributes with missing values . . . 16
- 2.4 Random forest . . . 17
  - 2.4.1 Training . . . 17
  - 2.4.2 Testing . . . 17

3 K-nearest neighbours **18**
- 3.1 Introduction . . . 18
- 3.2 Measuring the instance between instances . . . 18
  - 3.2.1 Metric or distance definition . . . 18
  - 3.2.2 Euclidean distance . . . 18
- 3.3 Algorithms . . . 19
  - 3.3.1 Classification . . . 19

3.3.2 Regression . . . 19
3.4 Characteristics . . . 19
3.5 Distance weighted k-nearest neighbour . . . 19

4 Linear algebra **21**
4.1 Vector space . . . 21
4.1.1 Properties and operations . . . 21
4.1.2 Basis . . . 22
4.2 Matrices . . . 22
4.2.1 Linear maps . . . 22
4.2.2 Linear maps as matrices . . . 22
4.2.3 Matrix properties . . . 23
4.2.4 Matrix derivatives . . . 24
4.2.5 Metric structure . . . 24
4.2.6 Dot product . . . 24
4.3 Eigenvalues and eigenvectors . . . 25
4.3.1 Cardinality . . . 25
4.3.2 Singular matrices . . . 25
4.3.3 Symmetric matrices . . . 25
4.3.4 Eigen-decomposition . . . 26
4.4 Principal component analysis . . . 27
4.4.1 Procedure . . . 28
4.4.2 Dimensionality reduction . . . 28

5 Probability theory **29**
5.1 Discrete random variables . . . 29
5.1.1 Probability mass function . . . 29
5.1.2 Expected value . . . 29
5.1.3 Variance . . . 29
5.1.4 Properties of mean and variance . . . 30
5.1.5 Probability distributions . . . 30
5.1.6 Pairs of discrete random variables . . . 31
5.2 Conditionally probability . . . 32
5.2.1 Basic rules . . . 32
5.2.2 Bayes' rule . . . 32
5.3 Continuous random variables . . . 33
5.3.1 Cumulative distribution function . . . 33
5.3.2 Probability density function . . . 33
5.3.3 Probability distribution . . . 33
5.4 Probability laws . . . 35
5.4.1 Expectation of an average . . . 35
5.4.2 Variance of an average . . . 35
5.4.3 Chebyshev's inequality . . . 35
5.4.4 Law of large numbers . . . 36
5.4.5 Central limit theorem . . . 36
5.5 Information theory . . . 36
5.5.1 Entropy . . . 36
5.5.2 Cross entropy . . . 37

5.5.3 Relative entropy . . . 37
5.5.4 Conditional entropy . . . 37
5.5.5 Mutual information . . . 37

6 Evaluation **38**
6.1 Introduction . . . 38
6.2 Performance measures . . . 38
6.2.1 Training loss and performance measures . . . 38
6.2.2 Binary classification . . . 38
6.2.3 Multiclass classification . . . 40
6.2.4 Regression . . . 40
6.3 Hypothesis testing . . . 41
6.3.1 Test statistic . . . 41
6.3.2 Glossary . . . 42
6.3.3 T-test . . . 42
6.3.4 Comparing learning algorithms . . . 42

7 Bayesian decision theory **44**
7.1 Introduction . . . 44
7.1.1 Input-output pairs . . . 44
7.1.2 Expected error . . . 44
7.1.3 Bayes decision rule . . . 45
7.2 Representing classifiers . . . 45
7.2.1 Discriminant functions . . . 45
7.2.2 Decision regions . . . 45
7.3 Multivariate normal density . . . 46
7.3.1 Hyperellipsoids . . . 46
7.3.2 Discriminant functions for normal density . . . 46
7.4 Arbitrary inputs and outputs . . . 48
7.4.1 Setting . . . 48
7.4.2 Risk . . . 48
7.4.3 Bayes decision rule . . . 48
7.5 Handling features . . . 48
7.5.1 Handling missing features - marginalize over missing variables . . . 48
7.5.2 Handling noisy features - marginalize over true variables . . . 49

8 Parameter estimation **50**
8.1 Introduction . . . 50
8.1.1 Setting . . . 50
8.1.2 Task . . . 50
8.1.3 Multi class classification . . . 50
8.2 Maximum likelihood . . . 51
8.2.1 Setting . . . 51
8.2.2 Maximizing log-likelihood . . . 51
8.2.3 Univariate Gaussian case . . . 51
8.2.4 Multivariate Gaussian case . . . 52
8.2.5 General Gaussian case . . . 54
8.3 Bayesian estimation . . . 54

8.3.1 Setting . . . 55
8.3.2 Univariate normal case - unknown μ, known σ^2 . . . 55
8.3.3 Multivariate normal case - unknown μ, known Σ . . . 56
8.3.4 Gamma distribution . . . 57
8.3.5 Univariate normal case - unknown μ and $\lambda = \frac{1}{\sigma^2}$ . . . 57
8.3.6 Wishart distribution . . . 58
8.3.7 Multivariate normal case - unknown μ and Σ . . . 58
8.4 Sufficient statistics . . . 59
8.4.1 Definition . . . 59
8.4.2 Conjugate priors . . . 59
8.5 Bernoulli distribution . . . 59
8.5.1 Setting . . . 59
8.5.2 Maximum likelihood estimation . . . 59
8.5.3 Bayesian estimation . . . 60
8.6 Multinomial distribution . . . 60
8.6.1 Setting . . . 60
8.6.2 Maximum likelihood estimation . . . 60
8.6.3 Bayesian estimation . . . 61

9 Bayesian networks **62**
9.1 Introduction . . . 62
9.1.1 Bayesian networks semantics . . . 62
9.1.2 Graph and distributions . . . 62
9.1.3 Factorization . . . 63
9.1.4 Bayesian network definition . . . 64
9.2 Conditional independence . . . 64
9.2.1 D separation . . . 64
9.2.2 General d-separation criterion . . . 66
9.3 BN independence revisited . . . 66
9.3.1 Independence assumptions . . . 66
9.3.2 BN equivalence classes . . . 66
9.3.3 I-maps and distributions . . . 67
9.3.4 Building Bayesian networks . . . 67
9.4 Markov blanket or boundary . . . 67
9.4.1 Definition . . . 67
9.4.2 d-separation . . . 68

10 Bayesian networks inference **69**
10.1 Inference in graphical models . . . 69
10.1.1 Efficiency . . . 69
10.1.2 Inference on a chain . . . 69
10.1.3 Inference as message passing . . . 70
10.1.4 Full message passing . . . 70
10.1.5 Adding evidence . . . 70
10.1.6 Computing conditional probability given evidence . . . 71
10.1.7 Inference on trees . . . 71
10.2 Factor graphs . . . 71
10.2.1 Description . . . 71

10.2.2 Sum-product algorithm . . . 71
10.3 Finding the most probable configuration . . . 72
10.3.1 The max-product algorithm . . . 73
10.4 Approximate inference . . . 74
10.4.1 Underflow issues . . . 74
10.4.2 Exact inference on general graphs . . . 74
10.4.3 Problem with exact inference . . . 74
10.4.4 Loopy belief propagation . . . 74
10.4.5 Variational methods . . . 74
10.4.6 Sampling methods . . . 75
10.5 Markov chain . . . 75
10.5.1 Definition . . . 75
10.5.2 Homogeneous chains . . . 75
10.5.3 Chain dynamics . . . 75
10.5.4 Convergence . . . 76
10.5.5 Stationary distribution . . . 76
10.5.6 Requirements . . . 76
10.5.7 Regular chains . . . 76
10.5.8 Markov chains for graphical models . . . 76
10.5.9 Transition model . . . 76
10.5.10 Gibbs sampling . . . 77
10.5.11 Computing local transition probabilities . . . 77
10.5.12 Generating samples . . . 77

11 Learning Bayesian networks **78**
11.1 Parameter estimation . . . 78
11.1.1 Maximum likelihood estimation - complete data . . . 78
11.1.2 Learning graphical models - adding priors . . . 79
11.1.3 Learning with missing data . . . 80
11.2 Learning the structure of graphical models . . . 80
11.2.1 Model averaging approach . . . 80
11.2.2 Constraint based approach . . . 81
11.2.3 Score based approach . . . 81

12 Naive Bayes classifier **84**
12.1 Setting . . . 84
12.1.1 Learning problem . . . 84
12.2 Definition . . . 84
12.2.1 Single distribution case . . . 84
12.3 Parameters learning . . . 85
12.4 Examples . . . 85
12.4.1 Text classification . . . 85

13 Linear discriminant functions **87**
13.1 Discriminative learning . . . 87
13.1.1 Pros of discriminative learning . . . 87
13.1.2 Cons of discriminative learning . . . 87
13.2 Linear discriminant functions . . . 87

13.2.1 Description . . . 87
13.2.2 Linear binary classifier . . . 87
13.3 Perceptron . . . 88
13.3.1 Biological motivation . . . 88
13.3.2 Representational power . . . 89
13.3.3 Augmented feature or weight vectors . . . 89
13.3.4 Parameter learning . . . 89
13.3.5 Perceptron training rule . . . 89
13.3.6 Stochastic perceptron training rule . . . 90
13.3.7 Perceptron regression . . . 90
13.4 Multiclass classification . . . 91
13.4.1 One-vs-all . . . 91
13.4.2 All-pairs . . . 91
13.5 Generative linear classifiers . . . 92
13.5.1 Gaussian distributions . . . 92
13.5.2 Naive Bayes classifier . . . 92

14 Support vector machines **93**
14.1 Introduction . . . 93
14.2 Maximum margin classifiers . . . 93
14.2.1 Classifier margin . . . 93
14.2.2 Canonical hyperplane . . . 93
14.3 Hard margin support vector machine . . . 94
14.3.1 Margin error bound theorem . . . 94
14.3.2 Learning problem . . . 94
14.3.3 Constrained optimization - Karush-Kuhn-Tucker (KKT) approach . . . 94
14.3.4 KKT approach in SVM . . . 95
14.3.5 Decision function . . . 95
14.3.6 KKT conditions . . . 96
14.3.7 Support vectors . . . 96
14.3.8 Decision function bias . . . 96
14.4 Soft margin SVM . . . 96
14.4.1 Slack variables . . . 96
14.4.2 Regularization theory . . . 97
14.4.3 Lagrangian . . . 97
14.4.4 Dual formulation . . . 97
14.4.5 Karush-Khun-Tucker conditions . . . 98
14.4.6 Support vectors . . . 98
14.5 Large-scale SVM learning . . . 98
14.5.1 Stochastic gradient descent . . . 98
14.5.2 Dual version . . . 99

15 Non linear support vector machines **100**
15.1 Non-linearly separable problems . . . 100
15.2 Non-linear support vector machines - feature map . . . 100
15.2.1 Polynomial mapping . . . 100
15.2.2 Linear separation in feature space . . . 101
15.3 Support vector regression . . . 101

15.3.1 ϵ-insensitive loss . . . 101
15.3.2 Optimization problem . . . 101
15.3.3 Lagrangian . . . 101
15.3.4 Dual formulation . . . 102
15.3.5 Regression function . . . 102
15.3.6 KKT conditions . . . 103
15.3.7 Support vectors . . . 103
15.4 Smallest enclosing hypersphere . . . 103
15.4.1 Optimization problem . . . 103
15.4.2 Lagrangian . . . 103
15.4.3 Dual formulation . . . 104
15.4.4 Distance function . . . 104
15.4.5 KKT conditions . . . 105
15.4.6 Support vectors . . . 105
15.4.7 Decision function . . . 105
15.5 Support vector ranking . . . 105
15.5.1 Optimization problem . . . 105
15.5.2 Support vector classification on pairs . . . 105
15.5.3 Decision function . . . 106

16 Kernel machines **107**
16.1 Kernel trick . . . 107
16.1.1 Support vector classification . . . 107
16.1.2 Polynomial kernel . . . 107
16.2 Valid kernels . . . 108
16.2.1 Dot product in feature space . . . 108
16.2.2 Gram matrix . . . 108
16.2.3 positive definite kernels . . . 109
16.2.4 Verifying kernel validity . . . 109
16.3 Support vector regression . . . 109
16.3.1 Dual problem . . . 109
16.3.2 Stochastic Perceptron . . . 109
16.3.3 Kernel Perceptron . . . 110
16.4 Kernels . . . 110
16.4.1 Basic kernels . . . 110
16.4.2 Gaussian kernel . . . 110
16.4.3 Kernels on structured data . . . 110
16.4.4 Kernels on sequences - spectrum kernel . . . 111
16.4.5 Kernel combination . . . 111
16.4.6 Kernels on graphs . . . 112

17 Deep networks **114**
17.1 Need for deep networks . . . 114
17.2 Multilayer perceptron . . . 114
17.2.1 Activation function . . . 114
17.2.2 Output layer . . . 115
17.2.3 Representational power of a multilayer perceptron . . . 115
17.2.4 Shallow and deep structures for boolean functions . . . 116

17.2.5 Training MLP . . . 116
17.2.6 Stopping criterion and generalization . . . 118
17.2.7 Vanishing gradient . . . 118
17.3 Trick of the trade . . . 118
17.3.1 Introduction . . . 118
17.3.2 Activation functions . . . 118
17.3.3 Regularization . . . 119
17.3.4 Initialization . . . 119
17.3.5 Gradient descent . . . 119
17.3.6 Adaptive gradient . . . 120
17.3.7 Batch normalization . . . 120
17.3.8 Pre-training . . . 121
17.4 Popular deep architectures . . . 121
17.4.1 Autoencoders . . . 121
17.4.2 Convolutional networks . . . 121
17.4.3 Long Short-Term Memory networks . . . 122
17.4.4 Generative adversarial networks . . . 122
17.4.5 Graph neural networks . . . 122

18 Unsupervised learning **123**
18.1 Setting . . . 123
18.2 K-means clustering . . . 123
18.2.1 Setting . . . 123
18.2.2 Algorithm . . . 123
18.2.3 Defining similarity . . . 123
18.3 Gaussian Mixture model GMM . . . 124
18.3.1 Parameter estimation . . . 124
18.3.2 Estimating means of k univariate Gaussians . . . 124
18.4 Expectation-Maximization (EM) . . . 125
18.4.1 Formal setting . . . 125
18.4.2 Generic algorithm . . . 125
18.4.3 Estimating means of k univariate Gaussians . . . 126
18.5 Choosing the number of clusters . . . 127
18.5.1 Elbow method . . . 127
18.5.2 Average silhouette method . . . 127
18.6 Hierarchical clustering . . . 128
18.6.1 Setting . . . 128
18.6.2 Top-down approach . . . 128
18.6.3 Bottom-up approach . . . 128
18.6.4 Dendograms . . . 128
18.6.5 Agglomeritive hierarchical clustering . . . 128
18.6.6 Measuring cluster similarities . . . 129
18.6.7 Stepwise optimal hierarchical clustering . . . 129

19 Reinforcement learning **130**
19.1 Introduction . . . 130
19.1.1 Setting . . . 130
19.1.2 Sequential decision making . . . 130
19.2 Markov decision process MPD . . . 130
19.2.1 Utilities over time . . . 131
19.2.2 Taking decisions . . . 131
19.2.3 Computing an optimal policy . . . 131
19.3 Dealing with partial knowledge . . . 133
19.3.1 Policy evaluation in an unknown environment . . . 133
19.3.2 Policy learning in an unknown environment . . . 134
19.4 Scaling to large state spaces . . . 135
19.4.1 Function approximation . . . 135
19.4.2 Learning the approximation function . . . 136

Chapter 1

Introduction

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T as measured by P, improves with experience E.
From that it is understood that machine learning is a form of inductive learning: it generalize from examples to a concept. There is no certainty of correctness.

1.1 Formalization of a machine learning problem

1.1.1 Components of a machine learning problem

The components of a machine learning problem are:

- Task to be addressed by the system.
- Performance measure to evaluate the learned system.
- Training experience to train the learning system.

1.1.2 Designing a machine learning system

The designing of a machine learning system can be described in a process that consist of different phases:

1. Formalize the learning task.
2. Collect data.
3. Extract features.
4. Choose class of learning models.
5. Train model.
6. Evaluate model.

1.1.2.1 Formalize the learning task

To formalize the learning task means to define the task that should be addressed by learning system. This type of task, or learning problem, is often composed of a number of related tasks, sub-problems or side-tasks. It is also needed an appropriate performance measure for evaluating the learned system.

1.1.2.2 Collect data

To collect the data means to collect a set of training example in machine readable format. Data collection is often the most cumbersome part of the process, implying manual intervention especially in labelling examples for supervised learning. Recent approaches to the problem of data labelling try to make use of the availability of unlabelled data (semi-supervised learning it tries to learn using both supervised and unsupervised examples).

1.1.2.3 Extract features

A relevant set of features need to be extracted from the data in order to provide inputs to the learning system. Prior knowledge is usually necessary in order to choose the appropriate features for the task in mind. Extracting too few features can miss relevant information preventing the system from learning the task with reasonable performance. Extracting too many features can make the learning problem harder and require a number of examples greater than those available for training. Another problem arises when considering noisy features. It is noticeable that there is a need to choose the correct number and type of feature to permit a correct and efficient solution.

1.1.2.4 Choose class of learning models

Every problem has a class of learning model that is able to learn it best. A simple model like a linear classifier is easy to train but insufficient for non linearly separable data. A too complex model can memorize noise in training data failing to generalize to new examples. The algorithm need not to optimize but it needs to generalize, there can be outliers or labelling error. The more complex the model the more it will tend to overfit training noise.

1.1.2.5 Train model

Training a model implies searching though the space of possible models given the chosen model class. Such search typically aims at fitting the available training examples well according to the chosen performance measure. However the learned model should perform well on unseen data (generalization) and not simply memorize training examples (overfitting). Different techniques can be used to improve generalization, usually by trading off model complexity with training set fitting.

1.1.2.6 Evaluate model

The learned model is evaluated according to its ability to generalize to unseen examples. These example are collected in a test set. Evaluation can provide insights into the model weaknesses and suggest directions for refining and modifying it. Evaluation can imply comparing different models or learners in order to decide the best performing one. Statistical significance of observed differences between performance of different models should be assessed with appropriate statistical tests.

1.2 Learning settings

1.2.1 Supervised learning

The learner is provided with a set of inputs/output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The learned model $f : x \to \mathcal{Y}$ should map input examples into their output. A domain expert is typically involved in labelling input examples with output examples in the training set.

1.2.1.1 Tasks

1.2.1.1.1 Classification

- Binary: assign an example to one of two possible classes, often a positive and a negative one.
- Multiclass: assign an example to one of $n > 2$ possible classes.
- Multilabel: assign an example to a subset $m \leq n$ of the possible classes.

1.2.1.1.2 Regression Assign a real value to an example.

1.2.1.1.3 Ordinal regression or ranking Order a set of examples according to their relative importance or quality with respect to the class.

1.2.2 Unsupervised learning

The learner is provided a set of input examples $x_i \in \mathcal{X}$ with no labelling information. The learner models training examples, for example clustering them together into clusters according to their similarity.

1.2.2.1 Tasks

1.2.2.1.1 Dimensionality reduction Reduce dimensionality of the data maintaining as much information as possible.

1.2.2.1.2 Clustering Cluster data into homogeneous groups according to their similarity.

1.2.2.1.3 Novelty detection Detect novel examples which differ from the distribution of a certain set of data.

1.2.3 Semi-supervised learning

The learner is provided with a set of input output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. A typically much bigger additional set of unlabelled examples $x_i \in \mathcal{X}$ is also provided. Like in supervised learning the learned model $f : \mathcal{X} \to \mathcal{Y}$ should map input examples into their output. Unlabelled data can be exploited to improve performance, by forcing the model to produce similar outputs for similar inputs, or by allowing to learn a better representation of examples.

1.2.4 Reinforcement learning

The learner is provided a set of possible states S and for each state a set of possible actions A moving it to a next state. In performing action a from state s the learner is provided an immediate reward $r(s, a)$. The task is to learn a policy allowing to choose for each state s the action a maximizing the overall reward. The learner has to deal with problems of delayed reward coming from future moves and trade-off between exploitation and exploration. Typical application include moving policies for robots and sequential scheduling problems in general.

1.3 Probabilistic reasoning

Probabilistic reasoning is the reasoning in presence of uncertainty. It evaluates the effect of a certain piece of evidence on other related variables. It estimates probabilities and relations between variables from a set of informations. They depends on variables and their relations.

1.4 Choice of learning algorithms

1.4.1 Based on information available

- Full knowledge of probability distributions of data: Bayesian decision theory.
- Form of probabilities known, parameters unknown: parameter estimation from training data.
- Form of probabilities unknown, training examples available: discriminative methods: do not model input data, learn a function predicting the desired output given the input.
- Form of probabilities unknown, training examples unavailable: unsupervised methods, cluster examples by similarity.

Chapter 2

Decision trees learning

2.1 Introduction

Decision trees tend to be interpretable, it is easy to see the reason for a certain decision. They represent a disjunction of conjunctions of constraints over attribute values. Each path from the root to a leaf is a conjunction of the constraints specified in the nodes along it: they can be seen as a disjunctive normal formula. In this way every class can be written as a DNF. The leaf contains the label to be assigned to instances reaching it. The disjunction of all paths is the logical formula expressed by the tree.

2.1.1 Appropriate problems for decision trees

The class of problems that can be solved by decision trees are:

- Binary or multiclass classification with an extension to regression (with a linear regression as the leaf).
- Instances represented as attribute-value pairs.
- Different explanations for the concept are possible (disjunction).
- Some instances have missing attributes, its dealing done with probabilistic models.
- There is need for an interpretable explanation of the output. In fact the main reason for using decision tress is interpretability.

2.2 Learning decision trees

2.2.1 Greedy top-down strategy

For each node, starting from the root with full training set:

1. Choose best attribute to be evaluated.
2. Add a child for each attribute value.
3. Split node training set into children according to value of chosen attribute.
4. Stop splitting a node if it contains examples from a single class or there are no more attributes to test.

It is also known as the divide et impera approach.

2.2.2 Choosing the best attribute

A measure to choose the attribute is entropy. It measures the amount of information contained in a collection of instances S which can take a number c of possible values:

$$H(s) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

Where p_i is the fraction of S taking value i. In our case instances are training examples and values are class labels. The entropy of a set of labelled examples measures its label's inhomogeneity.

2.2.2.1 Information gain

Expected reduction in entropy obtained by partitioning a set S according to the value of a certain attribute A.

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_V)$$

Where $Values(A)$ is the set of possible values taken by A and S_v is the subset of S taking value v at attribute A. The second term represents the sum of entropies of subsets of examples obtained partitioning over A values, weighted by their respective sizes. An attribute with high information gain tends to produce homogeneous groups in terms of labels, thus favouring their classification. The idea is to use the greedy strategy in which at each choice it is chosen the attribute with the maximum information gain.

2.3 Issues in decision tree learning

2.3.1 Overfitting avoidance

Requiring that each leaf has only examples of a certain class can lead to very complex trees. A complex tree can easily overfit the training set, incorporating random regularities not representative of the full distribution, or noise in the data. It is possible to accept impure leaves, assigning them the label of the majority of their training examples. Two possible strategies to prune a decision tree are:

- Pre-pruning; decide whether to stop splitting a node even if it contains training examples with different labels.
- Post-pruning: learn a full tree and then prune it.

2.3.2 Post-pruning

In post-pruning you expand the tree and then you prune it. It is introduced the validation set.

1. For each node in the tree evaluate the performance on the validation set when removing the subtree rooted at it.
2. If all node removals worsen performance, stop.
3. Choose the node whose removal has the

best performance inprovement.

4. Replace the subtree rooted at it with a leaf.
5. Assign to the leaf the majority label of all examples in the subtree.
6. Return to 1.

2.3.3 Dealing with continues valued attributes

Continuous valued attributes need to be discretized in order to be used in internal node tests. Discretization threshold can be chosen in order to maximize the attribute quality criterion.

1. Examples are sorted according to their continuous attribute values.
2. For each pair of successive examples having different labels, a candidate threshold is placed as the average of the two attribute values.
3. For each candidate threshold the infogain achieved splitting examples according to it is computed.
4. The threshold producing the higher infogain is used to discretize the attribute.

2.3.4 Alternative attribute test measures

The information gain criterion tends to prefer attributes with a large number of possible values. As an extreme the unique ID of each examples is an attribute perfectly splitting the data into singletons, but it will be no use on new examples. A measure of such spread is the entropy of the dataset with respect to the attribute value instead of the class value:

$$H_A(S) = -\sum_{v \in Values(A)} \frac{|S_V|}{|S|} \log_2 \frac{|S_V|}{|S|}$$

The gain ratio measures downweights the information gain by such attribute value entropy:

$$IGR(S, A) = \frac{IG(S, A)}{H_A(S)}$$

2.3.5 Handling attributes with missing values

Assume example x with class $c(x)$ has missing value for attribute A. When attribute A is to be tested at node n:

- Simple solution: Assign to x the most common attribute values among examples in n or the most common of examples in n with class $c(x)$.
- Complex solution: Propagate x to each of the children of n with a fractional value equal to the proportion of examples with the corresponding attribute value.

The complex solution implies that at test time, for each candidate class, all fractions of the test example which reached a leaf with that class are summed, and the example is assigned the class with highest overall value.

2.4 Random forest

Random forests are an ensemble of decision trees. An ensemble is a method by which predictions are given by a set of predictors and is taken the prediction most represented. They improve stability and accuracy of the predictions. Random forests are effective and one of the methods of choice in case of tabular data.

2.4.1 Training

To train a random forest:

1. Given a training set of N examples, sample N examples with replacement.
2. Train a decision tree on the sample, selecting at each node m features at random among which to choose the best one.
3. Repeat the first two step M times in order to generate a forest of M trees.

2.4.2 Testing

To test a random forest:

1. Test the example with each tree in the forest.
2. Return the majority class among the predictions.

Chapter 3

K-nearest neighbours

3.1 Introduction

The K-nearest neighbours is an algorithm that, given a training set represented as a vector of features, gives the label for a new sample as label of the majority of the K nearest sample in the training set.

3.2 Measuring the instance between instances

3.2.1 Metric or distance definition

Given a set $\mathcal{X}$ a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a metric for $\mathcal{X}$ if for any $x, y, z \in \mathcal{X}$ the following properties are satisfied:

- Reflexivity $d(x, y) = 0 \Leftrightarrow x = y$.
- Symmetry $d(x, y) = d(y, x)$.
- Triangle inequality $d(x, y) + d(y, z) \geq d(x, z)$.

3.2.2 Euclidean distance

The euclidean distance in $\mathbb{R}^n$ is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.3 Algorithms

3.3.1 Classification

```
: Knn-Classification()
foreach test examples x do
    foreach training examples (x_i, y_i) do
        compute distance d(x, x_i)
    select k-nearest neighbours of x
    %return class of x as majority class among neighbours
    return arg max_y Σ_{i=1}^{k} δ(y, y_i)
```

Where:

$$\delta(x, y) = \begin{cases} 1 & if\ x = y \\ 0 & otherwise \end{cases}$$

3.3.2 Regression

```
: Knn-Regression()
foreach test examples x do
    foreach training examples (x_i, y_i) do
        compute distance d(x, x_i)
    select k-nearest neighbours of x
    %return the average output value among neighbours
    return (1/k) Σ_{i=1}^{k} y_i
```

3.4 Characteristics

- Instance-based learning: the model used for prediction is calibrated for the test example to be processed.
- Lazy learning: the computation is mostly deferred to the classification phase.
- Local learner: assumes prediction should mainly influenced by nearby instances.
- Uniform feature weighting: all feature are uniformly weighted in computing dinstances.

3.5 Distance weighted k-nearest neighbour

The distance weighted k-nearest neighbour is a variant of the classic k-nearest neighbour in which the distance is weighted. The weight of a point is calculated as:

$$w_i = \frac{1}{d(x, x_i)}$$

The class is decided for classification according to the formula:

$$\arg\max_{x_y} \sum_{i=1}^{k} w_i \delta(y, y_i)$$

For regression the formula is instead:

$$\frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

Chapter 4

Linear algebra

4.1 Vector space

A set $\mathcal{X}$ is called a vector space over $\mathbb{R}$ if addition and scalar multiplication are defined and satisfy for all $\vec{x}, \vec{y}, \vec{z} \in \mathcal{X}$ and $\lambda, \mu \in \mathcal{R}$:

- Addition:
  - Association: $\vec{x} + (\vec{y} + \vec{z}) = (\vec{x} + \vec{y}) + \vec{z}$.
  - Commutation: $\vec{x} + \vec{y} = \vec{y} + \vec{x}$.
  - There is an identity element: $\exists \vec{0} \in \mathcal{X} : \vec{x} + \vec{0} = \vec{x}$.
  - There is an inverse element: $\forall \vec{x} \in \mathcal{X} \exists \vec{x}' \in \mathcal{X} : \vec{x} + \vec{x}' = \vec{0}$.
- Scalar multiplication:
  - Is distributive over elements: $\lambda(\vec{x} + \vec{y}) = \lambda\vec{x} + \lambda\vec{y}$.
  - Is distributive over scalars: $(\lambda + \mu)\vec{x} = \lambda\vec{x} + \mu\vec{x}$.
  - Is associative over scalars: $\lambda(\mu\vec{x}) = (\lambda\mu)\vec{x}$.
  - There is an identity element: $\exists 1 \in \mathbb{R} : 1\vec{x} = \vec{x}$.

4.1.1 Properties and operations

4.1.1.1 Subspace

A subspace is any non-empty subset of $\mathcal{X}$ being itself a vector space.

4.1.1.2 Linear combination

Given $\lambda_i \in \mathbb{R} \wedge \vec{x}_i \in \mathcal{X}$, a linear combination is:

$$\sum_{i=1}^{n} \lambda_i \vec{x}_i$$

4.1.1.3 Span

The span of vectors $\vec{x}_1, \ldots, \vec{x}_n$ is defined as the set of their linear combination:

$$\{\sum_{i=1}^{n} \lambda_i \vec{x}_i, \lambda_i \in \mathbb{R}\}$$

4.1.1.4 Linear independence

A set of vector $\vec{x}_i$ is linearly independent if none of them can be written as a linear combination of the others.

4.1.2 Basis

A set of vectors $\vec{x}_i$ is a basis for $\mathcal{X}$ if any element in $\mathcal{X}$ can be uniquely written as a linear combination of vectors $\vec{x}_i$. The vectors $\vec{x}_i$ need to be linearly independent. All bases of $\mathcal{X}$ have the same number of elements, called the dimension of the vector space.

4.2 Matrices

4.2.1 Linear maps

Given two vector spaces $\mathcal{X}$ and $\mathcal{Z}$ a function $f : \mathcal{X} \to \mathcal{Z}$ is a linear map if $\forall \vec{x}, \vec{y} \in \mathcal{X} \lambda \in \mathbb{R}$:

- $f(\vec{x} + \vec{y}) = f(\vec{x}) + f(\vec{y})$.
- $f(\lambda \vec{x}) = \lambda f(\vec{x})$.

4.2.2 Linear maps as matrices

A linear map between two finite dimensional spaces $\mathcal{X}$ and $\mathcal{Z}$ of dimension n and m can always be written as a matrix. Let $\{\vec{x}_1, \ldots, \vec{x}_n\}$ and $\{\vec{z}_1, \ldots, \vec{z}_m\}$ be some bases for $\mathcal{X}$ and $\mathcal{Z}$ respectively. For any $\vec{x} \in \mathcal{X}$:

$$f(\vec{x}) = f(\sum_{i=1}^{n} \lambda_i \vec{x}_i) = \sum_{i=1}^{n} \lambda_i f(\vec{x}_i)$$
$$f(\vec{x}_i) = \sum_{j=1}^{m} a_{ij} \vec{z}_j$$
$$f(\vec{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_i a_{ji} \vec{z}_j = \sum_{j=1}^{m} (\sum_{i=1}^{n} \lambda_i a_{ji}) \vec{z}_j = \sum_{j=1}^{m} \mu_j \vec{z}_j$$

4.2.2.1 Matrix of basis transformation

A matrix can be used to transform the basis is:

$$M \in \mathbb{R}^{m \times n} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

The mapping from basis to basis coefficient is done:

$$M\vec{\lambda} = \vec{\mu}$$

4.2.2.2 Matrix changing the coordinates, 2D examples

Let $B = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$ be the standard basis in $\mathbb{R}^2$ and $B' = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}$ be an alternative basis. The change of coordinate matrix from B' to B is:

$$P = \begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix}$$

So that:

$$[\vec{v}]_B = P[\vec{v}]_{B'} \qquad \wedge \qquad [\vec{v}]_{B'} = P^{-1}[\vec{v}]_B$$

For arbitrary B and B' P's columns must be the B' vectors written in terms of the B ones.

4.2.3 Matrix properties

4.2.3.1 Transpose

The transpose matrix is the matrix obtained exchanging the rows with column M^T.

$$(MN)^T = N^T M^T$$

4.2.3.2 Trace

The trace is the sum of the diagonal elements of a matrix:

$$tr(M) = \sum_{i=1}^{n} M_{ii}$$

4.2.3.3 Inverse

The inverse is the matrix which multiplied with the original matrix gives the identity:

$$MM^{-1} = I$$

4.2.3.4 Rank

The rank of an $n \times m$ matrix is the dimension of the space spanned by its columns.

4.2.4 Matrix derivatives

$$\begin{aligned}
\frac{\partial M\vec{X}}{\partial \vec{x}} &= M \\
\frac{\partial \vec{y}^T M\vec{x}}{\partial \vec{x}} &= M^T\vec{y} \\
\frac{\partial \vec{x}^T M\vec{x}}{\partial \vec{x}} &= (M^T + M)\vec{x} \\
\frac{\partial \vec{x}^T M\vec{x}}{\partial \vec{x}} &= 2M\vec{x} \qquad \text{if } M \text{ is symmetric} \\
\frac{\partial \vec{x}^T \vec{x}}{\partial \vec{x}} &= 2\vec{x}
\end{aligned}$$

Results are columns vectors. Transposing the matrix gives the row vectors.

4.2.5 Metric structure

4.2.5.1 Norm

A function $||\cdot|| : \mathcal{X} \to \mathbb{R}_0^+$ is a norm $\forall \vec{x}, \vec{y} \in \mathcal{X}, \lambda \in \mathbb{R}$:

- $||\vec{x} + \vec{y}|| \leq ||\vec{x}|| + ||\vec{y}||$
- $||\lambda\vec{x}|| = |\lambda|||\vec{x}||$
- $||\vec{x}|| > 0 \Leftrightarrow \vec{x} \neq \vec{0}$

4.2.5.2 Metric

A norm defines a metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$:

$$d(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||$$

Not every metric gives rise to a norm.

4.2.6 Dot product

A dot product $\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric bilinear form which is positive semi-definite:

$$\langle \vec{x}, \vec{x} \rangle \geq 0 \forall \vec{x} \in \mathcal{X}$$

A positive definite dot product satisfies:

$$\langle \vec{x}, \vec{x} \rangle = 0 \Leftrightarrow \vec{x} = \vec{0}$$

4.2.6.1 Norm

Any dot product defines a corresponding norm via:

$$||\vec{x}|| = \sqrt{\langle \vec{x}, \vec{x} \rangle}$$

4.2.6.2 Properties

4.2.6.2.1 Angle The angle θ between two vectors is defined as:

$$\cos\theta = \frac{\langle \vec{x}, \vec{z} \rangle}{||\vec{x}||||\vec{z}||}$$

4.2.6.2.2 Orthogonal Two vectors are orthogonal if $\langle \vec{x}, \vec{y} \rangle = 0$.

4.2.6.2.3 Orthonormal A set of vectors $\{\vec{x}_1, \ldots, \vec{x}_n\}$ is orthonormal is

$$\langle \vec{x}_i, \vec{x}_j \rangle = \delta_{ij}$$

Where $\delta_{ij} = 1$ if $i = j$, 0 otherwise. If $\vec{x}$ and $\vec{y}$ are n-dimensional column vectors, their dot product is computed as:

$$\langle \vec{x}, \vec{y} \rangle = \vec{x}^T \vec{y} = \sum_{i=1}^{n} \vec{x}_i \vec{y}_i$$

4.3 Eigenvalues and eigenvectors

Given a $n \times n$ matrix M the real value λ and non zero vector $\vec{x}$ are eigenvalue and the corresponding eigenvector of M if:

$$M\vec{x} = \lambda\vec{x}$$

4.3.1 Cardinality

An $n \times n$ matrix has n eigenvalues, but less than n distinct ones. The number of eigenvalues is the number of linear independent eigenvectors.

4.3.2 Singular matrices

A matrix is singular if it has a zero eigenvalue:

$$M\vec{x} = 0\vec{x} = 0$$

A singular matrix has linearly dependent columns.

4.3.3 Symmetric matrices

Eigenvectors corresponding to distinct eigenvalues are orthogonal:

$$\begin{aligned}\lambda\langle\vec{x},\vec{z}\rangle &= \langle A\vec{x},\vec{z}\rangle = \\ &= (A\vec{x})^T\vec{z} = \\ &= \vec{x}^T A^T\vec{z} = \\ &= \vec{x}^T A\vec{z} = \\ &= \langle\vec{x},A\vec{z}\rangle = \\ &= \mu\langle\vec{x},\vec{z}\rangle\end{aligned}$$

4.3.4 Eigen-decomposition

4.3.4.1 Raleigh quotient

$$A\vec{x} = \lambda\vec{x}$$
$$\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}} = \lambda\frac{\vec{x}^T\vec{x}}{\vec{x}^T\vec{x}} = \lambda$$

4.3.4.2 Finding eigenvector

To find the eigenvector you need to maximize the eigenvalue:

$$x = \max_{\vec{v}} \frac{\vec{v}^T A\vec{v}}{\vec{v}^T\vec{v}}$$

After that you need to normalize it so the solution is invariant to rescaling:

$$\vec{x} \leftarrow \frac{\vec{x}}{||\vec{x}||}$$

4.3.4.3 Deflating matrix

$$\bar{A} = A - \lambda\vec{x}\vec{x}^T$$

The deflation turns $\vec{x}$ into a zero eigenvalue eigenvector:

$$\tilde{A}\vec{x} = A\vec{x} - \lambda\vec{x}\vec{x}^T\vec{x}$$
$$A\vec{x} - \lambda\vec{x} = 0$$

Other eigenvalues are unchanged as eigenvectors with distinct eigenvalues are orthogonal because the matrix is symmetric:

$$\tilde{A}\vec{z} = A\vec{z} - \lambda\vec{x}\vec{x}^T\vec{z}$$
$$\tilde{A}\vec{z} = A\vec{z}$$

4.3.4.4 Iterating

The maximization procedure is repeated on the deflated matrix until solution is zero. Minimization is iterated to get eigenvectors with negative eigenvalues. Eigenvectors with zero eigenvalues are obtained extending the obtained set to an orthonormal basis.

4.3.4.5 Eigen-decomposition

Let $V = [\vec{v}_1 \dots \vec{v}_n]$ be a matrix with orthonormal eigenvectors as columns. Let Λ be the diagonal matrix of corresponding eigenvalues. A square symmetric matrix can be diagonalizes as:

$$V^T A V = \Lambda$$

A diagonalized matrix is simpler to manage and has the same properties as the original one.

4.3.4.5.1 Proof

$$A[\vec{v}_1 \dots \vec{v}_n] = [\vec{v}_1 \dots \vec{v}_n] \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}$$

$$AV = V\Lambda$$

$$V^{-1}AV = V^{-1}V\Lambda$$

$$V^T A V = \Lambda$$

V is a unitary matrix with orthonormal columns for which $V^{-1} = V^T$.

4.3.4.6 Positive semi-definite matrix

An $n \times n$ symmetric matrix M is positive semi-definite if all its eigenvalues are non-negative. Positive semi-definite:

- $\Leftrightarrow \forall \vec{x} \in \mathbb{R}^n : \vec{x}^T M \vec{x} \geq \vec{0}$
- $\Leftrightarrow \exists B : M = B^T B$

4.3.4.7 Scaling transformation in standard basis

Let $\vec{x}_1 = [1, 0]$ and $\vec{x}_2 = [0, 1]$ the standard orthonormal basis in $\mathbb{R}^2$. Let $\vec{x} = [x_1, x_2]$ am arbitrary vector in $\mathbb{R}^2$. A linear transformation is a scaling transformation if it only stretches $\vec{x}$ along its directions.

4.3.4.8 Scaling transformation in eigenbasis

Let A be a non-scaling transformation in $\mathbb{R}$. Let $\{\vec{v}_1, \vec{v}_2\}$ be an eigenbasis for A. By representing vectors in $\mathbb{R}^2$ in terms of the $\{\vec{v}_1, \vec{v}_2\}$ basis A becomes a scaling transformation.

4.4 Principal component analysis

Principal component analysis or PCA is a non-supervised machine learning technique that accomplishes dimensionality reduction. Let X be a data matrix with correlated coordinates. PCA is a linear transformation mapping data to a system of uncorrelated coordinates. It corresponds to fitting an ellipsoid to the data, whose axis are the coordinates of the new space.

4.4.1 Procedure

Given a dataset $X \in \mathbb{R}^{n \times d}$ in d dimension: Compute the mean of the data, where X_i are the row vectors of X:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Center the data into the origin:

$$X - \begin{bmatrix} \bar{x} \\ \vdots \\ \bar{x} \end{bmatrix}$$

Compute the data covariance:

$$C = \frac{1}{n} X^T X$$

Compute the orthonormal eigen-decomposition of C:

$$V^T C V = \Lambda$$

Us it as the new coordinate system:

$$\vec{x}' = V^{-1} \vec{x} = V^T \vec{x}$$

Where $V^1 = V^T$ as V is unitary. This method assumes linear correlation and Gaussian distribution.

4.4.2 Dimensionality reduction

Each eigenvalue corresponds to the amount of variance in that direction. Select only the k eigenvalues with largest eigenvalue for dimensionality reduction:

$$W = [\vec{v}_1, \ldots, \vec{v}_k]$$

$$\vec{x}' = W^T \vec{x}$$

Chapter 5

Probability theory

5.1 Discrete random variables

5.1.1 Probability mass function

Given a discrete random variable X taking values in $\mathcal{X} = \{v_1, \ldots, v_m\}$ its probability mass function $P : \mathcal{X} \to [0, 1]$ is defined as:

$$P(v_i) = Pr[X = v_i]$$

This function satisfies:

- $P(x) \geq 0$
- $\sum_{x \in \mathcal{X}} = 1$

5.1.2 Expected value

The expected value, mean or average of a random variable x is:

$$\mathbb{E}[x] = \mu = \sum_{x \in \mathcal{X}} xP(x) = \sum_{i=1}^{m} v_i P(v_i)$$

The expectation operator is linear:

$$\mathbb{E}[\lambda x + \lambda' y] = \lambda \mathbb{E}[x] + \lambda' \mathbb{E}[y]$$

5.1.3 Variance

The variance of a random variable is the moment of inertia of its probability mass function:

$$Var[x] = \sigma^2 = \mathbb{E}[(x - \mu)^2] = \sum_{x \in \mathcal{X}} (x - \mu)^2 P(x)$$

The standard deviation σ indicates the typical amount of deviation from the mean one should expect for a randomly drawn value for x.

5.1.4 Properties of mean and variance

5.1.4.1 Second moment

$$\mathbb{E}[x^2] = \sum_{x \in \mathcal{X}} x^2 P(x)$$

5.1.4.2 Variance in term of expectation

$$Var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$$

5.1.4.3 Variance and scalar multiplication

$$Var[\lambda x] = \lambda^2 Var[x]$$

5.1.4.4 Variance of uncorrelated variables

$$Var[x + y] = Var[x] + Var[y]$$

5.1.5 Probability distributions

5.1.5.1 Bernoulli distribution

The Bernoulli distribution indicates a variable for which there are two values: 1 for success and 0 for failure. Its parameter is p the probability of success. Its probability mass function:

$$P(x; p) = \begin{cases} p & if\ x = 1 \\ 1 - p & if\ x = 0 \end{cases}$$

- $\mathbb{E}[x] = p$
- $Var[x] = p(1 - p)$

5.1.5.1.1 Proof of mean

$$\begin{aligned} \mathbb{E}[x] &= \sum_{x \in \mathcal{X}} xP(x) \\ &= \sum_{x \in \{0;1\}} xP(x) \\ &= 0 \cdot (1 - p) + 1 \cdot p = p \end{aligned}$$

5.1.5.1.2 Proof of variance

$$\begin{aligned} Var[x] &= \sum_{x \in \mathcal{X}} (x - \mu)^2 P(x) \\ &= \sum_{x \in \{0;1\}} (x - \mu)^2 P(x) \\ &= (0 - p)^2 (1 - p) + (1 - p)^2 p \\ &= p^2 (1 - p) + (1 - p)(1 - p)p \\ &= (1 - p)(p^2 + p - p^2) \\ &= (1 - p)p \end{aligned}$$

5.1.5.2 Binomial distribution

The binomial distribution is the probability of a certain number of successes in n independent Bernoulli trials. Its parameters are p the probability of success and n the number of trials. Its probability mass function:

$$P(x; p, n) = \binom{n}{x} p^x (1 - p)^{n - x}$$

- $\mathbb{E}[x] = np$
- $Var[x] = np(1-p)$

5.1.6 Pairs of discrete random variables

5.1.6.1 Probability mass function

Given a pair of discrete random variables X and Y taking values $\mathcal{X} = \{v_1, \ldots, v_m\}$ and $\mathcal{Y} = \{w_1, \ldots, w_n\}$ the joint probability mass function is defined as:

$$P(v_i, w_j) = Pr[X = v_i, Y = w_j]$$

This satisfies:

- $P(x, y) \geq 0$
- $\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) = 1$

5.1.6.2 Properties

5.1.6.2.1 Expected value

$$\mu_x = \mathbb{E}[x] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} xP(x, y)$$

$$\mu_y = \mathbb{E}[y] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} yP(x, y)$$

5.1.6.2.2 Variance

$$\sigma_x^2 = Var[(x - \mu_x)^2] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (x - \mu_x)^2 P(x, y)$$

$$\sigma_y^2 = Var[(y - \mu_y)^2] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (y - \mu_y)^2 P(x, y)$$

5.1.6.2.3 Covariance

$$\sigma_{xy} = \mathbb{E}[(x - \mu_x)(y - \mu_y)] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (x - \mu_x)(y - \mu_y) P(x, y)$$

5.1.6.2.4 Correlation coefficient

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

5.1.6.3 Multinomial distribution

Given n samples of an event with m possible outcomes, the multinomial distribution models the probability of a certain distribution of outcomes. It has parameters $p_1, \ldots, p_m$ probability of each outcome and n the number of samples. Its probability mass function assumes $\sum_{i=1}^{m} x_i = n$ and it is:

$$P(x_1, \ldots, x_m; p_1, \ldots, p_m, n) = \frac{n!}{\prod_{i=1}^{m} x_i!} \prod_{i=1}^{m} p_i^{x_i}$$

- $\mathbb{E}[x_i] = np_i$
- $Var[x_i] = np_i(1 - p_i)$
- $Cov[x_i, x_j] = -np_ip_j$

5.2 Conditionally probability

The conditional probability is the probability of x once y is observed:

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

Variables X and Y are statistical independent if and only if:

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

This implies:

- $P(x|y) = P(x)$
- $P(y|x) = P(y)$

5.2.1 Basic rules

5.2.1.1 Law of total probability

The marginal distribution of a variable is obtained from a joint distribution summing over all possible values of the other variable:

- $P(x) = \sum_{y \in \mathcal{Y}} P(x,y)$
- $P(y) = \sum_{x \in \mathcal{X}} P(x,y)$

5.2.1.2 Product rule

Conditional probability definition implies that:

$$P(x,y) = P(x|y)P(y) = P(y|x)P(x)$$

5.2.2 Bayes' rule

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

This allows to invert statistical connection between effects x and cause y:

$$posterior = \frac{likelihood \times prior}{evidence}$$

The evidence can be obtained using the sum rule from likelihood and prior:

$$P(x) = \sum_y P(x,y) = \sum_y P(x|y)P(y)$$

5.3 Continuous random variables

5.3.1 Cumulative distribution function

To generalize the probability mass function to continuous domains it is considered the probability of intervals:

$$W = (a < X \leq b) \qquad A = (X \leq a) \qquad B = (X \leq b)$$

W and A are mutually exclusive, so:

$$P(B) = P(A) + P(W) \qquad P(W) = P(B) - P(A)$$

$F(q) = P(X \leq q)$ is the cumulative distribution function of X a monotonic function such that the probability of an interval is the difference:

$$P(a < X \leq b) = F(b) - F(a)$$

5.3.2 Probability density function

The derivative of the cumulative distribution function:

$$p(x) = \frac{d}{dx}F(x) \qquad F(q) = P(X \leq q) = \int_{-\infty}^{q} p(x)dx$$

So that it respect the properties:

- $p(x) \geq 0$
- $\int_{-\infty}^{\infty} p(x)dx = 1$

The probability density function of a value x can be greater than one provided the integral is one.

- $\mathbb{E}[x] = \mu = \int_{-\infty}^{\infty} xp(x)dx$
- $Var[x] = \sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx$

Definitions and formulae for discrete random variables carry over to continuous random variables with sums replaced by integrals.

5.3.3 Probability distribution

5.3.3.1 Gaussian or normal distribution

The normal distribution is a bell-shaped curved with parameters μ mean and σ^2 variance. Its probability density function is:

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

Where:

- $\mathbb{E}[x] = \mu$
- $Var[x] = \sigma^2$

The standard normal distribution is $N(0, 1)$. Every normal distribution can be transformed in a standard one:

$$z = \frac{x - \mu}{\sigma}$$

5.3.3.2 Beta distribution

The beta distribution is defined in the interval $[0, 1]$ with parameters α and β. Its probability density function is:

$$p(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1 - x)^{\beta-1}$$

Where:

- $\mathbb{E}[x] = \frac{\alpha}{\alpha+\beta}$
- $\Gamma(x + 1) = x\Gamma(x)$
- $Var[x] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$
- $\Gamma(1) = 1$

It models the posterior distribution of parameter p of a binomial distribution after observing $\alpha - 1$ independent events with probability p and $\beta - 1$ with probability $1 - p$.

5.3.3.3 Multivariate normal distribution

The multivariate normal distribution is the normal distribution for d-dimensional vectorial data. Its parameter are $\vec{\mu}$ mean vector and Σ covariance matrix. Its probability density function:

$$p(\vec{x}, \vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})}$$

Where:

- $\mathbb{E}[\vec{x}] = \vec{\mu}$
- $Var[\vec{x}] = \Sigma$

The standard measure of instance to mean is the squared Mahalanobis distance:

$$r^2 = (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})$$

5.3.3.4 Dirichlet distribution

The Dirichlet distribution is defines in $x \in [0, 1]^m$, $\sum_{i=1}^{m} x_i = 1$ It has parameters $\vec{\alpha} = \alpha_1, \ldots, \alpha_m$. Its probability density function is:

$$p(x_1, \ldots, x_m; \vec{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{m} \Gamma(\alpha_i)} \prod_{i=1}^{m} x_i^{a_i - 1}$$

Where $\alpha_0 = \sum_{j=1}^{m} \alpha_j$. Also:

- $\mathbb{E}[x_i] = \frac{\alpha_i}{\alpha_0}$
- $Var[x_i] = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0+1)}$
- $Cov[x_i, x_j] = \frac{-\alpha_i \alpha_j}{a_0^2(\alpha_0+1)}$

This distribution models the posterior distribution of parameters p of a multinomial distribution after observing $\alpha_i - 1$ times each mutually exclusive event.

5.4 Probability laws

5.4.1 Expectation of an average

Consider a sample of $X_1, \ldots, X_n$ instances drawn from a distribution with mean μ and variance σ^2. Consider the random variable $\bar{X}_n$ measuring the sample average:

$$\bar{X}_n = \frac{X_1 + \cdots + X_n}{n}$$

Considering that $\mathbb{E}[a(X + Y)] = a(\mathbb{E}[x] + \mathbb{E}[y])$:

$$\mathbb{E}[\bar{X}_n] = \frac{1}{n}(\mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n]) = p$$

Or that the expectation of an average is the true mean of the distribution.

5.4.2 Variance of an average

Consider the random variable $\bar{X}_n$ measuring the sample average. Its variance is computed as $Var[a(X + Y)] = a^2(Var[X] + Var[Y])$ if X and Y are independent:

$$Var[\bar{X}_n] = \frac{1}{n^2}(Var[X_1] + \cdots Var[X_n]) = \frac{\sigma^2}{n}$$

The variance of an average decreases with the number of observations: the more examples are seen, the more likely it is to estimate the correct average.

5.4.3 Chebyshev's inequality

Consider a random variable X with mean μ and variance σ^2. For all $a > 0$:

$$Pr[|X - \mu| \geq a] \leq \frac{\sigma^2}{a^2}$$

Considering $a = k\sigma$, for $k > 0$:

$$Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

Most of the probability mass of a random variable stays within few standard deviations from its mean.

5.4.4 Law of large numbers

Consider a sample $X_1, \ldots, X_n$ instances drawn from a distribution with mean μ and variance σ^2. For any $\varepsilon > 0$ its sample average $\bar{X}_n$ obeys:

$$\lim_{n\to\infty} Pr[|\bar{X}_n - \mu| > \varepsilon] = 0$$

It can be shown using Chebyshev's inequality and the facts that $\mathbb{E}[\bar{X}_n] = \mu$ and $Var[\bar{X}_n] = \frac{\sigma^2}{n}$:

$$Pr[|\bar{X}_n - \mathbb{E}[\bar{X}_n]| \geq \varepsilon] \leq \frac{\sigma^2}{n\varepsilon^2}$$

This tells that the accuracy of an empirical statistic increases with the number of samples.

5.4.5 Central limit theorem

Consider a sample of $X_1, \ldots, X_n$ instances drawn from a distribution with mean μ and variance σ^2. Regardless of the distribution of X, for $n \to \infty$ the distribution of the sample average $\bar{X}_n$ approaches a normal distribution. Its mean approaches μ and its variance $\frac{\sigma^2}{n}$. Thus the normalized sample average:

$$z = \frac{\bar{X}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Approaches a normal distribution $N(0, 1)$.

5.4.5.1 Interpretation

The sum of a sufficiently large sample of random measurements is approximately normally distributed. The form of their distribution can be arbitrary. This justifies the importance of the Normal distribution in real world applications.

5.5 Information theory

5.5.1 Entropy

Consider a discrete set of symbols $\mathcal{V} = \{v_1, \ldots, v_n\}$ with mutually exclusive probabilities. To design a binary code for each symbol minimizing the average length of messages, Shannon and Weaver proved that the optimal code assigns to each symbol a number of bits equal to $-\log P(v_i)$. The entropy of the set of symbols is the expected length of a message encoding a symbol assuming such optimal coding:

$$H[\mathcal{V}] = \mathbb{E}[-\log P(v)] = -\sum_{i=1}^{n} P(v_i) \log P(v_i)$$

5.5.2 Cross entropy

Consider two distributions P and Q over variable X. The cross entropy between P and Q measures the expected number of bits needed to code a symbol sampled from P using Q instead:

$$H(P,Q) = \mathbb{E}_P[-\log Q(v)] = -\sum_{i=1}^{n} P(v_i) \log Q(v_i)$$

It is often used as a loss for binary classification with P true distribution and Q the predicted one.

5.5.3 Relative entropy

Consider two distributions P and Q over variable X. The relative entropy or Kullback-Leibler divergence measures the expected length difference when coding instances sampled from P using Q instead.

$$\begin{aligned} D_{KL}(p||q) &= H(P,Q) - H(P) = \\ &= -\sum_{i=1}^{n} P(v_i) \log Q(v_i) + \sum_{i=1}^{n} P(v_i) \log P(v_i) = \\ &= \sum_{i=1}^{n} P(v_i) \log \frac{P(v_i)}{Q(v_i)} \end{aligned}$$

The KL-divergence is not a distance as it is not symmetric.

5.5.4 Conditional entropy

Consider two variables V and W with possibly different distributions P. The conditional entropy is the entropy remaining for variable W once V is known:

$$\begin{aligned} H(W|V) &= \sum_{v} P(v) H(W|V=v) = \\ &= -\sum_{v} P(v) \sum_{w} P(w|v) \log P(w|v) \end{aligned}$$

5.5.5 Mutual information

Consider two variables V and W with possibly different distributions P. The mutual information or information gain is the reduction in entropy for W once V is known:

$$\begin{aligned} I(W,V) &= H(W) - H(W|V) \\ &= -\sum_{w} p(w) \log p(w) + \sum_{v} P(v) \sum_{w} P(w|v) \log P(w|v) \end{aligned}$$

It is used in selecting the best attribute to use in building a decision tree, where V is the attribute and W is the label.

Chapter 6

Evaluation

6.1 Introduction

Evaluation requires to define the performance measures to be optimized. Performance of learning algorithms cannot be evaluated on the entire domain because of the generalization error, so there is a need for approximation. Performance evaluation is needed for:

- Tuning the hyperparameters of the learning method.
- Evaluating the quality of the learned predictor.
- Computing statistical significance of difference between different learning algorithms.

6.2 Performance measures

6.2.1 Training loss and performance measures

The training loss function measures the cost paid for predicting $f(x)$ for output y. It is designed to boost effectiveness and efficiency of learning algorithm. It is not necessarily the best measure of final performance, for example misclassification cost is never used as it is a piecewise constant no amenable to gradient descent. Multiple performance measures could be used to evaluate different aspects of a learner.

6.2.2 Binary classification

For binary classification the confusion matrix reports the effective label on the rows and the predicted one on the columns. Each entry contains the number of examples having label in row and predicted as column.

- TP true positives: positives predicted as positive.
- FP false positives: negative predicted as positive.
- TN true negative: negative predicted as negative.
- FN false negatives: positives predicted as negatives.

6.2.2.1 Accuracy

Accuracy is the fraction of correctly labelled examples among all predictions. It is one minus the misclassification cost:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

For strongly unbalanced datasets it is not informative because predictions are dominated by the larger class and predicting everything as negative maximizes accuracy. One way of resolving this problem consists of introducing the rebalancing cost: a single positive counts as $\frac{N}{P}$ where $N = TN + FP$ and $P = TP + FN$.

6.2.2.2 Precision

Precision is the fraction of positives among examples predicted as positives. It measures the precision of the learner when predicting positive:

$$Pre = \frac{TP}{TP + FP}$$

6.2.2.3 Recall or sensitivity

Recall is the fraction of positive examples predicted as positives. It measures the coverage of the learner in returning positive examples:

$$Rec = \frac{TP}{TP + FN}$$

6.2.2.4 F-measure

Precision and recall are complementary: increasing prediction typically reduces recall. F-measures combines the two measures balancing the two aspects with a parameter β that defines the trade-off between precision and recall:

$$F_\beta = \frac{(1 + \beta^2)(Pre \cdot Rec)}{\beta^2 Pre + Rec}$$

If $\beta = 1$ the measure is called F_1 and it is the harmonic mean of precision and recall:

$$F_1 = \frac{2(Pre \cdot Rec)}{Pre + Rec}$$

6.2.2.5 Precision-recall curve

Classifiers often provide a confidence in the prediction and a hard decision is made setting a threshold on the classifier. Accuracy, precision, recall and F_1 all measures performance of a classifier for a specific threshold. It is possible to change the threshold if the interest is in maximizing a specific performance. By varying the threshold from minimum to maximum possible values we obtain a curve of performance measures. This curve can be shown plotting one measure like recall against the complementary one like precision. It is possible to investigate performance of the learner in different scenarios. The area under this curve can be used as a single aggregate value that combines performance of the algorithm for all possible threshold.

6.2.3 Multiclass classification

For multiclass classification the confusion matrix is a generalized evasion of the binary one such that n_{ij} is the number of examples with class y_i predicted as y_j. The main diagonal contains true positives for each class. The sum of off-diagonal elements along a column is the number of false positive for the column label and the sum of off-diagonal elements along a row is the number of false negatives for the raw label:

- $FP_i = \sum_{j \neq i} n_{ji}$
- $FN_i = \sum_{j \neq i} n_{ij}$

6.2.3.1 Muliclass accuracy

Accuracy, precision, recall and F_1 carry over to a per-class measure considering as negative examples from other classes:

- $Pre_i = \frac{n_{ii}}{n_{ii}+FP_i}$
- $Rec_i = \frac{n_{ii}}{n_{ii}+FN_i}$

Multiclass accuracy is the overall fraction of correctly classified examples:

$$MAcc = \frac{\sum_i n_{ii}}{\sum_i \sum_j n_{ij}}$$

6.2.4 Regression

6.2.4.1 Root mean squared error

The root mean squared error for dataset $\mathcal{D}$ with $n = |\mathcal{D}|$ is:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(\vec{x}_i) - y_i)^2}$$

6.2.4.2 Pearson correlation coefficient

The Pearson correlation coefficient for random variable X and Y is:

$$\rho = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \bar{X})(Y - \bar{Y}]}{\sqrt{\mathbb{E}[(X - \bar{X})^2]\mathbb{E}[(Y - \bar{Y})^2]}}$$

For regression on $\mathcal{D}$ it becomes:

$$\rho = \frac{\sum_{i=1}^{n}(f(\vec{x}_i) - \bar{f}(\vec{x}_i))(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^{n}(f(\vec{x}_i) - \bar{f}(\vec{x}_i))^2 \sum_{i=1}^{n}(y_i - \bar{y}_i)^2}}$$

Where $\bar{z}$ is the average of z on $\mathcal{D}$.

6.2.4.3 Hold-out procedure

Computing the performance measure on training set would be optimistically biased, so there is a need to retain an independent set on which to compute performance:

- Validation set: when used to estimate performance of different algorithmic settings.
- Test set: when used to estimate final performance of selected model.

Hold-out procedure depends on the specific test and validation set chosen.

6.2.4.4 K-fold cross validation

In the k-fold cross validation $\mathcal{D}$ is split in k equal sized disjoint subsets $\mathcal{D}_i$. Then for $i \in [1, k]$:

1. Train predictor on $\mathcal{T} = \mathcal{D} \backslash \mathcal{D}_i$.
2. Compute score S of predictor $L(\mathcal{T}_i)$ on test set $\mathcal{D}_i$: $S_i = S_{\mathcal{D}_i}[L(\mathcal{T}_i)]$.

Then the average score across folds is returned:

$$S = \frac{1}{k} \sum_{i=1}^{k} S_i$$

6.2.4.4.1 Variance The variance of the average score is computed as:

$$Var[\bar{S}] = Var\left[\frac{S_1 + \cdots + S_k}{k}\right] = \frac{1}{k^2} \sum_{j=1}^{k} Var[S_j]$$

$Var[S_j]$ cannot be exactly computed, so it is approximated with the unbiased variance across folds"

$$Var[S_j] = Var[S_h] \approx \frac{1}{k-1} \sum_{i=1}^{k} (S_i - \bar{S})^2$$

Giving:

$$Var[\bar{S}] \approx \frac{1}{k^2} \sum_{i=1}^{k} (S_i - \bar{S})^2 = \frac{1}{k(k-1)} \sum_{i=1}^{k} (S_i - \bar{S})^2$$

6.3 Hypothesis testing

There is a need to compare generalization performance of two learning algorithms. There is a need to know whether observed difference in performance is statistically significant. Hypothesis testing allows to test the statistical significance of an hypothesis.

6.3.1 Test statistic

Given a null hypothesis H_0, the default hypothesis, for rejecting which evidence should be provided, a sample of k realizations of random variables $X_1, \ldots, X_k$, a test statistic is a statistic $T = h(X_1, \ldots, X_n)$ whose value is used to decide whether to reject H_0 or not.

6.3.2 Glossary

- **Tail probability**: probability that T is at least as great (right tail) or at least as small (left tail) as the observed value t.
- **p-value**: probability of obtaining a value T at least as extreme as the observed t in case H_0 is true.
- **Type I error**: reject the null hypothesis when it's true.
- **Type II error**: accept the null hypothesis when it's false.
- **Significance level**: larges acceptable probability for committing a type I error.
- **Critical region**: set of values of T for which the null hypothesis is rejected.
- **Critical values**: values on the boundary of the critical region.

6.3.3 T-test

In the t-test the test statistics is given by the standardized or studentized mean:

$$T = \frac{\bar{X} - \mu_0}{\sqrt{\tilde{Var}[\bar{X}]}}$$

Where $\tilde{Var}[\bar{X}]$ is the approximated variance using the unbiased sample one. Assuming the samples come from an unknown normal distribution, the test statistics has a t_{k-1} distribution under the null hypothesis. The null hypothesis can be rejected at significance level α if:

$$T \leq -t_{k-1,\frac{\alpha}{2}} \qquad \vee \qquad T \geq t_{k-1,\frac{\alpha}{2}}$$

6.3.3.1 t_{k-1} distribution

The t_{k-1} distribution is a bell-shaped distribution similar to the Normal one. It is wider and shorter, reflecting greater variance due to using $\tilde{Var}[\bar{X}]$ instead of the true unknown variance of the distribution. $k-1$ is the number of degree of freedom of the distribution and it's related to the number of independent events observed. t_{k-1} tends to the standardized normal z for $k \to \infty$.

6.3.4 Comparing learning algorithms

6.3.4.1 Hypothesis testing

After having run k-fold cross validation procedure for algorithms A and B, the mean performance difference for them is computed:

$$\bar{\delta} = \frac{1}{k}\sum_{i=1}^{k}\delta_i = \frac{1}{k}\sum_{i=1}^{k} S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)] - S_{\mathcal{D}_i}[L_B(\mathcal{T}_i)]$$

The null hypothesis is that the mean difference is zero.

6.3.4.2 T-test

At significance level α:

$$\frac{\bar{\delta}}{\sqrt{\tilde{Var}[\bar{\delta}]}} \leq -t_{k-1,\frac{\alpha}{2}} \qquad \vee \qquad \frac{\bar{\delta}}{\sqrt{\tilde{var}[\bar{\delta}]}} \geq t_{k-1,\frac{\alpha}{2}}$$

Where:

$$\sqrt{\tilde{Var}[\bar{\delta}]} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$$

6.3.4.3 Notes

- If the two hypothesis where evaluated over identical samples the paired test is used.
- If no prior knowledge can tell the direction of difference the two-tailed test is used, otherwise the one-tailed test.

Chapter 7

Bayesian decision theory

7.1 Introduction

Bayesian decision theory allows to take optimal decisions in a fully probabilistic setting. It assumes all relevant probabilities are known, allowing to provide upper bounds on achievable errors and to evaluate classifiers accordingly. Bayesian reasoning can be generalized to cases when the probabilistic structure is not entirely known.

7.1.1 Input-output pairs

In binary classification assume that examples $(x, y) \in \mathcal{X} \times \{-1, 1\}$ are drawn from a known distribution $p(x, y)$. The task is predicting the class y of examples given x. This can be written in probabilistic terms using Bayes rule:

$$P(y|x) = \frac{p(x|y)P(y)}{p(x)}$$

7.1.1.1 Output given input

Bayes rule allows to compute the posterior probability given likelihood, prior and evidence:

$$posterior = \frac{likelihood \times prior}{evidence}$$

- The posterior $P(y|x)$ is the probability that class is y given that x was observed.
- The likelihood $p(x|y)$ is the probability of observing x given that its class is y.
- The prior $P(y)$ is the prior probability of the class without any evidence.
- The evidence $p(x)$ is the probability of the observation and following the law of total probability can be computed as:

$$p(x) = \sum_{i=1}^{2} p(x|y)P(y)$$

7.1.2 Expected error

The probability of error given x is:

$$P(error|x) = \begin{cases} P(y_2|x) & if\ y_1\ is\ chosen \\ P(y_1|x) & if\ y_2\ is\ chosen \end{cases}$$

The average probability of error is:

$$P(error) = \int_{-\infty}^{\infty} P(error|x)p(x)dx$$

7.1.3 Bayes decision rule

7.1.3.1 Binary case

$$y_B = \arg \max_{y_i \in \{-1,1\}} P(y_i|x) = \arg \max_{y_i \in \{-1,1\}} p(x|y_i)P(y_i)$$

7.1.3.2 Multiclass case

$$y_B = \arg \max_{y_i \in \{1,\ldots,c\}} P(y_i|x) = \arg \max_{y_i \in \{1,\ldots,c\}} p(x|y_i)P(y_i)$$

7.1.3.3 Optimal rule

The probability of error given x is:

$$P(error|x) = 1 - P(y_B|x)$$

The Bayes decision rule minimizes the probability of error.

7.2 Representing classifiers

7.2.1 Discriminant functions

A classifier can be represented as a set of discriminant functions $g_i(\vec{x}), i \in 1, \ldots, c$, giving:

$$y = \arg \max_{i \in 1,\ldots,c} g_i(\vec{x})$$

A discriminant function is not unique and the most convenient one can be chosen for computational or explanatory reasons:

- $g_i(\vec{x}) = P(y_i|\vec{x}) = \frac{p(\vec{x}|y_i)P(y_i)}{p(\vec{x}x)}$.
- $g_i(\vec{x}) = p(\vec{x}|y_i)P(y_i)$.
- $g_i(\vec{x}) = \ln p(\vec{x}|y_i) + \ln P(y_i)$.

7.2.2 Decision regions

The feature space is divided into decision regions $\mathcal{R}_1, \ldots, \mathcal{R}_x$ such that:

$$\vec{x} \in \mathcal{R}_i \qquad if\ g_i(\vec{x}) > g_j(\vec{x}) \forall j \neq i$$

This regions are separated by decision boundaries, regions in which ties occur among the largest discriminant functions.

7.3 Multivariate normal density

The multivariate normal density is the function

$$d(\vec{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1}(\vec{x}-\vec{\mu})}$$

The covariance matrix Σ is always symmetric and positive semi-definite and strictly positive definite is the dimension of the feature space is d.

7.3.1 Hyperellipsoids

The loci of constant density are hyperellipsoids of constant Mahalanobis distance from $\vec{x}$ to $\vec{\mu}$. The principal axes of such hyperellipsoids are the eigenvectors of Σ, their lengths are given by the corresponding eigenvalues.

7.3.2 Discriminant functions for normal density

Let the discriminant function be:

$$\begin{aligned} g_i(\vec{x}) &= \ln p(\vec{x}|y_i) + \ln P(y_i) = \\ &= -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^t \Sigma_i^{-1}(\vec{x}-\vec{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| + \ln P(y_i) \end{aligned}$$

Discarding those terms which are independent of i:

$$g_i(x) = -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^t \Sigma_i^{-1}(\vec{x}-\vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln P(y_i)$$

7.3.2.1 Case $\Sigma_i = \sigma^2 I$

In the case $\Sigma_i = \sigma^2 I$ all features are statistically independent and all of them have the same variance σ^2. The covariance determinant $|\Sigma_i| = \sigma^{2d}$ can be ignored as being independent of i. The covariance inverse is given by $\Sigma_i^{-1} = \frac{1}{\sigma^2} I$. The discriminant function then becomes:

$$g_i(\vec{x}) = -\frac{||\vec{x}-\vec{\mu}_i||^2}{2\sigma^2} + \ln P(y_i)$$

Then, expanding the quadratic form:

$$g_i(\vec{x}) = -\frac{1}{2\sigma^2}[\vec{x}^t\vec{x} - 2\vec{\mu}_i^t\vec{x} + \vec{\mu}_i^t\vec{\mu}_i] + \ln P(y_i)$$

Discarding all the terms independent of i the linear discriminant function is obtained:

$$g_i(x) = \underbrace{\frac{1}{\sigma^2}\vec{\mu}_i^t}_{\vec{w}_i^t}\vec{x} \underbrace{- \frac{1}{2\sigma^2}\vec{\mu}_i^t\vec{\mu}_i + \ln P(y_i)}_{w_{i0}}$$

7.3.2.1.1 Separating hyperplane Setting $g_i(\vec{x}) = g_j(\vec{x})$ the decision boundaries are pieces of hyperplanes.

$$\underbrace{(\vec{\mu}_i - \vec{\mu}_j)^t}_{\vec{w}^t}\left(\vec{x} - \underbrace{\left(\frac{1}{2}(\vec{\mu}_i + \vec{\mu}_j) - \frac{\sigma^2}{||\vec{\mu}_i - \vec{\mu}_j||^2}\ln\frac{P(y_i)}{P(y_j)}(\vec{\mu}_i - \vec{\mu}_j)\right)}_{\vec{x}_0}\right)$$

This hyperplane is orthogonal to vector $\vec{w}$, the line linking the means and passes through $\vec{x}_0$: if the prior probabilities of the classes are equal, $\vec{x}_o$ is halfway between the means, otherwise it shifts away from the more likely mean.

7.3.2.1.1.1 Derivation of the separating hyperplane

$$\begin{aligned}
g_i(\vec{x}) - g_j(\vec{x}) &= 0 \\
\frac{1}{\sigma^2}\vec{\mu}_i^t\vec{x} - \frac{1}{2\sigma^2}\vec{\mu}_i^t\vec{\mu}_i + \ln P(y_i) - \frac{1}{\sigma^2}\vec{\mu}_j^t\vec{x} + \frac{1}{2\sigma^2}\vec{\mu}_j^t\vec{\mu}_j - \ln P(y_j) &= 0 \\
(\vec{\mu}_i - \vec{\mu}_j)^t\vec{x} - \frac{1}{2}(\vec{\mu}_i^t\vec{\mu}_i - \vec{\mu}_j^t\vec{\mu}_j) + \sigma^2\ln\frac{P(y_i)}{P(y_j)} &= 0 \\
\vec{w}^t(\vec{x} - \vec{x}_o) &= 0 \\
\vec{w} &= \vec{\mu}_i - \vec{\mu}_j \\
(\vec{\mu}_i - \vec{\mu}_j)^t\vec{x}_0 &= \frac{1}{2}(\vec{\mu}_i^t\vec{\mu}_i - \vec{\mu}_j^t\vec{\mu}_j) - \sigma^2\ln\frac{P(y_i)}{P(y_j)} \\
\vec{\mu}_i^t\vec{\mu}_i - \vec{\mu}_j^t\vec{\mu}_j &= (\vec{\mu}_i - \vec{\mu}_j)^t(\vec{\mu}_i + \vec{\mu}_j) \\
\ln\frac{P(y_i)}{P(y_j)} = \frac{(\vec{\mu}_i - \vec{\mu}_j)^t(\vec{\mu}_i - \vec{\mu}_j)}{(\vec{\mu}_i - \vec{\mu}_j)^t(\vec{\mu}_i - \vec{\mu}_j)}\ln\frac{P(y_i)}{P(y_j)} &= (\vec{\mu}_i - \vec{\mu}_j)^t\frac{\vec{\mu}_i - \vec{\mu}_j}{||\vec{\mu}_i - \vec{\mu}_j||^2}\ln\frac{P(y_i)}{P(y_j)} \\
\vec{x}_0 &= \frac{1}{2}(\vec{\mu}_i + \vec{\mu}_j) - \sigma^2\frac{\vec{\mu}_i - \vec{\mu}_j}{||\vec{\mu}_i - \vec{\mu}_j||^2}\ln\frac{P(y_i)}{P(y_j)}
\end{aligned}$$

7.3.2.2 Case $\Sigma_i = \Sigma$

In the case where all classes have the same covariance matrix the discriminant function become:

$$g_i(\vec{x}) = -\frac{1}{2}(\vec{x} - \vec{\mu}_i)^t\Sigma^{-1}(\vec{x} - \vec{\mu}_i) + \ln P(y_i)$$

Expanding the quadratic form and discarding terms independent of i we obtain the linear discriminant function:

$$g_i(\vec{x}) = \underbrace{\vec{\mu}_i^t\Sigma^{-1}}_{\vec{w}_i^t}\vec{x} \underbrace{- \frac{1}{2}\vec{\mu}_i^t\Sigma^{-1}\vec{\mu}_i + \ln P(y_i)}_{w_{i0}}$$

The separating hyperplanes are not necessarily orthogonal to the line linking the means:

$$\underbrace{(\vec{\mu}_i - \vec{\mu}_j)^t\Sigma^{-1}}_{\vec{w}^t}\left(\vec{x} - \underbrace{\frac{1}{2}(\vec{\mu}_i + \vec{\mu}_j) - \frac{\ln\frac{P(y_i)}{P(y_j)}}{(\vec{\mu}_i - \vec{\mu}_j)^t\Sigma^{-1}(\vec{\mu}_i - \vec{\mu}_j)}(\vec{\mu}_i - \vec{\mu}_j)}_{\vec{x}_0}\right))$$

7.3.2.3 Case Σ_i arbitrary

In the case where Σ_i is arbitrary the discriminant functions are inherently quadratic:

$$g_i(\vec{x}) = \vec{x}^t \underbrace{\left(-\frac{1}{2}\Sigma_i^{-1}\right)}_{W_i} \vec{x} + \underbrace{\vec{\mu}_i^t \Sigma_i^{-1}}_{\vec{w}_i^t} \vec{x} \underbrace{-\frac{1}{2}\vec{\mu}_i^t \Sigma_i^{-1} \vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln P(y_i)}_{w_{i0}}$$

In the two category case the decision surfaces are hyperquadratics like hyperplanes, pairs of hyperplanes, hyperspheres or hyperellipsoids.

7.4 Arbitrary inputs and outputs

7.4.1 Setting

Examples are input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ generated with probability $p(x, y)$.

7.4.2 Risk

7.4.2.1 Conditional risk

The conditional risk of predicting y^* given $\vec{x}$ is:

$$R(y^*|\vec{x}) = \int_{\mathcal{Y}} l(y^*, y) P(y|\vec{x}) dy$$

7.4.2.2 Overall risk

The overall risk of a decision rule g is:

$$R[f] = \int R(f(x)|x)p(x)dx = \int_{\mathcal{X}} \int_{\mathcal{Y}} l(f(x), y)p(y, x)dxdy$$

7.4.3 Bayes decision rule

Bayes decision rule states:

$$y^B = \arg\min_{y \in \mathcal{Y}} R(y|x)$$

7.5 Handling features

7.5.1 Handling missing features - marginalize over missing variables

To marginalize over missing variables one must assume that input $\vec{x}$ consists of an observed part $\vec{x}_o$ and a missing part $\vec{x}_m$. The posterior probability of y_i given the observation can be obtained from probabilities over entire inputs by marginalizing over the missing part:

$$P(y_i|\vec{x}_o) = \frac{p(y_i,\vec{x}_o)}{p(\vec{x}_o)} = \frac{\int p(y_i,\vec{x}_o,\vec{x}_m)d\vec{x_m}}{p(\vec{x}_o)} =$$
$$= \frac{\int P(y_i|\vec{x}_o,\vec{x}_m)p(\vec{x}_o,\vec{x}_m)d\vec{x}_m}{\int p(\vec{x}_o,\vec{x}_m)d\vec{x}_m} =$$
$$= \frac{\int P(y_i|\vec{x})p(\vec{x})d\vec{x}_m}{\int p(\vec{x})d\vec{x}_m}$$

7.5.2 Handling noisy features - marginalize over true variables

To marginalize over true variables on must assume that $\vec{x}$ consists of a clean part $\vec{x}_c$ and a noisy part $\vec{x}_n$. Also there is a need for a noise model for the probability of the noisy feature given its true version $p(\vec{x}_n|\vec{x}_t)$. The posterior probability of y_i given the observation can be obtained from probabilities over clean inputs by marginalizing over true variables via the noise model:

$$P(y_i|\vec{x}_x,\vec{x}_n) = \frac{p(y_i,\vec{x}_c,\vec{x}_n)}{p(\vec{x}_c,\vec{x}_n)} = \frac{\int p(y_i,\vec{x}_c,\vec{x}_n,\vec{x}_t)d\vec{x}_t}{\int p(\vec{x}_c,\vec{x}_n,\vec{x}_t)d\vec{x}_t)} =$$
$$= \frac{\int p(y_i|\vec{x}_c,\vec{x}_n,\vec{x}_t)p(\vec{x}_c,\vec{x}_n,\vec{x}_t)d\vec{x}_t}{\int p(\vec{x}_c,\vec{x}_n,\vec{x}_t)d\vec{x}_t} =$$
$$= \frac{\int p(y_i|\vec{x}_c,\vec{x}_t)p(\vec{x}_n|\vec{x}_c,\vec{x}_t)p(\vec{x}_c,\vec{x}_t)d\vec{x}_t}{\int p(\vec{x}_n|\vec{x}_c,\vec{x}_t)p(\vec{x}_c,\vec{x}_t)d\vec{x}_t}$$
$$= \frac{\int p(y_i|\vec{x})p(\vec{x}_n|\vec{x_t})p(\vec{x})d\vec{x}_t}{\int p(\vec{x}_n|\vec{x}_t)p(\vec{x})d\vec{x}_t}$$

Chapter 8

Parameter estimation

8.1 Introduction

8.1.1 Setting

Data is sampled from a probability distribution $p(x, y)$, whose form is known but its parameters are unknown. The training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of examples sampled independent and identically distributed according to $p(x, y)$.

8.1.2 Task

The task is to estimate the unknown parameters of p from training data $\mathcal{D}$. This is the same as Bayesian decision theory: there is a need to compute the posterior probability of classes given examples, except the parameters of the distributions are unknown and a training set is provided instead.

8.1.3 Multi class classification

The training set can be divided into $\mathcal{D}_1, \ldots, \mathcal{D}_c$ subsets, one for each class such that $\mathcal{D}_i = \{\vec{x}_1, \ldots, \vec{x}_n\}$ contains independent and identically distributed examples for target class y_i. For any new example $\vec{x}$ the posterior probability of the class given the example and the full training set $\mathcal{D}$ is computed:

$$P(y_i|\vec{x}, \mathcal{D}) = \frac{p(\vec{x}|y_i, \mathcal{D})p(y_i|\mathcal{D})}{p(\vec{x}|\mathcal{D})}$$

8.1.3.1 Simplifications

$\vec{x}$ can be assumed independent of $\mathcal{D}_j$ with $j \neq i$ given y_i and $\mathcal{D}_i$. Without additional knowledge $p(y_i|\mathcal{D})$ can be computed as the fraction of examples with that class in the dataset. The normalizing factor $p(\vec{x}|\mathcal{D})$ can be computed marginalizing $p(\vec{x}|y_i, \mathcal{D}_i)p(y_i|\mathcal{D})$ over possible classes. There is a need to estimate class-dependent parameters $\vec{\theta}_i$ for $p(\vec{x}|y_i, \mathcal{D}_i)$.

8.2 Maximum likelihood

Maximum likelihood or maximum a posteriori estimation assumes parameters $\vec{\theta_i}$ have fixed but unknown values. These are computed as those maximizing the probability of the observed examples $\mathcal{D}_i$. Obtained values are used to compute the probability for new examples:

$$p(\vec{x}|y_i, \mathcal{D}_i) \approx p(\vec{x}|\vec{\theta_i})$$

It assumes a prior distribution for the parameters $p(\vec{\theta_i})$ is available. This maximizes the likelihood of the parameters with respect to the training samples and there is no assumption about prior distributions for parameters.

$$\vec{\theta_i^*} = \arg\max_{\vec{\theta_i}} = p(\vec{\theta_i}|\mathcal{D}_i, y_i) = \arg\max_{\vec{\theta_i}} p(\mathcal{D}_i, y_i|\vec{\theta_i})p(\vec{\theta_i}) = \arg\max_{\vec{\theta_i}} p(\mathcal{D}_i, y_i|\vec{\theta_i})$$

8.2.1 Setting

A training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_n\}$ of independent and identically distributed examples for the target class y is available. Assuming the parameter vector $\vec{\theta}$ as a fixed but unknown value, the value maximizing its likelihood with respect to the training data is estimated:

$$\vec{\theta^*} = \arg\max_{\vec{\theta}} p(\mathcal{D}|\vec{\theta}) = \arg\max_{\vec{\theta}} \prod_{j=1}^{n} p(\vec{x}_j|\vec{\theta})$$

The joint probability over $\mathcal{D}$ decomposes into a product as examples are independent and identically distributed.

8.2.2 Maximizing log-likelihood

It is usually simpler to maximize the logarithm of the likelihood because of its monotonic nature:

$$\vec{\theta^*} = \arg\max_{\vec{\theta}} \ln p(\mathcal{D}|\vec{\theta}) = \arg\max_{\vec{\theta}} \sum_{j=1}^{n} \ln p(\vec{x}_j|\vec{\theta})$$

The necessary conditions for the maximum can be obtained zeroing the gradient with respect to $\vec{\theta}$:

$$\nabla_{\vec{\theta}} \sum_{j=1}^{n} \ln p(\vec{x}_j|\vec{\theta}) = \vec{0}$$

Points zeroing the gradient can be local or global maxima depending on the form of the distribution.

8.2.3 Univariate Gaussian case

For the univariate Gaussian with unknown μ and σ^2 the log likelihood is:

$$\mathcal{L} = \sum_{j=1}^{n} -\frac{1}{\sigma^2}(x_j - \mu)^2 - \frac{1}{2}\ln 2\pi\sigma^2$$

8.2.3.1 Mean

The gradient with respect to μ is:

$$\frac{\partial \mathcal{L}}{\partial \mu} = 2\sum_{j=1}^{n} -\frac{1}{2\sigma^2}(x_j - \mu)(-1) = \sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j - \mu)$$

Setting the gradient to zero gives mean:

$$\sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j - \mu) = 0 = \sum_{j=1}^{n}(x_j - \mu)$$
$$\sum_{j=1}^{n} x_j = \sum_{j=1}^{n} \mu$$
$$\sum_{j=1}^{n} = n\mu$$
$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$

8.2.3.2 Variance

The gradient with respect to σ^2 is:

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = \sum_{j=1}^{n} -(x_j - \mu)^2 \frac{\partial}{\partial \sigma^2}\frac{1}{2\sigma^2} - \frac{1}{2}\frac{1}{2\pi\sigma^2}2\pi =$$
$$= \sum_{j-1}^{n} -(x_j - \mu)^2 \frac{1}{2}(-1)\frac{1}{\sigma^4} - \frac{1}{2\sigma^2}$$

Setting the gradient to zero gives variance:

$$\sum_{j=1}^{n} \frac{1}{2\sigma^2} = \sum_{j=1}^{n} \frac{(x_j - \mu)^2}{2\sigma^4}$$
$$\sum_{j=1}^{n} \sigma^2 = \sum_{j=1}^{n}(x_j - \mu)^2$$
$$\sigma^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^2$$

8.2.4 Multivariate Gaussian case

For the multivariate Gaussian with unknown $\vec{\mu}$ and Σ the log-likelihood is:

$$\sum_{j=1}^{n} -\frac{1}{2}(\vec{x}_j - \vec{\mu})^t \Sigma^{-1}(\vec{x}_j - \vec{\mu}) - \frac{1}{2}\ln(2\pi)^d|\Sigma|$$

The maximum-likelihood estimates are:

$$\vec{\mu} = \frac{1}{n}\sum_{j=1}^{n}\vec{x}_j$$

$$\Sigma = \frac{1}{n}\sum_{j=1}^{n}(\vec{x}_j - \vec{\mu})(\vec{x}_j - \vec{\mu})^t$$

8.2.4.1 Proof for the mean

The gradient with respect to the mean is:

$$\nabla\vec{\mu}\sum_{j=1}^{n} -\frac{1}{2}(\vec{x}_j - \vec{\mu})^t\Sigma^{-1}(\vec{x}_j - \vec{\mu}) - \frac{1}{2}\ln(2\pi)^d|\Sigma| = \sum_{j=1}^{n}\Sigma^{-1}(\vec{x}_j - \vec{\mu})$$

Noting that $\frac{\partial}{\partial\vec{x}}\vec{x}^T A\vec{x} = A^T\vec{x} + A\vec{x} = 2A\vec{x}$ for symmetric A. Setting the gradient to zero:

$$\sum_{j=1}^{n}\Sigma^{-1}(\vec{x}_j - \vec{\mu}) = \vec{0}$$

$$\sum_{j=1}^{n}(\vec{x}_j - \vec{\mu}) = \Sigma\vec{0} = \vec{0}$$

$$\sum_{j=1}^{n}\vec{x}_j = \sum_{j=1}^{n}\vec{\mu} = n\vec{\mu}$$

$$\vec{\mu} = \frac{1}{n}\sum_{j=1}^{n}\vec{x}_j$$

8.2.4.2 Proof for the covariance

The gradient with respect to the covariance is:

$$\frac{\partial}{\partial\Sigma}\sum_{j=1}^{n} -\frac{1}{2}(\vec{x}_j - \vec{\mu})^t\Sigma^{-1}(\vec{x}_j - \vec{\mu}) - \frac{1}{2}\ln(2\pi)^d|\Sigma| =$$

$$-\frac{1}{2}\left(\sum_{j=1}^{n}\frac{\partial}{\partial\Sigma}(\vec{x}_j - \vec{\mu})^t\Sigma^{-1}(\vec{x}_j - \vec{\mu}) + \sum_{j=1}^{n}\frac{\partial}{\partial\Sigma}\ln(2\pi)^d|\Sigma|\right)$$

Now, expanding the first derivative:

$$\frac{\partial}{\partial\Sigma}(\vec{x}_j-\vec{\mu})^t\Sigma^{-1}(\vec{x}_j-\vec{\mu}) =$$
$$(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t\frac{\partial}{\partial\Sigma}\Sigma^{-1} =$$
$$-(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t\Sigma^{-2}$$

Using the matrix derivative rule: $\frac{\partial}{\partial B}tr(ABC) = CA$, where $A = (\vec{x}_j-\vec{\mu})^t$, $B = \Sigma^{-1}$ and $C = (\vec{x}_j-\vec{\mu})$ and $tr(ABC) = ABC$ as ABC is scalar. Now, expanding the second derivative:

$$\frac{\partial}{\partial\Sigma}\ln(2\pi)^d|\Sigma| = \frac{1}{(2\pi)^d}|\Sigma|^{-1}\frac{\partial}{\partial\Sigma}(2\pi)^d|\Sigma| =$$
$$\frac{1}{(2\pi)^d}|\Sigma|^{-1}(2\pi)^d\frac{\partial}{\partial\Sigma}|\Sigma| = |\Sigma|^{-1}|\Sigma|\Sigma^{-1} = \Sigma^{-1}$$

Using the matrix derivative rule: $\frac{\partial}{\partial A}|A| = |A|A^{-1}$. Combining those results and putting them equal to zero:

$$-\frac{1}{2}\left(\sum_{j=1}^{n}\overbrace{-(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t\Sigma^{-2}}^{\frac{\partial}{\partial\Sigma}(\vec{x}_j-\vec{\mu})^t\Sigma^{-1}(\vec{x}_j-\vec{\mu})}+\sum_{j=1}^{n}\overbrace{\Sigma^{-1}}^{\frac{\partial}{\partial\Sigma}\ln(2\pi)^d|\Sigma|}\right)=0$$

$$\sum_{j=1}^{n}\Sigma^{-1} = \sum_{j=1}^{n}(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t\Sigma^{-2}$$
$$\Sigma^2\sum_{j=1}^{n}\Sigma^{-1} = \Sigma^2\sum_{j=1}^{n}(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t\Sigma^{-2}$$
$$\sum_{j=1}^{n}\Sigma = \sum_{j=1}^{n}(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t$$
$$n\Sigma = \sum_{j=1}^{n}(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t$$
$$\Sigma = \frac{1}{n}\sum_{j=1}^{n}(\vec{x}_j-\vec{\mu})(\vec{x}_j-\vec{\mu})^t$$

8.2.5 General Gaussian case

In the case of a general Gaussian the maximum likelihood estimates for Gaussian parameters are their empirical estimates over the samples: the mean is the sample mean and the covariance matrix is the mean of the sample covariances.

8.3 Bayesian estimation

Bayesian estimation assumes parameters $\vec{\theta}_i$ are random variables with some known prior distribution. Observing examples turns prior distribution over parameters into a posterior distribution. Predictions for new examples are obtained integrating over all possible values for the parameters:

$$p(\vec{x}|y_i, \mathcal{D}_i) = \int_{\vec{\theta}_i} p(\vec{x}, \vec{\theta}_i|y_i, \mathcal{D}_i) d\vec{\theta}_i$$

8.3.1 Setting

Bayesian estimation assumes parameters $\vec{\theta}_i$ are random variables with some known prior distribution. Predictions for new examples are obtained integrating over all possible values for the parameters:

$$p(\vec{x}|y_i, \mathcal{D}_i) = \int_{\vec{\theta}_i} p(\vec{x}, \vec{\theta}_i|y_i, \mathcal{D}_i) d\vec{\theta}_i$$

Because probability of $\vec{x}$ given each class y_i is independent of the other classes y_i:

$$p(\vec{x}|\mathcal{D}) = \int_{\vec{\theta}} p(\vec{x}, \vec{\theta}|\mathcal{D}) d\vec{\theta} = \int p(\vec{x}|\vec{\theta}) p(\vec{\theta}|\mathcal{D}) d\vec{\theta}$$

$p(\vec{x}|\vec{\theta})$ can be easily computed because the form and the parameters of the distribution are known, so there is a need to estimate the parameter posterior density given the training set:

$$p(\vec{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\vec{\theta}) p(\vec{\theta})}{p(\mathcal{D})}$$

$P(\mathcal{D})$ is a constant independent of $\vec{\theta}$ so it will not influence the final Bayesian decision, if the final probability is needed it can be computed:

$$P(\mathcal{D}) = \int_{\vec{\theta}} p(\mathcal{D}|\vec{\theta}) p(\vec{\theta}) d\vec{\theta}$$

8.3.2 Univariate normal case - unknown μ, known σ^2

In this case the examples are drawn from $p(x|\mu) \sim N(\mu, \sigma^2)$. The Gaussian mean prior distribution is itself normal: $p(\mu) \sim N(\mu_0, \sigma_0^2)$. The Gaussian mean posterior given the dataset is computed as:

$$p(\mu|\mathcal{D}) = \frac{p(\mathcal{D}|\mu) p(\mu)}{p(\mathcal{D})} = \alpha \prod_{j=1}^{n} p(x_j|\mu) p(\mu)$$

Where $\alpha = \frac{1}{p(\mathcal{D})}$ is independent of μ.

8.3.2.1 A posteriori parameter density

$$p(\mu|\mathcal{D}) = \alpha \prod_{j=1}^{n} \overbrace{\frac{1}{\sqrt{2\pi}\sigma} e^{[-\frac{1}{2}(\frac{x_j-\mu}{\sigma})^2]}}^{p(x_j|\mu)} \overbrace{\frac{1}{\sqrt{2\pi}\sigma_0} e^{[-\frac{1}{2}(\frac{\mu-\mu_0}{\sigma_0})^2]}}^{p(\mu)} =$$

$$= \alpha' e^{[-\frac{1}{2}(\sum_{j=1}^{n}(\frac{\mu-x_j}{\sigma})^2 + (\frac{\mu-\mu_0}{\sigma_0})^2)]} =$$

$$= \alpha'' e^{[-\frac{1}{2}[(\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2})\mu^2 - 2(\frac{1}{\sigma^2}\sum_{j=1}^{n} x_j + \frac{\mu_0}{\sigma_0^2})\mu]]}$$

Where the normal distribution:

$$p(\mu|\mathcal{D}) = \frac{1}{\sqrt{2\pi}\sigma} e^{[-\frac{1}{2}(\frac{\mu-\mu_n}{\sigma^n})^2]}$$

8.3.2.2 Recovering mean and variance

$$\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right)\mu^2 - 2\left(\frac{1}{\sigma^2}\sum_{j=1}^{n} x_j + \frac{\mu_0}{\sigma_0^2}\right)\mu + \alpha''' = \left(\frac{\mu-\mu_n}{\sigma_n}\right)^2$$
$$= \frac{1}{\sigma_n^2}\mu^2 - 2\frac{\mu_n}{\sigma_n^2}\mu + \frac{\mu_n^2}{\sigma_n^2}$$

Solving for μ_n and σ_n^2:

$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_o^2 + \sigma^2}\right)\bar{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\mu_0 \qquad \sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2 + \sigma^2}$$

Where the sample mean $\bar{\mu}_n = \frac{1}{n}\sum_{j=1}^{n} x_j$.

8.3.2.3 Interpreting the posterior

The mean is a linear combination of the prior μ_0 and the sample means $\bar{\mu}_n$. The more training examples the more sample mean dominates over the prior mean. The more training examples, the more variance decreases making the distribution sharply peaked over its mean:

$$\lim_{n\to\infty} \frac{\sigma_0^2\sigma^2}{n\sigma_0^2 + \sigma^2} = \lim_{n\to\infty} \frac{\sigma^2}{n} = 0$$

8.3.2.4 Computing the class conditional density

$$p(x|\mathcal{D}) = \int p(x|\mu)p(\mu|\mathcal{D})d\mu =$$
$$= \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{1}{2}(\frac{\mu-\mu_n}{\sigma_n})^2} d\mu =$$
$$\sim N(\mu_n, \sigma^2 + \sigma_n^2)$$

The probability of x given the dataset for the class is a Gaussian with mean equal to the posterior mean, variance equal to the sum of the known variance σ^2 and an additional variance σ_n^2 due to the uncertainty of the mean.

8.3.3 Multivariate normal case - unknown μ, known Σ

This is a generalization of the univariate case:

- $p(\vec{x}|\vec{\mu}) \sim N(\vec{\mu}, \Sigma)$
- $p(\vec{\mu}) \sim N(\vec{\mu}_0, \Sigma_0)$
- $p(\vec{\mu}|\mathcal{D}) \sim N(\vec{\mu}_n, \Sigma_n)$
- $p(\vec{x}|\mathcal{D}) \sim N(\vec{\mu}_n, \Sigma + \Sigma_n)$

8.3.4 Gamma distribution

The gamma distribution is defined in the interval $[0, \infty]$. Its parameters are $\alpha > 0$ the shape and $\beta > 0$ the rate. Its probability density function is:

$$p(x; \alpha, \beta) = \frac{\mu^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

With $\mathbb{E}[x] = \frac{\alpha}{\beta}$ and $Var[x] = \frac{\alpha}{\beta^2}$. It is used to model the prior distribution of the precision.

8.3.5 Univariate normal case - unknown μ and $\lambda = \frac{1}{\sigma^2}$

In this case the samples are drawn from $p(x|\mu, \lambda) \sim N(\mu, \frac{1}{\lambda})$. The prior of mean and precision is the Normal Gamma distribution:

$$\begin{aligned} p(\mu, \lambda) &= p(\mu|\lambda)p(\lambda) = N\left(\mu|\mu_0, \frac{1}{k_0\lambda}\right) Ga(\lambda|\alpha_0, \beta_0) = \\ &= NG(\mu, \lambda|\mu_0, k_0, \alpha_0, \beta_0) \end{aligned}$$

8.3.5.1 A posteriori parameter density

$$p(\mu, \lambda|\mathcal{D}) = \frac{1}{\mathcal{D}} \prod_{j=1}^{n} \overbrace{\frac{\lambda^{\frac{1}{2}}}{\sqrt{2\pi}} e^{-\frac{\lambda}{2}(x_j-\mu)^2}}^{p(x_j|\mu,\lambda)} \overbrace{\frac{(k_0\lambda)^{\frac{1}{2}}}{\sqrt{2\pi}} e^{-\frac{k_0\lambda}{2}(\mu-\mu_0)^2}}^{p(\mu|\lambda)} \overbrace{\frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)} \lambda^{\alpha_0-1} e^{-\beta_0\lambda}}^{p(\lambda)} \propto$$

$$\propto \lambda^{\alpha_0+\frac{n}{2}-1} e^{-\beta_0\lambda} \lambda^{\frac{1}{2}} e^{\frac{\lambda}{2}[\sum_{j=1}^{n}(x_j-\mu)^2 - k_0(\mu-\mu_0)^2]}$$

The posteriori parameter density is still a Normal Gamma distribution:

$$p(\mu, \lambda|\mathcal{D}) = NG(\mu, \lambda|\mu_n, k_n, \alpha_n, \beta_n)$$

Where:

- $\mu_n = \frac{k_0\mu_0 + n\bar{\mu}_n}{k_0+n}$, the weighted average of prior μ_0 and sample means μ_n, weighted by k_0 and n respectively.
- $k_n = k_0 + n$, increased by the number of samples.
- $\alpha_n = \alpha_0 + \frac{n}{2}$ increased by half the number of samples.
- $\beta_n = \beta_0 + \frac{1}{2}\sum_{j=1}^{n}(x_j - \bar{\mu}_n)^2 + \frac{k_0 n(\bar{\mu}_n - \mu_0)^2}{2(k_0+n)}$, this is the sum of prior sum of squares β_0 and sample sum of squares and a term due to the discrepancy between the sample mean and the prior mean.

8.3.5.2 Computing the posterior predictive

$$p(x|\mathcal{D}) = \int_{\mu}\int_{\lambda} p(x|\mu,\lambda)p(\mu,\lambda|\mathcal{D})d\mu d\lambda$$
$$= \frac{P(x,\mathcal{D})}{P(\mathcal{D})} = t_{2\alpha_n}\left(x|\mu_n, \frac{\beta_n(k_n+1)}{\alpha_n k_n}\right)$$

This is a T-distribution with mean μ_n and precision $\frac{\beta_n(k_n+1)}{\alpha_n k_n}$.

8.3.6 Wishart distribution

The Wishart distribution is defined over $d \times d$ a positive semi-definite matrix. Its parameters are $\nu > d-1$, the degrees of freedom and $T > 0$ the $d \times d$ scale matrix. Its probability density function is:

$$p(X;\nu,T) = \frac{1}{2^{\nu\frac{d}{2}}|T|^{\frac{\nu}{2}}\Gamma_d(\frac{\nu}{2})}|X|^{\frac{\nu-d-1}{2}}e^{-\frac{1}{2}tr(T^{-1}X)}$$

With $\mathbb{E}[X] = \nu T$ and $Var[X_{ij}] = \nu(T_i i^2 + T_{ii}T_{jj})$. It is used to model the prior distribution of the precision matrix. T is the prior covariance.

8.3.7 Multivariate normal case - unknown μ and Σ

In this case the examples are drawn from $p(\vec{x}|\vec{\mu},\Lambda) \sim N(\vec{\mu},\Lambda^{-1})$. The prior of mean and precision is the Normal Wishart distribution:

$$p(\vec{\mu},\Lambda) = p(\vec{\mu}|\Lambda)p(\Lambda) = N(\vec{\mu}|\vec{\mu}_0,(k_0\Lambda)^{-1})Wi(\Lambda|\nu,T)$$

8.3.7.1 A posteriori parameter density

$$p(\vec{\mu},\Lambda|\mathcal{D}) = N(\vec{\mu}|\vec{\mu}_0,(k_0\Lambda)^{-1})Wi(\Lambda|\nu,T)$$

Where:

- $\mu_n = \frac{k_0\mu_0+n\bar{\mu}_n}{k_0+n}$
- $T_n = T + \sum_{i=1}^{n}(x_i-\bar{\mu}_n)(x_i-\bar{\mu}_n)^T + \frac{kn}{k+n}(\mu_0-\bar{\mu}_n)(\mu_0-\bar{\mu})n)^T$
- $\nu_n = \nu + n$
- $k_n = k + n$

8.3.7.2 Computing the posterior predictive

$$p(x|\mathcal{D}) = t_{\nu-d+1}\left(x|\mu_n, \frac{T_n(k_n+1)}{k_n(\nu_n-d+1)}\right)$$

8.4 Sufficient statistics

8.4.1 Definition

Any function on a set of samples $\mathcal{D}$ is a statistic. A statistic $\vec{s} = \phi(\mathcal{D})$ is sufficient for some parameters $\vec{\theta}$ if $P(\mathcal{D}|\vec{s}, \vec{\theta}) = P(\mathcal{D}|\vec{s})$. If $\vec{\theta}$ is a random variable, a sufficient statistic contains all relevant information $\mathcal{D}$ has for estimating it:

$$p(\vec{\theta}|\mathcal{D}, \vec{s}) = \frac{p(\mathcal{D}|\vec{\theta}, \vec{s})p(\vec{\theta}|\vec{s})}{p(\mathcal{D}|\vec{s})} = p(\vec{\theta}|\vec{s})$$

A sufficient statistic allows to compress a sample $\mathcal{D}$ into fewer values. Sample mean and covariance are sufficient statistics for true mean and covariance of the Gaussian distribution.

8.4.2 Conjugate priors

Given a likelihood function $p(\vec{x}|\vec{\theta})$ and a prior distribution $p(\vec{\theta})$, $p(\vec{\theta})$ is a conjugate prior for $p(\vec{x}|\vec{\theta})$ if the posterior distribution $p(\vec{\theta}|\vec{x})$ is in the same family as the prior $p(\vec{\theta})$

8.5 Bernoulli distribution

8.5.1 Setting

The Bernoulli distribution represent a boolean event with $x = 1$ for success and $x = 0$ for failure. Its parameter is θ, the probability of success. Its probability mass function is:

$$P(x|\theta) = \theta^x(1-\theta)^{1-x}$$

Its beta conjugate prior is:

$$P(\theta|\psi) = P(\theta|\alpha_h, \alpha_t) = \frac{\Gamma(\alpha)}{\Gamma(\alpha_h)\Gamma(\alpha_t)}\theta^{\alpha_h - 1}(1-\theta)^{\alpha_t - 1}$$

8.5.2 Maximum likelihood estimation

Let $\mathcal{D} = \{H, H, T, T, T, H, H\}$ of N realizations. Its likelihood function is:

$$p(\mathcal{D}|\theta) = \theta \cdot \theta \cdot (1-\theta) \cdot (1-\theta) \cdot (1-\theta) \cdot \theta \cdot \theta = \theta^h(1-\theta)^t$$

Its maximum likelihood parameter:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\ln p(\mathcal{D}|\theta) = 0 \Rightarrow \frac{\partial}{\partial\theta} h\ln\theta + t\ln(1-\theta) &= 0 \\
h\frac{1}{\theta} - t\frac{1}{1-\theta} &= 0 \\
h(1-\theta) &= t\theta \\
\theta &= \frac{h}{h+t}
\end{aligned}$$

h and t are the sufficient statistics.

8.5.3 Bayesian estimation

Parameter posterior is proportional to:

$$P(\theta|\mathcal{D},\psi) \propto P(\mathcal{D}|\theta)P(\theta|\psi) \propto \theta^h(1-\theta)^t\theta^{\alpha_h-1}(1-\theta)^{\alpha_t-1}$$

The posterior has a beta distribution with parameters $h+\alpha_h$ and $t+\alpha_t$:

$$P(\theta|\mathcal{D},\psi) \propto \theta^{h+\alpha_h-1}(1-\theta)^{t+\alpha_t-1}$$

The prediction for a new event is the expected value of the posterior beta:

$$\begin{aligned} P(x|\mathcal{D}) &= \int P(x|\theta)P(\theta|\mathcal{D},\psi)d\theta & &= \int \theta P(\theta|\mathcal{D},\psi)d\theta \\ &= \mathbb{E}_{P(\theta|\mathcal{D},\psi)}[\theta] & &= \frac{h+\alpha_h}{h+t+\alpha_h+\alpha_t} \end{aligned}$$

8.5.3.1 Interpreting priors

The prior knowledge is encoded as the number $\alpha = \alpha_h + \alpha_t$ of imaginary experiments. It is assumed α_t times heads was observed. α is the equivalent sample size and $\alpha \to 0$ reduces the estimation to the classical maximum likelihood approach.

8.6 Multinomial distribution

8.6.1 Setting

The multinomial distribution models categorical event with r states $x \in \{x^1, \dots, x^r\}$. One-hot encoding $\vec{z}(x) = [z_1(x), \dots, z_r(x)]$ with $z_k(x) = 1$ if $x = x^k$, o otherwise. Its parameters are $\vec{\theta} = [\theta_1, \dots, \theta_r]$ the probability of each state. Its probability mass function is:

$$P(x|\vec{\theta}) = \prod_{k=1}^{r} \theta_k^{z_k(x)}$$

Its Dirichlet conjugate prior:

$$P(\vec{\theta}|\psi) = P(\vec{\theta}|\alpha_1, \dots, \alpha_r) = \frac{\Gamma(\alpha)}{\prod_{k=1}^{r}\Gamma(\alpha_k)} \prod_{k=1}^{r} \theta_k^{\alpha_k-1}$$

8.6.2 Maximum likelihood estimation

Let $\mathcal{D}$ a dataset of N realizations. The likelihood function is:

$$p(\mathcal{D}|\vec{\theta} = \prod_{j=1}^{N}\prod_{k=1}^{r} \theta_k^{z_k(x_j)} = \prod_{k=1}^{r} \theta_k^{N_k}$$

The maximum likelihood parameter is $\theta_k = \frac{N_k}{N}$ and $N_1, \dots, N_r$ are the sufficient statistics.

8.6.3 Bayesian estimation

The parameter posterior is proportional to:

$$P(\vec{\theta}|\mathcal{D},\psi) \propto P(\mathcal{D}|\vec{\theta})P(\theta|\psi) \propto \prod_{k=1}^{r} \theta_k^{N_k}\theta_k^{\alpha_k-1}$$

The posterior has a Dirichlet distribution with parameters $N_k + \alpha_k$, where $k = 1, \ldots, r$"

$$P(\vec{\theta}|\mathcal{D},\psi) \propto \prod_{k=1}^{r} \theta_k^{N_k+\alpha_k-1}$$

The prediction for a new event is the expected value of the posterior Dirichlet:

$$P(x_k|\mathcal{D}) = \int \theta_k P(\vec{\theta}|\mathcal{D},\psi)d\vec{\theta} = \mathbb{E}_{P(\vec{\theta}|\mathcal{D},\psi)}[\theta_k] = \frac{N_k+\alpha_k}{N+\alpha}$$

Chapter 9

Bayesian networks

9.1 Introduction

All probabilistic inference and learning amount at repeated applications of the sum and product rules. Probabilistic graphical models are graphical representation of the qualitative aspects of probability distributions allowing to:

- Visualize the structure of a probabilistic model in a simple and intuitive way.
- Discover properties of the model such as conditional independences by inspecting the graph.
- Express complex computations for inference and learning in terms of graphical manipulation.
- Represent multiple probability distributions with the same graph abstracting from their quantitative aspects.

9.1.1 Bayesian networks semantics

A Bayesian network or BN structure $\mathcal{G}$ is a directed graphical model. Each node represents a random variable x_i. Each edge represents a direct dependency between two variables. The structure encodes these independence assumptions:

$$\mathcal{I}_l(\mathcal{G}) = \{\forall i \; x_i \perp NonDescendants_{x_i} | Parents_{x_i}\} \qquad \perp \equiv Independent$$

Each variable is independent of its non-descendants given its parents. These are the local independences encoded by the networks.

9.1.2 Graph and distributions

Let p be a joint distribution over variables $\mathcal{X}$. Let $\mathcal{I}(p)$ be the set of independence assertions holding in p. $\mathcal{G}$ is as independence map I-map for p if p satisfies the local independences in $\mathcal{G}$.

$$\mathcal{I}_l(\mathcal{G}) \subseteq \mathcal{I}(p)$$

The reverse is not necessarily true: there can be independences in p that are not modelled by $\mathcal{G}$.

9.1.3 Factorization

p factorizes according to $\mathcal{G}$ if:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i | Pa_{x_i})$$

If $\mathcal{G}$ is an I-map for p then p factorizes according to $\mathcal{G}$. If p factorizes according to $\mathcal{G}$ then $\mathcal{G}$ is an I-map for p.

9.1.3.1 Proof that from I-map follows factorization

If $\mathcal{G}$ is an I-map for p then p satisfies at least these local independences:

$$\{\forall i x_i \perp NonDescendants_{x_i} | Parents_{x_i}\}$$

Order variables in a topological order relative to $\mathcal{G}$:

$$x_i \to x_j \Rightarrow i < j$$

Decompose the joint probability using the chain rule as:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i | x_1, \ldots, x_{i-1})$$

Local independences imply that for each x_i:

$$p(x_i | x_1, \ldots, x_{i-1}) = p(x_i | Pa_{x_i})$$

9.1.3.2 Proof that from factorization follows I-map

If p factorizes according to $\mathcal{G}$ the joint probability can be written as:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i | Pa_{x_i})$$

Consider the last variable x_m. By the product and sum rules:

$$p(x_m | x_1, \ldots, x_{m-1}) = \frac{p(x_1, \ldots, x_m)}{p(x_1, \ldots, x_{m-1})} = \frac{p(x_1, \ldots, x_m)}{\sum\limits_{x_m} p(x_1, \ldots, x_m)}$$

Applying factorization and isolating the only term containing x_m we get:

$$= \frac{\prod_{i=1}^{m} p(x_i|Pa_{x_i})}{\sum_{x_m} \prod_{i=1}^{m} p(x_i|Pa_{x_i})} =$$

$$= \frac{p(x_m|Pa_{x_m}) \prod_{i=1}^{m-1} p(x_i|Pa_{x_i})}{\prod_{i=1}^{m-1} p(x_i|Pa_{x_i}) \sum_{x_m} p(x_m|Pa_{x_m})} =$$

$$= \frac{p(x_m|Pa_{x_m})}{\sum_{x_m} p(x_m|Pa_{x_m})} = p(x_m|Pa_{x_m})$$

This is extendable for all the variables going backward.

9.1.4 Bayesian network definition

A Bayesian network is a pair $(\mathcal{G}, p)$ where p factorizes over $\mathcal{G}$ and it is represented as a set of conditional probability distributions associated with the nodes of $\mathcal{G}$.

9.1.4.1 Factorized probability

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i|Pa_{x_i})$$

9.2 Conditional independence

Two variables a, b are independent if:

$$p(a, b) = p(a)p(b)$$

Two variables are conditionally independent given c if:

$$p(a, b|c) = p(a|c)p(b|c)$$

Independence assumptions can be verified by repeated applications of sum and product rules. Graphical models allow to directly verify them through the d-separation criterion.

9.2.1 D separation

9.2.1.1 Tail to tail

Joint distribution:

$$p(a, b, c) = p(a|c)p(b|c)p(c)$$

a and b are not independent:

$$p(a,b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$$

a and b are conditionally independent given c:

$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = p(a|c)p(b|c)$$

c is tail-to-tail with respect to the path $a \to b$ as it is connected to the tails of the two arrows.

9.2.1.2 Head to tail

Joint distribution:

$$p(a,b,c) = p(b|c)p(c|a)p(a) = p(b|c)p(a|c)p(c)$$

a and b are not independent:

$$p(a,b) = p(a)\sum_c p(b|c)p(c|a) \neq p(a)p(b)$$

a and b are conditionally independent given c:

$$p(a,b|c) = \frac{p(b|c)p(a|c)p(c)}{p(c)} = p(b|c)p(a|c)$$

c is head-to-tail with respect to the path $a \to b$ as it is connected to the head of an arrow and to the tail of the other.

9.2.1.3 Head to head

Joint distribution:

$$p(a,b,c) = p(c|a,b)p(a)p(b)$$

a and b are independent:

$$p(a,b) = p(a)\sum_c p(c|a,b)p(a)p(b) = p(a)p(b)$$

a and b are not conditionally independent given c:

$$p(a,b|c) = \frac{p(c|a,b)p(a)p(b)}{p(c)} \neq p(a|c)p(b|c)$$

c is head-to-head with respect to the path $a \to b$ as it is connected to the heads of the two arrows.

9.2.1.4 General head-to-head

Let a descendant of a node x be any node which can be reached from x with a path following the direction of the arrows. A head-to-head node c unblocks the dependency path between its parents if either itself or any of its descendants receives evidence.

9.2.2 General d-separation criterion

The sets A and B are independent given C ($A \perp B|C$) if they are d-separated by C.

9.2.2.1 d-separation definition

Given a generic Bayesian network and A, B, C arbitrary non intersecting sets of node. The sets A and B are d-separated by C ($dsep(A; B|C)$) if all paths from any node in A to any node in B are blocked. A path is blocked if it includes at least one node such that either:

- The arrows on the path meet tail-to-tail or head-to-tail at a node in C.
- The arrows on the path meet head-to-head at a node and neither it nor any of its descendants is in C.

9.3 BN independence revisited

9.3.1 Independence assumptions

A BN stricture $\mathcal{G}$ encodes a set of local independence assumptions:

$$\mathcal{I}_l(\mathcal{G}) = \{\forall i \; x_i \perp NonDescendants_{x_i} | Parents_{x_i}\}$$

A BN structure $\mathcal{G}$ encodes a set of global or Markov independence assumptions:

$$\mathcal{I}(\mathcal{G}) = \{(A \perp B|C) : dsep(A; B|C)\}$$

9.3.2 BN equivalence classes

9.3.2.1 I-equivalence

Quite different BN structures can encode the exact same set of independence assumptions. Two BN structures $\mathcal{G}$ and $\mathcal{G}'$ are I-equivalent if $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{G}')$. The space of BN structures over $\mathcal{X}$ is partitioned into a set of mutually exclusive and exhaustive I-equivalence classes.

9.3.2.1.1 Sufficient conditions If two structures $\mathcal{G}$ and $\mathcal{G}'$ have the same skeleton and the same set of v-structures then they are I-equivalent.

9.3.2.1.2 Necessary and sufficient conditions Two structures $\mathcal{G}$ and $\mathcal{G}'$ are I-equivalent if and only if they have the same skeleton and the same set of immoralities.

9.3.2.1.3 Equivalence class

9.3.2.1.3.1 Partially directed acyclic graph A partially directed acyclic graph or PDAG is an acyclic graph with both directed and undirected edges.

9.3.2.1.3.2 Representing an equivalence class An equivalence class for a structure $\mathcal{G}$ can be represented by a PDAG $\mathcal{K}$ such that:

- If $x \to y \in \mathcal{K}$ then $x \to y$ should appear in all structures which are I-equivalent to $\mathcal{G}$.
- If $x - y \in \mathcal{K}$ then we can find a structure $\mathcal{G}'$ that is I-equivalent to $\mathcal{G}$ such that $x \to y \in \mathcal{G}'$

9.3.2.1.3.3 Generating members Representative from $\mathcal{K}$ can be obtained by adding directions to undirected edges. One needs to check that the resulting structure has the same set of immoralities as $\mathcal{K}$.

9.3.3 I-maps and distributions

9.3.3.1 Minimal I-maps

For a structure $\mathcal{G}$ to be an I-map for p, it does not need to encode all its independences: a fully connected graph is an I-map of any p defined over its variables. A minimal I-map for p is an I-map $\mathcal{G}$ which can't be reduced into a $\mathcal{G}' \subset \mathcal{G}$ by removing edges that is also an I-map for p. A minimal I-map for p does not necessarily capture all the independences in p.

9.3.3.2 Perfect maps P-maps

A structure $\mathcal{G}$ is a perfect map for p if it captures all and only its dependences:

$$\mathcal{I}(\mathcal{G}) = \mathcal{I}(p)$$

The algorithm for finding such P-map is exponential in the in-degree of the P-map. The algorithm returns an equivalence class rather than a single structure. Not all distributions have a P-map: some cannot be modelled exactly by the BN formalism.

9.3.4 Building Bayesian networks

1. Get together with a domain expert.
2. Define variables for entities that can be observed or that you can be interested in predicting.
3. Try following causality considerations in adding edges.
4. In defining probabilities for configurations never assign zero probabilities.
5. If data is available, use them to help in learning parameters and structure.

9.4 Markov blanket or boundary

9.4.1 Definition

Given a directed graph with m nodes the Markov blanket of node x_i is the minimal set of nodes making it x_i independent on the rest of the graph:

$$p(x_i|x_{j\neq i}) = \frac{p(x_1,\ldots,x_m)}{p(x_{j\neq i})} = \frac{p(x_1,\ldots,x_m)}{\int p(x_1,\ldots,x_m)dx_i} = \\ = \frac{\prod_{k=1}^{m} p(x_k|pa_k)}{\int \prod_{k=1}^{m} p(x_k|pa_k)dx_i}$$

All components which do not include x_i will cancel between numerator and denominator. The only remaining components are:

- $p(x_i|pa_i)$ the probability of x_i given its parents.
- $p(x_j|pa_j)$ where pa_j includes x_i, the children of x_i with their co-parents.

9.4.2 d-separation

Each parent x_j of x_i will be head-to-tail or tail-to-tail in the path to x_i and any of x_j other neighbours, so the paths are blocked. Each child x_j of x_i will be head-to-tail in the path to x_i and any of x_j children, so the paths are blocked. Each co-parent x_k of a child x_j of x_i will be head-to-tail or tail-to-tail in the path to x_j and any of x_k other neighbours, so all paths are blocked.

Chapter 10

Bayesian networks inference

10.1 Inference in graphical models

Assume that there is evidence $\vec{e}$ on the state of a subset of variables $\vec{W}$ in the model. Inference amounts at computing the posterior probability of a subset $\vec{X}$ of the non-observed variables given the observations: $p(\vec{X}|\vec{E} = \vec{e})$.

10.1.1 Efficiency

The posterior probability can always be computed as the ratio of two joint probabilities:

$$p(\vec{X}|\vec{E} = \vec{e}) = \frac{p(\vec{X}, \vec{E} = \vec{e})}{p(\vec{E} = \vec{e})}$$

The problem consists of estimating such joint probabilities when dealing with a large number of variables. Directly working on the full joint probabilities requires time exponential in the number of variables: if all N variables are discrete and take one of K possible values, a joint probability table has K^N entries. There is a way to exploit the structure in graphical models to do inference more efficiently.

10.1.2 Inference on a chain

Let a probability chain be:

$$p(\vec{X}) = p(X_1)p(X_2|X_1)p(X_3|X_2) \cdot p(X_N|X_{N-1})$$

The marginal probability of an arbitrary X_n is:

$$p(X_n) = \sum_{X_1} \cdots \sum_{X_{n-1}} \sum_{X_{n+1}} \cdot \sum_{X_N} p(\vec{X})$$

Only the $p(X_n|X_{N-1})$ is involved in the last summation which can be computed first, giving a function of X_{N-1}:

$$\mu_\beta(X_{N-1}) = \sum_{X_N} p(X_N|X_{N-1})$$

The marginalization can be iterated as:

$$\mu_\beta(X_{N-2}) = \sum_{X_{N-1}} p(X_{N-1}|X_{N-2})\mu_\beta(X_{N-1})$$

Down to the desired variable X_n, giving:

$$\mu_\beta(X_n) = \sum_{X_{n+1}} p(X_{n+1}|X_n)\mu_\beta(X_{n+1})$$

The same procedure can be applied starting from the other end of the chain, giving:

$$\mu_\alpha(X_2) = \sum_{X_1} p(X_1)p(X_2|X_1)$$

Up to $\mu_\alpha(X_n)$. The marginal probability is now computed as the product of the contributions coming from both ends:

$$p(X_n) = \mu_\alpha(X_n)\mu_\beta(X_n)$$

10.1.3 Inference as message passing

$\mu_\alpha(X_n)$ can be though as a message passing from X_{n-1} to X_n:

$$\mu_\alpha(X_n) = \sum_{X_{n-1}} p(X_n|X_{n-1})\mu_\alpha(X_{N-1})$$

$\mu_\beta(X_n)$ can be thought as a message passing from X_{n+1} to X_n:

$$\mu_\beta(X_n) = \sum_{X_{n+1}} p(X_{n+1}|X_n)\mu_\beta(X_{n+1})$$

Each outgoing message is obtained multiplying the incoming message by the local probability and summing over the node values.

10.1.4 Full message passing

Let's suppose we want to know marginal probabilities for a number of different variables X_i:

1. A message is sent from $\mu_\alpha(X_1)$ up to $\mu_\alpha(X_N)$.
2. A message is set from $\mu_\beta(X_N)$ down to $\mu_\beta(X_1)$.

If all nodes store messages any marginal probability can be computed as:

$$p(X_i) = \mu_\alpha(X_i)\mu_\beta(X_i)$$

For any i having sent a double number of messages with respect to a single marginal computation.

10.1.5 Adding evidence

If some nodes $\vec{X}_e$ are observed their observed values are used instead of summing over all possible values when computing their messages.

10.1.6 Computing conditional probability given evidence

When adding evidence, the message passing procedure computes the joint probability of the variable and the evidence, and it has to be normalized to obtain the conditional probability given the evidence:

$$p(X_n|\vec{X}_e = \vec{x}_e) = \frac{p(X_n, \vec{X}_e = \vec{x}_e)}{\sum_{X_n} p(X_n, \vec{X}_e = \vec{x}_e)}$$

10.1.7 Inference on trees

Efficient inference can be computed for the broad family of tree-structured models: undirected trees, directed trees and directed polytrees.

10.2 Factor graphs

10.2.1 Description

Efficient inference algorithms can be better explained using an alternative graphical representation called factor graph. A factor graph is a graphical representation of a graphical model highlighting its factorizations (conditional probabilities). The factor graph has one node for each node in the original graph, and one additional of a different type for each factor. A factor node has undirected links to each of the node variables in the factor.

10.2.2 Sum-product algorithm

The sum-product algorithm is an efficient algorithm for exact inference on tree-structured graphs. It is a message passing algorithm as its simpler version for chain. It can be applied to undirected models and directed ones.

10.2.2.1 Computing marginals

The marginal probability of X is $p(X) = \sum_{\vec{X}\setminus X} p(\vec{X})$. Generalizing the message passing scheme, this can be computed as the product of messages coming from all neighbouring factors f_s:

$$p(X) = \prod_{f_s \in ne(X)} \mu_{f_s \to X}(X)$$

10.2.2.2 Factor messages

Each factor message is the product of messages coming from nodes other than X, times the factor, summed over all possible values of the factor variables other than X:

$$\mu_{f_s \to X}(X) = \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{X_m \in ne(f_s)\setminus X} \mu_{X_m \to f_s}(X_m)$$

10.2.2.3 Node messages

Each message from node X_m to factor f_s is the product of the factor messages to X_m coming from factors other than f_s:

$$\mu_{X_m \to f_s}(X_m) = \prod_{f_l \in ne(X_m) \setminus f_s} \mu_{f_l \to X_m}(X_m)$$

10.2.2.4 Initialization

Message passing start from leaves, factors or nodes. Messages from leaf factors are initialized to the factor itself: $\mu_{f \to x}(x) = f(x)$. Messages from leaf nodes are initialized to 1: $\mu_{x \to f}(x) = 1$.

10.2.2.5 Message passing scheme

The node X whose marginal has to be computed is designed as root. Messages are sent from all leaves to their neighbours. Each internal node sends its message towards the root as soon as it received messages from all other neighbours. Once the root as collected all messages, the marginal can be computed as the product of them.

10.2.2.6 Full message passing scheme

In order to be able to compute marginals for any node, messages need to pass in all directions:

1. Choose an arbitrary node as root.
2. Collect messages for the root starting from leaves.
3. Send messages from the root down to the leaves.

All messages are passed in all directions using only twice the number of computations used for a single marginal.

10.2.2.7 Adding evidence

If some nodes $\vec{X}_e$ are observed, their values are used instead of summing over all possible values when computing their messages. After normalization, this gives the conditional probability given the evidence.

10.3 Finding the most probable configuration

Given a joint probability distribution $p(\vec{X})$, there is a need to find the configuration for variables $\vec{X}$ having the highest probability:

$$\vec{X}^{\max} = \arg\max_{\vec{X}} p(\vec{X})$$

For which the probability is $p(\vec{X}^{\max}) = \max_{\vec{X}} p(\vec{X})$. Because the focus is the jointly maximal for all variables the sum-product algorithm cannot be used.

10.3.1 The max-product algorithm

$$p(\vec{X}^{\max}) = \max_{\vec{X}} p(\vec{x}) = \max_{X_1} \cdots \max_{X_M} p(\vec{X})$$

As for the sum-product algorithm the distribution factorization can be exploited to efficiently compute the maximum. To do so it is sufficient to replace the sum with the max in the sum-product algorithm.

10.3.1.1 Linear chain

$$\max_{\vec{X}} p(\vec{X}) = \max_{X_1} \cdots \max_{X_N} [p(X_1)p(X_2|X_1) \cdots p(X_N|X_{N-1})] =$$
$$\max_{X_1} [p(X_1)p(X_2|X_1)[\cdots \max_{X_N} p(X_n|X_{N-1})]]$$

10.3.1.2 Message passing

As for the sum-product algorithm, the max-product can be seen as message passing over the graph. The algorithm is easily applied to tree-structured graphs via their factor trees:

$$\mu_{f\to X}(X) = \max_{X_1,\ldots,X_M} \left[f(X, X_1, \ldots, X_M) \prod_{X_m \in ne(f)\setminus X} \mu_{X_m \to f} f(X_m) \right]$$
$$\mu_{X\to f}(X) = \prod_{f_l \in ne(X)\setminus f} \mu_{f_l \to X}(X)$$

10.3.1.3 Recovering maximal configuration

Messages are passed from leaves to an arbitrarily chosen root X_r. The probability of maximal configuration is obtained as:

$$p(\vec{X}^{\max}) = \max_{X_r} \left[\prod_{f_l \in ne(X_r)} \mu_{f_l \to X_r}(X_r) \right]$$

The maximal configuration for the root is obtained as:

$$X_r^{\max} = \arg\max_{X_r} \left[\prod_{f_l \in ne(X_r)} \mu_{f_l \to X_r}(X_r) \right]$$

The maximal configuration is to recover for the other variables. When sending a message towards x each factor node should store the configuration of the other variables which gave the maximum:

$$\phi_{f\to X}(X) = \arg \max_{X_1,\ldots,X_M} \left[f(X, X_1, \ldots, X_M) \prod_{X_m \in ne(f)\setminus X} \mu_{X_m \to f}(X_m) \right]$$

When the maximal configuration for the root node X_r has been obtained it can be used to retrieve the maximal configuration for the variables in neighbouring factors from:

$$X_1^{\max}, \ldots, X_M^{\max} = \phi_{f\to X_r}(X_r^{\max})$$

The procedure can be repeated back-tracking to the leaves, retrieving maximal values for all variables.

10.3.1.4 Trellis for linear chain

A trellis or lattice diagram shows the K possible states of each variable X_n one per row. For each state k of a variable $X_n, \phi_{f_{n-1,n} \to X_n}(X_n)$ defines a unique and maximal previous state linked by an edge in the diagram. Once the maximal state for the last variable X_N is chosen, the maximal states for other variables are recovered following the edges backward.

10.4 Approximate inference

10.4.1 Underflow issues

The max-product algorithm relies on products, which for many small probabilities can lead to an underflow problem. This can be addressed computing the logarithm of the probability instead. The logarithm is monotonic and the proper maximal configuration is recovered:

$$\log(\max_{\vec{X}} p(\vec{X})) = \max \vec{X} \log p(\vec{X})$$

The effect is replacing products with sums of logs in the max-product algorithm, giving the max-sum one.

10.4.2 Exact inference on general graphs

The sum-product and max-product algorithms can be applied to tree-structured graphs, with many application requiring graphs with undirected loops. An extension of this algorithms to generic graphs can be achieved with the junction tree algorithm. This algorithm works on junction trees, tree-structured graphs with nodes containing clusters of variables of the original graph. A message assing scheme analogous to the sum-product and max-product algorithms is run on the junction tree.

10.4.3 Problem with exact inference

The complexity on the algorithm is exponential on the maximal number of variables in a cluster, making it intractable for large complex graphs. In cases in which exact inference is intractable, approximate inference techniques are used.

10.4.4 Loopy belief propagation

In loopy belief propagation the message passing is done on the original graph even if it contains loops. This is done through the sum-product algorithm. All nodes are assumed in condition of sending messages. A message passing schedule is chosen in order to decide which nodes start sending messages, like flooding (all nodes send messages in all directions at each time step). Information flows many times around the graph, each message on a link replaces the previous one and is only based on the most recent messages received from the other neighbours. The algorithm can eventually converge depending on the specific model over which it is applied.

10.4.5 Variational methods

Variational methods are deterministic approximations, assuming the posterior probability given the evidence factorizes in a particular way.

10.4.6 Sampling methods

Sampling methods approximate the posterior sampling from the network. Given the joint probability distribution $p(\vec{X})$ a sample from the distribution is an instantiation of all the variables $\vec{X}$ according to probability p. Samples can be used to approximate the probability of a certain assignment $\vec{Y} = \vec{y}$ for a subset $\vec{Y} \subset \vec{X}$ of the variables, counting the fraction of samples which are consistent with the desired assignemnt. There is a need to sample from a posterior distribution given some evidence $\vec{E} = \vec{e}$.

10.4.6.1 Markov chain monte Carlo

It is usually to expensive to directly sample from the posterior distribution, so a random process is build that gradually samples from distributions closer and closer to the posterior. The state of the process is an instantiation of all the variables. The process can be seen as randomly traversing the graph of states moving from one state to another with a certain probability. After enough time, the probability of being in any particular state is the desired posterior. The random process is a Markov Chain. The state graph is very different from the graphical model of the joint distribution: nodes in the state graph are instantiations of all the variables and not single variables.

10.5 Markov chain

10.5.1 Definition

A Markov chain consist of a space $Val(\vec{X})$ of possible states, one for each possible instantiation of all variables $\vec{X}$. A transition probability model defining for each state $\vec{x} \in Val(\vec{X})$ a next-state distribution over $Val(\vec{X})$. The transition probability from $\vec{x}$ to $\vec{X}'$ is $\mathcal{T}(\vec{x} \to \vec{x}')$. A Markov chain defines a stochastic process that evolves from state to state.

10.5.2 Homogeneous chains

A Markov chain is homogeneous if its transition probability model does not change over time. Transition probability between states do not depend on the particular time instant in the process. Those chains are useful for sampling.

10.5.3 Chain dynamics

Considering a sequence of states of the random process. This at time t can be seen as a random variable $\vec{X}^{(t)}$. Assuming that the initial state $\vec{X}^{(0)}$ is distributed according to some initial probability $p^{(0)}(\vec{X}^{(0)})$, the distribution over subsequent states can be defined using the chain dynamics as:

$$p^{(t+1)}(\vec{X}^{(t+1)} = \vec{x}') = \sum_{\vec{x} \in Val(\vec{X})} p^{(t)}(\vec{X}^{(t)} = \vec{x}) \mathcal{T}(\vec{x} \to \vec{x}')$$

The probability of being in a certain state $\vec{x}$ at time t is the sum of the probabilities of being in any possible state $\vec{x}'$ at time $t-1$ times the probability of a transition from $\vec{x}'$ to $\vec{x}$.

10.5.4 Convergence

As the process converges, $p^{(t+1)}$ should become close to $p^{(t)}$:

$$p^{(t)}(\vec{x}') \approx p^{(t+1)}(\vec{x}') = \sum_{\vec{x} \in Val(\vec{X})} p^{(t)}(\vec{x}) \mathcal{T}(\vec{x} \to \vec{x}')$$

At convergence an equilibrium distribution $\pi(\vec{X})$ is reached and the probability of being in a state is the same as that of transitioning into it from a random predecessor.

10.5.5 Stationary distribution

A distribution $\pi(\vec{X})$ is a stationary distribution for a Markov chain $\mathcal{T}$ if:

$$\pi(\vec{X} = \vec{x}') = \sum_{\vec{x} \in Val(\vec{X})} \pi(\vec{X} = \vec{x}) \mathcal{T}(\vec{x} \to \vec{x}')$$

10.5.6 Requirements

For sampling a Markov Chain must have a unique stationary distribution which is reachable from any starting distribution $p^{(0)}$. For a Markov chain with a finite state space, regularity is a sufficient and necessary condition.

10.5.7 Regular chains

A Markov chain is regular if there exists a number k such that for all state pairs $\vec{x}, \vec{x}' \in Val(\vec{X})$ the probability of getting from $\vec{x}$ to $\vec{x}'$ in exactly k steps is greater than zero. Regularity is ensured if it is possible to get from any state to any other using a positive probability path in the state graph and for every state $\vec{x}$ there is a positive probability of transitioning from $\vec{x}$ to $\vec{x}$ in one step.

10.5.8 Markov chains for graphical models

The typical use is estimating the probability of a certain instantiation $\vec{x}$ of the variables $\vec{X}$ given some evidence $\vec{E} = \vec{e}$. This requires sampling from the posterior distribution $p(\vec{X}|\vec{E} = \vec{e})$. There is a need to define a chain for which $p(\vec{X}|\vec{E} = \vec{e})$ is the stationary distribution. States in the chain should be the subset of all possible instantiations which is consistent with the evidence: $\vec{x} \in Val(\vec{X})|\vec{x}_{\vec{E}} = \vec{e}$.

10.5.9 Transition model

A simple transition model consists of updating variables in $\bar{\vec{X}} = \vec{X} - \vec{E}$ one at a time. This can be seen as made of a set of $k = |\bar{\vec{X}}|$ local transition model $\mathcal{T}_i$, one for each variable $X_i \in \bar{\vec{X}}$. Let $\vec{U}_i = \vec{X} - X_i$ and $\vec{u}_i$ a possible instantiation of $\vec{U}_i$. The local transition model $\mathcal{T}_i$ defines transitions between states $(\vec{u}_i, x_i)$ and $(\vec{u}_i, x_i')$. By defining a transition as a sequence of k local transitions, one for each variable X_i, a homogeneous global transition model is obtained.

10.5.10 Gibbs sampling

Gibbs sampling is an efficient and effective simple Markov chain for factored state spaces. Local transitions $\mathcal{T}_i$ are defined forgetting about the value of X_i in the current state. This allows to sample a new value according to the posterior probability of X_i given the rest of the current state:

$$\mathcal{T}_i((\vec{u}_i, x_i) \to (\vec{u}_i, x_i')) = p(x_i'|\vec{u}_i)$$

The Gibbs chain samples a new state performing the rest of the current state:

$$\mathcal{T}_i((\vec{u}_i, x_i) \to (\vec{u}_i, x_i')) = p(x_i'|\vec{u}_i)$$

The Gibbs chain samples a new state performing k subsequent local moves. Samples are taken only after a full round of local moves.

10.5.10.1 Regularity

Gibbs chains are ensured to be regular if the distribution is strictly positive: every value of X_i given any instantiation of $\vec{U}_i$ has non-zero probability. In this case we can get from any state to any other in at most $k = |\vec{X}|$ local steps. Gibbs chains have the desired posterior $p(\vec{X}|\vec{E} = \vec{e})$ as stationary distribution.

10.5.10.2 Positivity

Positivity of the distributions is guaranteed if no conditional probability distribution or clique potential has zero value for any configuration of the variables.

10.5.11 Computing local transition probabilities

Local transition probabilities can be computed very efficiently for discrete graphical models. Only the Markov blanket of X_i is needed in order to compute its posterior $p(X_i|\vec{u}_i)$. Other models for which such posterior is not efficiently computable require more complex Markov chain methods such as the Metropolis-Hastings.

10.5.12 Generating samples

In order to generate samples from the correct posterior distribution the Markov chain has to converge to the stationary distribution. After that all subsequent samples could be used to estimate the desired probability. Consecutive samples are not independent. A possible approach consist of discarding an interval of d samples before collecting the next one. It is often the case that using all samples leads to better estimates if a fixed amount of time is provided.

Chapter 11

Learning Bayesian networks

11.1 Parameter estimation

Let's assume the structure of the model is given and there is a dataset of examples $\mathcal{D} = \{\vec{x}(1), \ldots, \vec{x}(N)\}$. Each example $\vec{x}(i)$ is a configuration for all or some variables in the model. There is a need to estimate the parameters of the model from the data. The simplest approach consists of learning the parameters maximizing the likelihood of the data:

$$\vec{\theta}^{\max} = \arg\max_{\vec{\theta}} p(\mathcal{D}|\vec{\theta}) = \arg\max_{\vec{\theta}} \mathcal{L}(\mathcal{D}, \vec{\theta})$$

11.1.1 Maximum likelihood estimation - complete data

11.1.1.1 Learning Bayesian networks

$$\begin{aligned}
p(\mathcal{D}|\vec{\theta}) &= \prod_{i=1}^{N} p(\vec{x}(i)|\vec{\theta}) && \text{examples independent given } \theta \\
&= \prod_{i=1}^{N}\prod_{j=1}^{m} p(x_j(i)|pa_j(i), \vec{\theta} && \text{factorization for BAN} \\
&= \prod_{i=1}^{N}\prod_{j=1}^{m} p(x_j(i)|pa_j(i), \vec{\theta}_{X_j|pa_j} && \text{disjoint CPD parameters}
\end{aligned}$$

11.1.1.2 Learning graphical models

The parameters of each CPD can be estimated independently:

$$\vec{\theta}^{\max}_{X_j|pa_j} = \arg\max_{\vec{\theta}_{X_j|pa_j}} \underbrace{\prod_{i=1}^{M} p(x_j(i)|pa_j(i), \vec{\theta}_{X_j|pa_j})}_{\mathcal{L}(\vec{\theta}_{x_j|pa_j}, \mathcal{D})}$$

A discrete CPD $P(X|\mathcal{U})$ can be represented as a table with a number of rows equal to the number $Val(X)$ of configurations for X, a number of columns equal to the number $Val(\vec{U})$ of configurations

for its parents $\vec{U}$ and each table entry $\theta_{x|\vec{u}}$ indicating the probability of a specific configuration of $X = x$ and its parents $\vec{U} = \vec{u}$. Replacing $p(x(i)|pa(i))$ with $\theta_{x(i)|\vec{u}(i)}$ the local likelihood of a single CPD becomes:

$$\begin{aligned}\mathcal{L}(\vec{\theta}_{X|pa}, \mathcal{D} &= \prod_{i=1}^{N} p(x(i)|pa(i), \vec{\theta}_{X|pa_j}) = \\ &= \prod_{i=1}^{N} \theta_{x(i)|\vec{u}(i)} = \\ &= \prod_{\vec{u} \in Val(\vec{U})} \left[\prod_{x \in Val(X)} \theta_{x|\vec{u}}^{N_{\vec{u},x}} \right]\end{aligned}$$

Where $N_{\vec{u},x}$ is the number of times the specific configuration $X = x$, $\vec{U} = \vec{u}$ was found in the data. A column in the CPD table contains a multinomial distribution over values of X for a certain configuration of the parents $\vec{U}$. Thus each column should sum to one: $\sum_x \theta_{x|\vec{u}} = 1$. Parameters of different columns can be estimated independently and for each multinomial distribution, zeroing the gradient of the maximum likelihood and considering the normalization constraint:

$$\theta_{x|\vec{u}}^{\max} = \frac{N_{\vec{u}|x}}{\sum_x N_{\vec{u},x}}$$

The maximum likelihood parameters are the fraction of times in which the specific configuration was observed in the data.

11.1.2 Learning graphical models - adding priors

ML estimation tends to overfit the training set: a configuration that does not appear in the training set will receive zero probability. A common approach consists of combining ML with a prior probability on the parameters, achieving a maximum-a-posteriori estimate:

$$\vec{\theta}^{\max} = \arg\max_{\vec{\theta}} p(\mathcal{D}|\vec{\theta}) p(\vec{\theta})$$

11.1.2.1 Dirichlet prior

The conjugate prior for a multinomial distribution is a Driblet distribution with parameters $\alpha_{x|\vec{u}}$ for each possible value of x. The resulting maximum-a-posteriori estimate is:

$$\theta_{x|\vec{u}}^{\max} = \frac{N_{\vec{u},x} + \alpha_{x|\vec{u}}}{\sum_x (N_{\vec{u},x} + \alpha_{x|\vec{u}})}$$

Introducing a prior is like observing $\alpha_{x|\vec{u}}$ imaginary samples with configuration $X = x$ and $\vec{U} = \vec{u}$.

11.1.3 Learning with missing data

With incomplete data, some of the examples miss evidence on some of the variables. Counts of occurrences of different configurations cannot be computed if not all data is observed. The full Bayesian approach of integrating over missing variables is often intractable in practice. There is a need to introduce approximate methods to deal with it.

11.1.3.1 Expectation-maximization for Bayesian networks

With missing data a sufficient statistics like counts cannot be computed. Missing data can be inferred using the current parameters getting expected counts solving the inference problem. The maximizing likelihood of such expected counts is used to compute the parameter and the process is iterated to improve their quality.

11.1.3.2 Expectation maximization algorithm

11.1.3.2.1 e-step During the e-step the sufficient statistics for the compete dataset are computed, with expectation taken with respect to the joint distribution for $\vec{X}$ conditioned of the current value of $\vec{\theta}$ and the known data $\mathcal{D}$:

$$\mathbb{E}_{p(\vec{x}|\mathcal{D},\vec{\theta})}[N_{ijk}] = \sum_{k=1}^{n} p(X_i(l) = x_k, pa_i(l) = pa_j|\vec{X}_l, \vec{\theta})$$

If $X_i(l)$ and $pa_j(l)$ are observed for $\vec{X}_l$ it is either zero or one, otherwise Bayesian inference is run to compute probabilities from observed variables.

11.1.3.2.2 m-step During the m-step parameters maximizing likelihood of the completed dataset $\mathcal{D}_c$ are computed using expected counts:

$$\vec{\theta}^* = \arg\max_{\vec{\theta}} p(\mathcal{D}_x|\vec{\theta})$$

For each multinomial the parameter evaluates to:

$$\theta^*_{ijk} = \frac{\mathbb{E}_{p(\vec{x}|\mathcal{D},\vec{\theta})}[N_{ijk}]}{\sum_{k=1}^{r_i} \mathbb{E}_{p(\vec{x},\mathcal{D},\vec{\theta})}[N_{ijk}]}$$

ML estimation can be replaced by maximum a-posteriori estimation:

$$\theta^*_{ijk} = \frac{\alpha_{ijk} + \mathbb{E}_{p(\vec{x}|\mathcal{D},\vec{\theta})}[N_{ijk}]}{\sum_{k=1}^{r_i} (\alpha_{ijk} + \mathbb{E}_{p(\vec{x},\mathcal{D},\vec{\theta})}[N_{ijk}])}$$

11.2 Learning the structure of graphical models

11.2.1 Model averaging approach

In the model-averaging approach a prior probability is assigned to each structure, and the average prediction over all possible structures weighted by their probability is taken. This is a full Bayesian

approach and often intractable. Let $\mathcal{S}$ be the space of possible structures or DAGS for domain $\vec{X}$. Let $\mathcal{D}$ be a dataset of observations. Prediction for a new instance are computed marginalizing over both structures and parameters:

$$\begin{aligned} p(\vec{X}_{N+1}|\mathcal{D} &= \sum_{S\in\mathcal{S}} \int_{\vec{\theta}} P(\vec{X}_{N+1}, S, \vec{\theta}|\mathcal{D} d\vec{\theta} = \\ &= \sum_{S\in\mathcal{S}} \int_{\vec{\theta}} P(\vec{X}_{N+1}|S, \vec{\theta}, \mathcal{D}) P(S, \vec{\theta}|\mathcal{D}) d\vec{\theta} = \\ &= \sum_{S\in\mathcal{S}} \int_{\vec{\theta}} P(\vec{X}_{N+1}|S, \vec{\theta}) P(\vec{\theta}|S, \mathcal{D}) P(S|\mathcal{D}) d\vec{\theta} = \\ &= \sum_{S\in\mathcal{S}} P(S|\mathcal{D}) \int_{\vec{\theta}} P(\vec{X}_{N+1}|S, \vec{\theta}) P(\vec{\theta}|S, \mathcal{D}) d\vec{\theta} \end{aligned}$$

11.2.1.1 Model selection

Averaging all possible structures is too expensive. So a best structure S^* is chosen and $P(S^*|\mathcal{D}) = 1$ is assumed. S^* can be chosen using a score-based or constraint-based approach.

11.2.2 Constraint based approach

In the constraint-based approach conditional independences are tested on the data and a model satisfying them is constructed.

11.2.3 Score based approach

In the score-based approach a score is assigned to each possible structure, a search procedure is defined to look for the structure maximizing the score. The structure can be scored using a maximum-likelihood score:

$$S^* = \arg\max_{S\in\mathcal{S}} p(\mathcal{D}|S)$$

Or using a maximum-a-posteriori score:

$$S^* = \arg\max_{S\in\mathcal{S}} p(\mathcal{D}|S)p(S)$$

11.2.3.1 Maximum likelihood approximation

The easiest solution is to approximate $P(\mathcal{D}|S)$ with the maximum-likelihood score over the parameters:

$$P(\mathcal{D}|S) \approx \max_{\theta} P(\mathcal{D}|S, \theta)$$

This is an addition of a connection between two variables if their empirical mutual information over the training set is non-zero. Because of noise, empirical mutual information between variables is almost never exactly zero, so it builds a fully connected network.

11.2.3.2 Bayesian-Dirichlet scoring

Let $P(\mathcal{D}|S) \equiv P_S(\mathcal{D})$.

11.2.3.2.1 Simple case

11.2.3.2.1.1 Setting Let X a single variable with r possible realizations. S is a single node. The probability distribution is a multinomial with $\alpha_i, \ldots, \alpha_r$ Dirichlet prior. $\mathcal{D}$ is a sequence of N realizations.

11.2.3.2.1.2 Approach Sort $\mathcal{D}$ according to outcome:

$$\mathcal{D} = \{x^1, x^1, \ldots, x^1, x^2, \ldots, x^2, \ldots, x^r, \ldots x^r\}$$

Its probability can be decomposed as:

$$P_S(\mathcal{D}) = \prod_{y=1}^{N} P_S(X(t)|\underbrace{X(t-1), \ldots, X(1)}_{\mathcal{D}(t-1)})$$

The prediction for a new event given the past is:

$$P_s(X(t+1) = x^t|\mathcal{D}(t)) = \mathbb{E}_{P_S(\vec{\theta}|\mathcal{D}(t))}[\theta_k] = \frac{\alpha_k + N_k(t)}{\alpha + t}$$

Where $N_k(t)$ is the number of times $X = x^k$ in the first t examples in $\mathcal{D}$. So:

$$P_S(\mathcal{D}) = \frac{\alpha_1}{\alpha}\frac{\alpha_1+1}{\alpha+1}\cdots\frac{\alpha_1+N_1-1}{\alpha+N_1-1}\frac{\alpha_2}{\alpha+N_1}\cdots\frac{\alpha_2+N_2-1}{\alpha+N_1+N_2-1}\frac{\alpha_r}{\alpha+N_1\cdots+N_{r-1}}\cdots\frac{\alpha_r+N_r-1}{\alpha+N-1} =$$
$$= \frac{\Gamma(\alpha)}{\Gamma(\alpha+N)}\prod_{k=1}^{r}\frac{\Gamma(\alpha_k+N_k)}{\alpha_k}$$

Where the Gamma function $\Gamma(x+1) = x\Gamma(x)$:

$$\alpha(1+\alpha)\cdots(N-1+\alpha) = \frac{\Gamma(N+\alpha)}{\Gamma(\alpha)}$$

11.2.3.2.2 General case

$$P_s(\mathcal{D}) = \prod_i\prod_j\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\alpha_{ijk}}$$

Where $i \in \{1, \ldots, n\}$ ranges over nodes in the networks, $j \in \{1, q_i\}$ ranges over configurations of X_i's parents and $k \in \{1, r_i\}$ ranges over states of X_i. This score is decomposable: it is the product of independent scores associated with the distribution of each node in the net.

11.2.3.3 Search strategy

The search strategy is NP-hard for nets whose nodes have at most $k > 1$ parents. To solve it heuristic strategies are employed. Because the search space is a set of DAGs with operations of adding, removing and reversing one arc, with an initial structure random or fully disconnected, strategies like hill climbing, best first and simulated annealing are employed. Decomposable scores allow to recompute local scores only for a single move.

Chapter 12

Naive Bayes classifier

12.1 Setting

Each instance x is described by a conjunction of attribute values $\langle a_1, \ldots, a_m \rangle$. The target function can take any value from a finite set of $\mathcal{Y}$. The task is predicting the MAP target value given the instance:

$$y^* = \arg\max_{y_i \in \mathcal{Y}} P(y_i|x) = \arg\max_{y_i \in \dagger} \frac{P(a_1, \ldots, a_m|y_i)p(y_i)}{P(a_1, \ldots, a_m)} = \\ = \arg\max_{y_i \in \mathcal{Y}} P(a_1, \ldots, a_m|y_i)P(y_i)$$

12.1.1 Learning problem

Class conditional probabilities $P(a_1, \ldots, a_m|y_i)$ are hard to learn, as the number of terms is equal to the number of possible instances times the number of target values. To simplify it attribute values are assumed independent of each other given the target value:

$$P(a_1, \ldots, a_m|y_i) = \prod_{j=1}^{m} P(a_j|y_i)$$

Parameters to be learned reduce to the number of possible attribute values time the number of possible target values.

12.2 Definition

$$y^* = \arg\max_{y_i \in \mathcal{Y}} \prod_{j=1}^{m} P(a_j|y_i)P(y_i)$$

12.2.1 Single distribution case

Assuming that all attribute values come from the same distribution, the probability of an attribute value given the class can be modelled as a multinomial distribution over the K possible values:

$$P(a_j|y_i) = \prod_{k=1}^{K} \theta_{ky_i}^{z_k(a_j)}$$

12.3 Parameters learning

Target prior $P(y_i)$ can be learned as the fraction of training set instances having each target value. The maximum-likelihood estimate for the parameter θ_{kc}, the probability of value v_k given class c is the fraction of times the value was observed in training examples of class c: $\theta_{kx} = \frac{N_{kn}}{N_c}$. The posterior distribution for attribute parameters is multinomial:

$$\theta_{kc} = \frac{N_{kc} + \alpha_{kc}}{N_c + \alpha_c}$$

12.4 Examples

12.4.1 Text classification

Classify documents in one of C possible classes. Each document is represented as the bag-of-words it contains. Let V be the vocabulary of all possible words and that a dataset of labelled document $\mathcal{D}$ is available.

12.4.1.1 Naive Bayes learning

Compute prior probabilities of classes as $P(y_i) = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$. Model the attributes with a multinomial distribution with $K = |V|$ possible states. Compute the probability of word w_k given class c as the fraction of times the word appear in documents of class y_i with respect to all words in document of class c:

$$\theta_{kc} = \frac{\sum_{\vec{x} \in \mathcal{D}_x} \sum_{j=1}^{|\vec{x}|} z_k(x[j])}{\sum_{\vec{x} \in \mathcal{D}_c} |\vec{x}}$$

12.4.1.2 Naive Bayes classification

$$\begin{aligned} y^* &= \arg\max_{y_i \in \mathcal{Y}} \prod_{j=1}^{|\vec{x}|} P(x[j]|y_i) P(y_i) = \\ &= \arg\max_{y_i \in \mathcal{Y}} \prod_{j=1}^{|\vec{x}|} \prod_{k=1}^{K} \theta_{ky_i}^{z_k(x[j])} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \end{aligned}$$

The assumption is that all attribute values come from the same distribution, otherwise attributes from different distributions have to be considered separately for parameter estimation.

12.4.1.3 Parameter estimation

Assume each instance x is represented as a vector of l attributes. Assume the j^{th} attribute can take $\{v_{j1}, \ldots, v_{jK_j}\}$ possible values. The parameter θ_{jkc} representing the probability of observing value v_{jk} for the j^{th} attribute given class c is estimated as:

$$\theta_{jkc} = \frac{\sum_{\vec{x} \in \mathcal{D}_c} z_{jk}(x[j])}{|\mathcal{D}_c|}$$

Chapter 13

Linear discriminant functions

13.1 Discriminative learning

If generative learning assumes knowledge of the distribution governing the data, discriminant learning focuses on directly modelling the discriminant function. For classification it directly models decision boundaries rather than inferring them from the modelled data distributions.

13.1.1 Pros of discriminative learning

When data is complex, modelling their distribution can be very difficult. If data discrimination is the goal, data distribution modelling is not needed. In this way the parameters are focused and thus the use of available training examples on the desired goal.

13.1.2 Cons of discriminative learning

The learned model is less flexible in its usage and it does not allow to perform arbitrary inference task. For example it is not possible to efficiently generate new data from a certain class.

13.2 Linear discriminant functions

13.2.1 Description

$$f(\vec{x}) = \vec{w}^T \vec{x} + w_0$$

The discriminant function is a linear combination of example features. w_0 is the bias or threshold. This is the simplest possible discriminant function and depending on the complexity of the task and the amount of data it can be the best option available.

13.2.2 Linear binary classifier

$$f(\vec{x}) = sign(\vec{w}^T \vec{x} + w_0)$$

The linear binary classifier is obtained taking the sign of the linear function. The decision boundary $f(\vec{x}) = 0$ is a hyperplane H. The weight vector $\vec{w}$ is orthogonal to the decision hyperplane:

$$\begin{aligned} \forall \vec{x}, \vec{x}' : f(\vec{x}) = f(\vec{x}') = 0 \\ \vec{w}^T \vec{x} + w_0 - \vec{w}^T \vec{x}' - w_o = 0 \\ \vec{w}^T (\vec{x} - \vec{x}') = 0 \end{aligned}$$

13.2.2.1 Functional margin

The value $f(\vec{x})$ of the function for a certain point $\vec{x}$ is called functional margin and can be seen as a confidence in the prediction.

13.2.2.2 Geometric margin

The distance from $\vec{x}$ to the hyperplane is the geometric margin: $r^x = \frac{f(\vec{x})}{||\vec{w}||}$. It is a normalized version of the functional margin. The distance from the origin to the hyperplane is:

$$r^0 = \frac{f(\vec{0})}{||\vec{w}||} = \frac{w_0}{||\vec{w}||}$$

A point can be expressed by its projection on H plus its distance to H times the unit vector in that direction:

$$\vec{x} = \vec{x}^p + r^x \frac{\vec{w}}{||\vec{w}||}$$

Then, as $f(\vec{x}^p) = 0$:

$$\begin{aligned} f(\vec{x}) &= \vec{w}^T \vec{x} + w_0 = \\ &= \vec{w}^T \left(\vec{x}^p + r^x \frac{\vec{w}}{||\vec{w}||} \right) + w_0 = \\ &= \underbrace{\vec{w}^T \vec{x}^p + w_0}_{f(\vec{x}^p)} + r^x \vec{w}^T \frac{\vec{w}}{||\vec{w}||} = \\ &= r^x ||\vec{w}|| = \\ \frac{f(\vec{x})}{||\vec{w}||} &= r^x \end{aligned}$$

13.3 Perceptron

The perceptron is a linear combination of input features with a threshold activation function. It is a single neuron architecture.

13.3.1 Biological motivation

The human brain is composed of densely interconnected networks of neurons. A neuron is made of a soma (central body containing the nucleus), dendrites (set of filaments departing from the body), an axon (a longer filament) and synapses (connections between dendrites and axons from other

neurons. Electrochemical reactions allow signals to propagate along neurons via axons, synapses and dendrites. Synapses can excite or inhibit a neuron potential and once a neuron potential exceeds a certain threshold, a signal is generated and transmitted along the axon.

13.3.2 Representational power

The perceptron can represent any linearly separable sets of examples like primitive boolean functions, so any logic formula can be represented by a network of two levels of perceptrons in disjunctive or conjunctive normal form. Non-linearly separable sets of examples cannot be separated and representing complex formulas can require a number of perceptrons exponential in the size of the input.

13.3.3 Augmented feature or weight vectors

The augmented feature vector is the weight vector incorporated with the bias: $\hat{\vec{w}} = \begin{pmatrix} w_0 \\ \vec{w} \end{pmatrix}$ and $\hat{\vec{x}} = \begin{pmatrix} 1 \\ \vec{x} \end{pmatrix}$ the search for weight vector and bias is replaced by the search for the augmented weight vector.

13.3.4 Parameter learning

13.3.4.1 Error minimization

There is a need to find a function of the parameters to be optimized. A reasonable function is the measure of error on the training set $\mathcal{D}$;

$$E(\vec{w}; \mathcal{D}) = \sum_{(\vec{x},y)\in\mathcal{D}} l(y, f(\vec{x}))$$

There is the problem of overfitting on the training data.

13.3.4.2 Gradient descent

Gradient descent is the technique by which the error is minimized:

1. Initialize $\vec{w}$, for example $\vec{w} = \vec{0}$.
2. Iterate until gradient is approximately zero: $\vec{w} = \vec{w} - \eta \nabla E(\vec{w}; \mathcal{D})$

η is the learning rate and controls the amount of movement at each gradient step. The algorithm is guaranteed to converge to a local optimum of $E(\vec{w}; \mathcal{D})$ for small enough η. Too low η implies slow convergence and there are techniques to adaptively modify it. The misclassification loss is piecewise constant so it's a poor candidate for gradient descent.

13.3.5 Perceptron training rule

$$E(\vec{w}; \mathcal{D}) = \sum_{(\vec{x},y)\in\mathcal{D}_E} -yf(\vec{x})$$

$\mathcal{D}_E$ is the set of current training errors for which $yf(\vec{x}) \leq 0$. The error is the sum of the functional margins of incorrectly classified examples. The error gradient is:

$$\nabla E(\vec{w};\mathcal{D}) = \nabla \sum_{(\vec{x}ly)\in\mathcal{D}_E} -yf(\vec{x}) =$$
$$= \nabla \sum_{(\vec{x},y)\in\mathcal{D}_E} -y(\vec{w}^T\vec{x}) =$$
$$= \sum_{(\vec{x},y)\in\mathcal{D}_E} -y\vec{x}$$

The amount to update is:

$$-\eta\nabla E(\vec{w};\mathcal{D} = \eta \sum_{(\vec{x},y)\in\mathcal{D}_E} y\vec{x}$$

13.3.6 Stochastic perceptron training rule

1. Initialize weights randomly.
2. Iterate until all examples are correctly classified: for each incorrectly classified training example $(\vec{x}, y)$ update the weight vector $\vec{w} \leftarrow \vec{w} + \eta y\vec{x}$.

The gradient step for each training error is done instead of the some of them in batch learning. Each gradient step is very fast and stochasticity can sometimes help to avoid local minima, being guided by various gradients for each training example.

13.3.7 Perceptron regression

13.3.7.1 Exact solution

Let $X \in \mathbb{R}^n \times \mathbb{R}^d$ be the input training matrix. Let $\vec{y} \in \mathbb{R}^n$ be the output training matrix, where y_i is the output for example $\vec{x}^{(i)}$. Regression learning could be stated as a set of linear equations:

$$X\vec{w} = \vec{y}$$

Giving solution $\vec{w} = X^{-1}\vec{y}$. Matrix X is rectangular and so not invertible, with more rows than columns. The system of equations is overdetermined, with more equations than unknowns and no exact solution typically exists.

13.3.7.2 Mean squared error

Because no exact solution exists the mean squared error MSE is used, resorting to loss minimization. The standard loss for regression is the mean squared error:

$$E(\vec{w};\mathcal{D}) = \sum_{(\vec{x},y)\in\mathcal{D}} (y - f(\vec{x}))^2 = (\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$$

A closed form solution exists and can always be solved by gradient descent, that can be faster. Mean squared error can also be used as a classification loss.

13.3.7.2.1 Closed form solution

$$\begin{aligned}\nabla E(\vec{w};\mathcal{D}) &= \nabla(\vec{y}-X\vec{w})^T(\vec{y}-X\vec{w}) = \\ &= 2(\vec{y}-X\vec{w})^T(-X) = 0 \\ &= -2\vec{y}^TX + 2\vec{w}^TX^TX = 0 \\ \vec{w}^TX^TX &= \vec{y}^TX \\ X^TX\vec{w} &= X^T\vec{y} \\ \vec{w} &= (X^TX)^{-1}X^T\vec{y}\end{aligned}$$

$(X^TX)^{-1}X^T$ is called left inverse. If X is square and nonsingular, inverse and left-inverse coincide and the MSE solution correspond to the exact one. The left inverse exists provided $(X^TX) \in \mathbb{R}^{d\times d}$ is full rank: this happens if features are linearly independent. If not the redundant one can be removed.

13.3.7.2.2 Gradient descent

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{(\vec{x},y)\in\mathcal{D}}(y-f(\vec{x}))^2 = \\ &= \frac{1}{2}\sum_{(\vec{x},y)\in\mathcal{D}}\frac{\partial}{\partial w_i}(y-f(\vec{x}))^2 = \\ &= \frac{1}{2}\sum_{(\vec{x},y)\in\mathcal{D}}2(y-f(\vec{x}))\frac{\partial}{\partial w_i}(y-\vec{w}^t\vec{x}) = \\ &= \sum_{(\vec{x},y)\in\mathcal{D}}(y-f(\vec{x}))(-x_i)\end{aligned}$$

13.4 Multiclass classification

13.4.1 One-vs-all

In the one-vs-all approach a binary classifier is learned for each class: positive examples are examples of the class, negative ones are examples of all other classes. A new example is predicted in the class with maximum functional margin. Decision boundaries for which $f_i(\vec{x}) = f_j(\vec{x})$ are pieces of hyperplanes so that:

$$\begin{aligned}\vec{w}_i^T\vec{x} &= \vec{w}_j^T\vec{x} \\ (\vec{w}_i-\vec{w}_j)^T\vec{x} &= 0\end{aligned}$$

13.4.2 All-pairs

In the all-pairs approach a binary classifier is learned for each pair of classes: positive examples are from one class and negative ones are from the other. A new example is predicted in the class winning the largest number of pairwise classifications.

13.5 Generative linear classifiers

13.5.1 Gaussian distributions

Linear decision boundaries are obtained when covariance is shared among classes: $\Sigma_i = \Sigma$.

13.5.2 Naive Bayes classifier

$$f_i(\vec{x}) = P(\vec{x}|y_i)P(y_i) = \prod_{j=1}^{|\vec{x}|}\prod_{k=1}^{K}\theta_{ky_i}^{z_k(x[j])}\frac{|\mathcal{D}_i|}{|\mathcal{D}|} = \\ = \prod_{k=1}^{K}\theta_{ky_i}^{N_k\vec{x}}\frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

Where $N_{k\vec{x}}$ is the number of times feature k appears in $\vec{x}$. Naive Bayes is a log-linear model as Gaussian distributions with shared Σ:

$$\log f_i(\vec{x}) = \underbrace{\sum_{k=1}^{K} N_{k\vec{x}}\log\theta_{ky_i}}_{\vec{w}^T\vec{x}'} + \underbrace{\log\left(\frac{|\mathcal{D}_i|}{|\mathcal{D}|}\right)}_{w_0}$$

Where $\vec{x}' = [N_{1\vec{x}}, \ldots, N_{K\vec{x}}]^T$ and $\vec{w} = [\log\theta_{1y_i}, \ldots, \log\theta_{Ky_i}]^T$.

Chapter 14

Support vector machines

14.1 Introduction

Support vector machines are linear classifiers selecting hyperplane maximizing separation margin between classes or are large margin classifiers. The solution depends only on a small subset of training examples or the support vectors. They have a sound generalization theory with bounds or error based on the margin and they can easily extended to non-linear separation by the kernel machines.

14.2 Maximum margin classifiers

14.2.1 Classifier margin

Given a training set $\mathcal{D}$, a classifier confidence margin is:

$$\rho = \min_{(\vec{x},y)\in\mathcal{D}} yf(\vec{x})$$

It is the minimal confidence margin for predicting the true label among training examples. A classifier geometric margin is:

$$\frac{\rho}{||\vec{w}||} = \min_{(\vec{x},y)\in\mathcal{D}} \frac{yf(\vec{x})}{||\vec{w}||}$$

14.2.2 Canonical hyperplane

There is an infinite number of equivalent formulation for the same hyperplane:

$$\begin{aligned} \vec{w}^T\vec{x} + w_0 &= 0 \\ \alpha(\vec{w}^T\vec{x} + w_0) &= 0 \qquad \forall\alpha \neq 0 \end{aligned}$$

The canonical hyperplane is the hyperplane having confidence margin equal to 1: $\rho = \min_{(\vec{x},y)\in\mathcal{D}} yf(\vec{x}) = 1$. Its geometric margin is:

$$\frac{\rho}{||\vec{w}||} = \frac{1}{||\vec{w}||}$$

14.3 Hard margin support vector machine

14.3.1 Margin error bound theorem

Consider the set of decision functions $f(\vec{x}) = sign(\vec{w}^T\vec{x})$ with $||\vec{w}|| \leq \Lambda$ and $||\vec{x}|| \leq R$ for some $R, \Lambda > 0$. Moreover let $\rho > 0$ and ν denote the fraction of training examples with margin smaller than $\frac{\rho}{||\vec{w}||}$, referred as the margin error. For all distributions P generating the data, with probability at least $1 - \delta$ over the drawing of the m training patterns, and for any $\rho > 0$ and $\delta \in]0; 1[$, the probability that a test pattern drawn from P will be misclassified is bound from above by:

$$\nu + \sqrt{\frac{c}{m}\left(\frac{R^2\Lambda^2}{\rho^2}\ln^2 m + \ln\frac{1}{\delta}\right)}$$

Where c is an universal constant.

14.3.1.1 Interpretation of the margin error bound

The probability of the test error depends on the number of margin errors ν, examples with margin smaller than $\frac{\rho}{||\vec{w}||}$, the number of training examples, $\sqrt{\frac{ln^2 m}{m}}$, the size of the margin $\frac{1}{\rho^2}$.

14.3.2 Learning problem

If ρ is fixed to 1, maximizing the margin corresponds to minimizing $||\vec{w}||$. The learning problem becomes: $\min_{\vec{w},w_0} \frac{1}{2}||\vec{w}||^2$ subject to $y_i(\vec{w}^T\vec{x}_i + w_0) \geq 1 \forall (\vec{x}_i, y_i) \in \mathcal{D}$. The constraints guarantee that all points are correctly classified. The minimization corresponds to maximizing the squared margin. It is a quadratic optimization problem: the objective is quadratic and the points satisfying the constraints form a convex set.

14.3.3 Constrained optimization - Karush-Kuhn-Tucker (KKT) approach

Following the KKT approach a constrained optimization problem can be addressed by converting it into an unconstrained problem with the same solution. Let's consider a constrained optimization problem as $\min_z f(z)$ subject to $g_i(z) \geq 0 \forall i$. Let's introduce a non-negative variable $\alpha_i \geq 0$ or a Lagrange multiplier for each constraint and rewrite the optimization problem as Lagrangian:

$$\min_z \max_{\vec{\alpha} \geq 0} f(z) - \sum_i \alpha_i g_i(z)$$

The optimal solutions z^* for this problem are the same as the optimal solutions for the original problem: if for a given z' at least one constraint is not satisfied, maximizing over α_i leads to an infinite value. If all constraints are satisfied, maximization over the α will set all elements of the summation to zero, so that z' is a solution of $\min_z f(z)$.

14.3.4 KKT approach in SVM

The constraint can be included in the minimization using $m = |\mathcal{D}|$ Lagrange multipliers $\alpha_i \geq 0$

$$L(\vec{w}, w_0, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\vec{w}^T \vec{x}_i + w_0) - 1)$$

The Lagrangian is minimized with respect to $\vec{w}$, w_0 and maximized with respect to α_i. The solution is a saddle point.

14.3.4.1 Dual formulation

Vanishing derivatives with respect to primal variables we get:

$$\frac{\partial}{\partial w_0} L(\vec{w}, w_0, \vec{\alpha}) = 0 \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0$$

$$\frac{\partial}{\partial \vec{w}} L(\vec{w}, w_0, \vec{\alpha}) = 0 \Rightarrow \vec{w} = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i$$

So, substituting in the Lagrangian:

$$\begin{aligned} &\frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\vec{w}^T \vec{x}_i + w_0) - 1) = \\ = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j - \underbrace{\sum_{i=1}^{m} \alpha_i y_i}_{=0} w_0 + \sum_{i=1}^{m} \alpha_i = \\ = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j = L(\alpha) \end{aligned}$$

Which is maximized with respect to the dual variables α. The resulting maximization problem including the constraints is $\max\limits_{\alpha \in \mathbb{R}^m} \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$ subject to $\alpha_i \geq 0 \forall i \in \{1, dots, m\}$ and $\sum\limits_{i=1}^{m} \alpha_i y_i = 0$. It is still a quadratic optimization problem. The dual formulation has simpler constraints and it's easier to solve. The primal formulation has $d + 1$ variables: $\min\limits_{\vec{w}, w_0} \frac{1}{2}||\vec{w}||^2$. The dual formulation has m variables, the number of training examples: $\max\limits_{\alpha \in \mathbb{R}^m} \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$. One can choose the primal formulation if it has much less variables.

14.3.5 Decision function

Substituting $\vec{w} = \sum\limits_{i=1}^{m} \alpha_i y_i \vec{x}_i$ in the decision function:

$$f(\vec{x}) = \vec{w}^T \vec{x} + w_o = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i^T \vec{x} + w_0$$

The decision function is a linear combination of dot products between training points and the test point. The dot product is a kind of similarly between points. Weights of the combination are $\alpha_i y_i$: large α_i implies large contribution towards class y_i times the similarity.

14.3.6 KKT conditions

$$L(\vec{w}, w_o, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\vec{w}^T \vec{x}_i + w_0) - 1)$$

At the saddle point it holds that for all i:

$$\alpha_i (y_i(\vec{w}^T \vec{x}_i + w_0) - 1) = 0$$

Thus, either the example does not contribute to the final $f(\vec{x})$ if $\alpha_i = 0$ or the example stays on the minimal confidence hyperplane from the decision one: $y_i(\vec{w}^T \vec{x}_i + w_0) = 1$.

14.3.7 Support vectors

Support vectors are points staying on the minimal confidence hyperplanes. All other points do not contribute to the final decision function and they could be removed from the training set. SVM are typically sparse because they have few support vectors.

14.3.8 Decision function bias

The bias w_0 can be computed from the KKT conditions. Given an arbitrary support vector $\vec{x}_i$ with $\alpha_i > 0$, the KKT condition implies:

$$\begin{aligned} y_i(\vec{w}^T \vec{x}_i + w_0) &= 1 \\ y_i \vec{w}^T \vec{x}_i + y_i w_0 &= 1 \\ w_0 &= \frac{1 - y_i \vec{w}^T \vec{x}_i}{y_i} \end{aligned}$$

For robustness, the bias is usually averaged over all support vectors.

14.4 Soft margin SVM

14.4.1 Slack variables

A slack variable ξ_i represents the penalty for example x_i not satisfying the margin constraint. The sum of the slacks is minimized together to the inverse margin. The regularization parameter $C \geq 0$ trades-off data fitting and size of the margin.

$$\min_{\vec{w} \in \mathcal{X}, w_0 \in \mathbb{R}, \vec{\xi} \in \mathbb{R}^m} \frac{1}{2}||vecw||^2 + C \sum_{i=1}^{m} \xi_i$$

Subject to $y_i(\vec{w}^T \vec{x}_i + w_0) \geq 1 - \xi_i \forall i \in \{1, \ldots, m\}$ and $\xi_i \geq 0 \forall i \in \{1, \ldots\}$.

14.4.2 Regularization theory

$$\min_{\vec{w}\in\mathcal{X},w_0\in\mathbb{R},\vec{\xi}\in\mathbb{R}^m} \frac{1}{2}||vecw||^2 + C\sum_{i=1}^{m} l(y_i, f(\vec{x}_i))$$

This is a regularized loss minimization problem. The loss term accounts for error minimization. The margin maximization term accounts for regularization: solution with larger margin are preferred. Regularization is a standard approach to prevent overfitting and it corresponds to a prior for simpler (more regular, smoother) solutions.

14.4.2.1 Hinge loss

$$l(y_i, f(\vec{x}_i)) = |1 - y_i f(\vec{x}_i)|_+ = |1 - y_i(\vec{w}^T\vec{x}_i + w_0)|_+$$

$|z|_+ = z$ if $z > 0$ and 0 otherwise and it corresponds to the slack variable ξ_i. All examples not violating the margin constraint have zero loss so there is a sparse set of support vectors.

14.4.3 Lagrangian

$$L = C\sum_{i=1}^{m}\xi_i + \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{m}\alpha_i(y_i(\vec{w}^T\vec{x}_i + w_0) - 1 + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i$$

Where $\alpha_i, \beta_i \geq 0$.
Vanishing derivatives with respect to primal variables we get:

$$\frac{\partial}{\partial w_0}L = 0 \Rightarrow \sum_{i=1}^{m}\alpha_i y_i = 0$$

$$\frac{\partial}{\partial \vec{w}}L = 0 \Rightarrow \vec{w} = \sum_{i=1}^{m}\alpha_i y_i \vec{x}_i$$

$$\frac{\partial}{\partial \xi_i}L = 0 \Rightarrow C - \alpha_i - \beta_i = 0$$

14.4.4 Dual formulation

Substituting the Lagrangian:

$$C\sum_{i=1}^{m}\xi_i + \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{m}\alpha_i(y_i(\vec{w}^T\vec{x}_i + w_0) - 1 + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i =$$

$$\sum_{i=1}^{m}\xi_i\underbrace{(C - \alpha_i - \beta_i)}_{=0} + \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j - \sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j - \underbrace{\sum_{i=1}^{m}\alpha_i y_i}_{=0} w_0 + \sum_{i=1}^{m}\alpha_i =$$

$$\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j = L(\alpha)$$

The box constraint for α_i comes from $C - \alpha_i - \beta_i = 0$ and the fact that both $\alpha_i \geq 0$ and $\beta_i \geq 0$.

$$\max_{\alpha \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$

Subject to $0 \geq \alpha_i \geq C \forall i \in \{1, \ldots, m\}$ and $\sum_{i=1}^{m} \alpha) iy) i = 0$.

14.4.5 Karush-Khun-Tucker conditions

At the saddle point it holds that for all i $\alpha_i(y_i(\vec{w}^T \vec{x}_i + w_0) - 1 + \xi_i) = 0$ and $\beta_i \xi_i = 0$, thus support vectors $\alpha_i > 0$ are examples for which $(y_i(\vec{w}^T \vec{x}_i + w_0) \leq 1$.

14.4.6 Support vectors

If $\alpha_i < C$, $C - \alpha_i - \beta_i = 0$ and $\beta_i \xi_i = 0$ imply that $\xi_i = 0$. These are the unbound support vectors $(y_i(\vec{w}^T \vec{x}_i + w_0) = 1$ and they stay on the confidence one hyperplane. If $\alpha_i = C$ or bound support vectors then ξ_i can be greater than zero, in which case the support vectors are margin errors.

14.5 Large-scale SVM learning

14.5.1 Stochastic gradient descent

$$\min_{\vec{w} \in \mathcal{X}} \frac{\lambda}{2} ||\vec{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} |1 - y_i \langle \vec{w}, \vec{x}_i \rangle|_+$$

The objective for a single example $(\vec{x}_i, y_i)$:

$$E(\vec{w}; (\vec{x}_i, y_i)) = \frac{\lambda}{2} ||\vec{w}||^2 + |1 - y_i \langle \vec{w}, \vec{x}_i \rangle|_+$$

The subgradient is:

$$\nabla_{\vec{w}} E(\vec{w}; (\vec{x}_i, y_i)) = \lambda \vec{w} - \mathbb{1}[y_i \langle \vec{w}, \vec{x}_i \rangle < 1] y_i \vec{x}_i$$

The indicator function is such that:

$$\mathbb{1}[y_i \langle \vec{w}, \vec{x}_i \rangle < 1] = \begin{cases} 1 & if\ y_i \langle \vec{w}, \vec{x}_i \rangle < 1 \\ 0 & otherwise \end{cases}$$

The subgradient of a function f at a point x_0 is any vector $\vec{v}$ such that for any x:

$$f(x) - f(x_0) \geq \vec{v}^T (x - x_0)$$

14.5.1.1 Pseudocode - pegasus

1. Initialize $\vec{w}_1 = \vec{0}$.
2. For $t = 1$ to T:
   (a) Randomly choose $(\vec{x}_{i_t}, t_{i_t})$ from $\mathcal{D}$.
   (b) Set $\eta_t = \frac{1}{\lambda t}$.

(c) Update $\vec{w}$:

$$\vec{w}_{t+1} = \vec{w}_t - \eta_t \nabla_{\vec{w}} E(\vec{w}; (\vec{x}_{i_t}, y_{i_t}))$$

3. Return $\vec{w}_{T+1}$.

The choice of the learning rate allows to bound the runtime for and ϵ-accurate solution to $O\left(\frac{d}{\lambda\epsilon}\right)$ with d maximum number of non-zero features in an example.

14.5.2 Dual version

It is easy to show that:

$$\vec{w}_{t+1} = \frac{1}{\lambda t} \sum_{i=1}^{t} \mathbb{1}[y_{i_t} \langle \vec{w}_t, \vec{x}_{i_t} \rangle < 1] y_{i_t} \vec{x}_{i_t}$$

$\vec{w}_{t+1}$ can be represented implicitly by storing in vector α_{t+1} the number of times each example was selected and had a non-zero loss:

$$\alpha_{t+1}[j] = |\{t' \leq t : i_{t'} = j \wedge y_j \langle \vec{w}_{t'}, \vec{x}_j \rangle < 1\}|$$

14.5.2.1 Pseudocode - pegasus dual

1. Initialize $\vec{\alpha}_1 = 0$.
2. For $t = 1$ to T:
   (a) Randomly choose $(\vec{x}_{i_t}, y_{i_t})$ from $\mathcal{D}$.
   (b) Set $\alpha_{t+1} = \alpha_t$.
   (c) If $y_{i_t} \frac{1}{\lambda t} \sum_{j=1}^{t} \alpha_t[j] y_j \langle \vec{x}_j, \vec{x}_{i_t} \rangle < 1$ then:

$$\alpha_{t+1}[i_t] = \alpha_{t+1}[i_t] + 1$$

3. Return α_{T+1}.

This will be useful when combined with kernels

Chapter 15

Non linear support vector machines

15.1 Non-linearly separable problems

Hard-margin SVM can address linearly separable problems and soft-margin SVM can address linearly separable problems with outliers. Non-linearly separable problems need a higher expressive power with more complex feature combinations, without loosing the advantages of linear separators like a large margin and theoretical guarantees. To do so the input examples are mapped into a higher dimensional feature space and the linear classification is performed in the higher dimensional space.

15.2 Non-linear support vector machines - feature map

A feature map is a function mapping each example to a higher dimensional space: $\Phi : \mathcal{X} \to \mathcal{H}$. Examples $\vec{x}$ are replaced with their feature mapping $\Phi(\vec{x})$. The feature mapping should increase the expressive power of the representation introducing features which are combination of input features and examples should be approximately linearly separable in the mapped space. Feature mapping can be homogeneous:

$$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Or inhomogeneous:

$$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

15.2.1 Polynomial mapping

Polynomial mapping maps feature to all possible conjunctions (products) of feature. If it homogeneous it maps them to a certain degree d, if it is inhomogeneous it maps them up to a certain degree.

15.2.2 Linear separation in feature space

A SVM algorithm is applied replacing $\vec{x}$ with $\Phi(\vec{x})$:

$$f(\vec{x}) = \vec{w}^T \Phi(\vec{x}) + w_0$$

A linear separation in the feature space corresponds to a non linear separation in the input space, for example:

$$f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = sgn(w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + w_0)$$

15.3 Support vector regression

Support vector regression retains combination of regularization and data fitting. Regularization means smoothness of the learned function, entailing smaller weights or lower complexity. A sparsifying loss can be used to have few support vectors.

15.3.1 ϵ-insensitive loss

$$l(f(\vec{x}), y) = |y - f(\vec{x})|_\epsilon = \begin{cases} 0 & if\ |y - f(\vec{x})| \leq \epsilon \\ |y - f(\vec{x})| - \epsilon & otherwise \end{cases}$$

The ϵ-insensitive loss allow to tolerate small ϵ deviation from the true value with no penalty. It defines a ϵ-tube of insensitiveness around true values. This also allows to trade off function complexity with data fitting playing with the ϵ value.

15.3.2 Optimization problem

$$\min_{\vec{w} \in \mathcal{X}, w_0 \in \mathbb{R}, \vec{\xi}, \vec{\xi}^* \in \mathbb{R}^m} \frac{1}{2}||\vec{w}||^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*)$$

Subject to $\vec{w}^T \Phi(\vec{x}_i) + w_0 - y_i \leq \epsilon + \xi_i$, $y_i - (\vec{w}^T \phi(\vec{x}_i) + w_0) \leq \epsilon + \xi_i^*$ and $\xi_i, \xi_i^* \geq 0$. There are two constraints for each example for the upper and lower sides of the tube. Slack variables ξ_i and ξ_i^* penalize predictions out of the ϵ-insensitive tube.

15.3.3 Lagrangian

The constraints are included in the minimization function using Lagrange multipliers $\alpha_i, \alpha_i^*, \beta_i, \beta_i^* \geq 0$:

$$L = \frac{1}{2}||\vec{w}||^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) - \sum_{i=1}^{m} (\beta_i \xi_i + \beta_i^* \xi_i^*) - \sum_{i=1}^{m} \alpha_i (\epsilon + \xi_i + y_i - \vec{w}^T \Phi(\vec{x}_i) - w_0) - \sum_{i=1}^{m} \alpha_i^* (\epsilon + \xi_i^* - y_i + \vec{w}^T \Phi(\vec{x}_i) + w_0)$$

15.3.3.1 Vanishing the derivatives with respect to the primal variables

Vanishing the derivatives with respect to the primal variables:

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)\Phi(\vec{x}_i) = 0 \rightarrow \vec{w} = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)\Phi(\vec{x}_i)$$

$$\frac{\partial L}{\partial w_0} = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \rightarrow \alpha_i \in [0, C]$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \beta_i^* = 0 \rightarrow \alpha_i^* \in [0, C]$$

15.3.4 Dual formulation

Substituting them in the Lagrangian:

$$\underbrace{\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\Phi(\vec{x}_i)^T\Phi(\vec{x}_j)}_{||\vec{w}||^2} + \sum_{i=1}^{m}\xi_i\underbrace{(C - \beta_i - \alpha_i)}_{=0} +$$
$$+\sum_{i=1}^{m}\xi_i^*\underbrace{(C - \beta_i^* - \alpha_i^*)}_{=0} - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m}y_i(\alpha_i^* - \alpha_i) +$$
$$+w_0\underbrace{\sum_{i=1}^{m}(\alpha_i - \alpha_i^*)}_{=0} - \sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\Phi(\vec{x}_i)^T\Phi(\vec{x}_j)$$

So the problem becomes to optimize:

$$\max_{\alpha\in\mathbb{R}^m} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\Phi(\vec{x}_i)^T\Phi(\vec{x}_j) - \epsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m}y_i(\alpha_i^* - \alpha_i)$$

Subject to $\sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0$ with $\alpha_i, \alpha_i^* \in [0, C] \forall i \in [1, m]$.

15.3.5 Regression function

$$f(\vec{x}) = \vec{w}^T\Phi(\vec{x}) + w_0 = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\Phi(\vec{x}_i)^T\Phi(\vec{x}) + w_0$$

15.3.6 KKT conditions

At the saddle point holds that for all i:

$$\begin{aligned}\alpha_i(\epsilon + \xi_i + y_i - \vec{w}^T\Phi(\vec{x}_i) - w_0) &= 0\\ \alpha_i^*(\epsilon + \xi_i^* + y_i - \vec{w}^T\Phi(\vec{x}_i) - w_0) &= 0\\ \beta_i\xi_i &= 0\\ \beta_i^*\xi_i^* &= 0\end{aligned}$$

Combined with $C - \alpha_i - \beta_i = 0$, $\alpha_i \geq 0$, $\beta_i \geq 0$, $C - \alpha_i^* - \beta_i^* = 0$, $\alpha_i^* \geq 0$, $\beta_i^* \geq 0$:

$$\begin{aligned}\alpha_i \in [0, C] \qquad & \alpha_i^* \in [0, C]\\ \alpha_i = C \; if \; \xi_i > 0 \qquad & \alpha_i^* = C \; if \; \xi_i^* > 0\end{aligned}$$

15.3.7 Support vectors

All patterns within the ϵ-tube, for which $|f(\vec{x})i - y)i| < \epsilon$ have $\alpha_i, \alpha_i^* = 0$ and don't contribute to the estimated function f. Pattern for which either $0 < \alpha_i < C$ or $0 < \alpha_i^* < C$ are on the border of the ϵ-tube: $|f(\vec{x}_i) - y_i| = \epsilon$ and are the unbound support vectors. The remaining patterns are margin errors: $\xi_i > 0 \vee \xi_i^* > 0$ and reside out of the ϵ-insensitive region and they are bound support vectors with corresponding $\alpha_i = C \vee \alpha_i^* = C$.

15.4 Smallest enclosing hypersphere

The smallest enclosing hypersphere characterize a set of examples defining boundaries enclosing them. The objective is to find the smallest hypersphere in the feature space enclosing the data points accounting for outliers paying a cost for leaving examples out of the sphere. This is done for one-class classification: it models a class when no negative examples exists or for anomaly or novelty detection: it detects test data falling outside of the sphere and return them as novel or anomalous.

15.4.1 Optimization problem

$$\min_{R\in\mathbb{R},\vec{O}\in\mathcal{H},\vec{\xi}\in\mathbb{R}^m} R^2 + C\sum_{i=1}^{m}\xi_i$$

Subject to $||\phi(\vec{x}_i) - \vec{o}||^2 \leq R^2 + \xi_i \forall i \in \{1, \ldots, m\}$ and $\xi_i \geq 0 \forall i \in \{1, \ldots, m\}$. With $\vec{o}$ the centre of the sphere, R the minimized radius and slack variables $\vec{\xi}$ gather costs for outliers.

15.4.2 Lagrangian

Considering Lagrange multipliers $\alpha_i, \beta_i \geq 0$ the Lagrangian is:

$$L = R^2 + C\sum_{i=1}^{m} - \sum_{i=1}^{m}\alpha_i(R^2 + \xi_i - ||\Phi(\vec{x}_i) - \vec{o}||^2) - \sum_{i=1}^{m}\beta_i\xi_i$$

15.4.2.1 Vanishing the derivatives with respect to the primal variables

$$\frac{\partial L}{\partial R} = 2R\left(1 - \sum_1^m \alpha_i\right) = 0 \rightarrow \sum_{i=1}^m \alpha_i = 1$$

$$\frac{\partial L}{\partial \vec{o}} = 2\sum_{i=1}^m \alpha_i(\Phi(\vec{x}_i) - \vec{o})(-1) = 0 \rightarrow \vec{o}\underbrace{\sum_{i=1}^m \alpha_i}_{=1} = \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)$$

$$\frac{\partial L}{\partial \xi_i} = X - \alpha_i - \beta_i = 0 \rightarrow \alpha_i \in [0, C]$$

15.4.3 Dual formulation

$$R^2 \underbrace{\left(1 - \sum_{i=1}^m \alpha_i\right)}_{=0} + \sum_{i=1}^m \xi_i \underbrace{(C - \alpha_i - \beta_i}_{=0} + \sum_{i=1}^m \alpha_i (\Phi(\vec{x}_i) - \underbrace{\sum_{j=1}^m \alpha_j \Phi(\vec{x}_j))}_{\vec{o}})^T (\Phi(\vec{x}_i) - \underbrace{\sum_{h=1}^m \alpha_h \Phi(\vec{x}_h))}_{\vec{0}}$$

So:

$$\sum_{i=1}^m \alpha_i (\Phi(\vec{x}_i) - \sum_{j=1}^m \alpha_j \Phi(\vec{x}_j))^T (\Phi(\vec{x}_i) - \sum_{h=1}^m \alpha_h \Phi(\vec{x}_h)) =$$

$$= \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \Phi(\vec{x}_i) - \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \sum_{h=1}^m \alpha_h \Phi(\vec{x}_h) - \sum_{i=1}^m \alpha_i \sum_{j=1}^m \alpha_j \Phi(\vec{x}_j)^T \Phi(\vec{x}_i) + \underbrace{\sum_{i=1}^m \alpha_i}_{=1} \sum_{j=1}^m \alpha_j \Phi(\vec{x}_j)^T \sum_{h=1}^m \alpha_h \Phi(\vec{x}_h) =$$

$$= \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \Phi(\vec{x}_i) - \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \sum_{j=1}^m \alpha_j \Phi(\vec{x}_j)$$

The optimization problem then becomes:

$$\max_{\vec{\alpha} \in \mathbb{R}^m} \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \Phi(\vec{x}_i) - \sum_{i,j=1}^m \alpha_i \alpha_j \Phi(\vec{x}_i)^T \Phi(\vec{x}_j)$$

Subject to $\sum_{i=1}^m \alpha_i = 1$, $0 \leq \alpha_i \leq C \forall i \in \{1, \ldots, m\}$.

15.4.4 Distance function

The distance of a point from the origin is:

$$\begin{aligned} R^2(\vec{x}) - ||\Phi(\vec{x} - \vec{o}||^2 \\ &= (\Phi(\vec{x} - \sum_{i=1}^m \alpha_i \Phi(\vec{x}_i))^T (\Phi(\vec{x}) - \sum_{j=1}^m \alpha_j \Phi(\vec{x}_j)) = \\ &= \Phi(\vec{x})^T \Phi(\vec{x}) - 2\sum_{i=1}^m \alpha_i \Phi(\vec{x}_i)^T \Phi(\vec{x}) + \sum_{i,j=1}^m \alpha_i \alpha_j \Phi(\vec{x}_i^T \Phi(\vec{x}_j) \end{aligned}$$

15.4.5 KKT conditions

At the saddle points it holds that for all i:

$$\beta_i \xi_i = 0$$
$$\alpha_i (R^2 + \xi_i - ||\Phi(\vec{x}_i) - \vec{o}||^2) = 0$$

15.4.6 Support vectors

Unbound support vectors $0 < \alpha_i < C$ have an image that lies on the surface of the enclosing sphere. Bound support vectors $\alpha_i = C$ have an image that lies outside of the enclosing sphere and correspond to outliers. All other points $\vec{\alpha} = \vec{0}$ have images inside the enclosing sphere.

15.4.7 Decision function

The radius R^* of the enclosing sphere can be computed using the distance function on any unbound support vector $\vec{x}$:

$$R^2(\vec{x}) = \Phi(\vec{x})^T \Phi(\vec{x}) - 2 \sum_{i=1}^{m} \alpha_i \Phi(\vec{x}_i)^T \Phi(\vec{x}) + \sum_{i,j=1}^{m} \alpha_i \alpha_j \Phi(\vec{x}_i)^T \Phi(\vec{x}_j)$$

A decision function for novelty detection could be:

$$f(\vec{x}) = sgn(R^2(\vec{x}) - (R^*)^2)$$

Positive if the examples lays outside the sphere and negative otherwise.

15.5 Support vector ranking

Support vector ranking order examples by relevance. It learns a scoring function predicting the quality of the example. It constraint the function to score $\vec{x}_i$ higher than $\vec{x}_j$ if it is more relevant doing pairwise comparison for training. It is easily formalized as a support vector classification task.

15.5.1 Optimization problem

$$\min_{\vec{w} \in \mathcal{X}, w_0 \in \mathbb{R}, \xi_{i,j} \in \mathbb{R}} \frac{1}{2} ||\vec{w}||^2 + C \sum_{i,j} \xi_{i,j}$$

Subject to $\vec{w}^T \Phi(\vec{x}_i) - \vec{w}^T \Phi(\vec{x}_j) \geq 1 - \xi_{i,j}$, $\xi_{i,j} \geq 0 \ \forall i, j : \vec{x}_i \prec \vec{x}_j$. There is one constraint for each pair of examples having ordering information and examples should be correctly ordered with a large margin.

15.5.2 Support vector classification on pairs

$$\min_{\vec{w} \in \mathcal{X}, w_0 \in \mathbb{R}, \xi_{i,j} \in \mathbb{R}} \frac{1}{2} ||\vec{w}||^2 + C \sum_{i,j} \xi_{i,j}$$

Subject to $y_{i,j} \vec{w}^T \underbrace{(\Phi(\vec{x}_i) - \Phi(\vec{x}_j))}_{\Phi(\vec{x}_{ij})} \geq 1 - \xi_{i,j}$, $\xi_{i,j} \geq 0 \ \forall i, j : \vec{x}_i \prec \vec{x}_j$. Where labels are always positive.

15.5.3 Decision function

The decision function is:

$$f(\vec{x}) = \vec{w}^T \Phi(\vec{x})$$

This is an unbiased standard support vector classification function and represents the score of the example for ranking it.

Chapter 16

Kernel machines

16.1 Kernel trick

Feature mapping $\Phi(\cdot)$ can be very high dimensional and so it can be highly expensive to explicitly compute it. Noting that feature mappings appear only in dot products in dual formulation, the kernel trick consists in replacing these dot products with an equivalent kernel function:

$$k(\vec{x}, \vec{x}') = \Phi(\vec{x})^T \Phi(\vec{x}')$$

The kernel function uses examples in the input space

16.1.1 Support vector classification

16.1.1.1 Dual optimization problem

$$\max_{\vec{\alpha} \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \underbrace{\Phi(\vec{x}_i)^T \Phi(\vec{x}_j)}_{k(\vec{x}_i, \vec{x}_j)}$$

Subject to $o \leq \alpha_i \leq C \forall i \in \{1, \ldots, m\}$ and $\sum_{i=1}^{m} \alpha_i y_i = 0$

16.1.1.2 Dual decision function

$$f(\vec{x}) = \sum_{i=1}^{m} \alpha_i y_i \underbrace{\Phi(\vec{x}_i)^T \Phi(\vec{x})}_{k(\vec{x}_i, \vec{x})}$$

16.1.2 Polynomial kernel

16.1.2.1 Homogeneous

$$k(\vec{x}, \vec{x}') = (\vec{x}^T \vec{x}')^d$$

With $d = 2$:

$$k\left(\begin{pmatrix}x_1\\x_2\end{pmatrix},\begin{pmatrix}x'_1\\x'_2\end{pmatrix}\right) = (x_1x'_1 + x_2x'_2)^2 =$$
$$= (x_1x'_1)^2 3 + (x_2x'_2)^2 + 2x_1x'_1x_2x'_2 =$$
$$= \underbrace{\begin{pmatrix}x_1^2 & \sqrt{2}x_1x_2 & x_2^2\end{pmatrix}^T}_{\Phi(\vec{x})^T} \underbrace{\begin{pmatrix}x_1'^2\\ \sqrt{2}x'_1x'_2\\ x_2'^2\end{pmatrix}}_{\Phi(\vec{x}')}$$

16.1.2.2 Inhomogeneous

$$k(\vec{x},\vec{x}') = (1 + \vec{x}^T\vec{x}')^d$$

With $d = 2$:

$$k\left(\begin{pmatrix}x_1\\x_2\end{pmatrix},\begin{pmatrix}x'_1\\x'_2\end{pmatrix}\right) = (1 + x_1x'_1 + x_2x'_2)^2 =$$
$$= 1 + (x_1x'_1)^2 3 + (x_2x'_2)^2 + 2x_1x'_1 + 2x_2x'_2 + 2x_1x'_1x_2x'_2 =$$
$$= \underbrace{\begin{pmatrix}1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & \sqrt{2}x_1x_2 & x_2^2\end{pmatrix}^T}_{\Phi(\vec{x})^T} \underbrace{\begin{pmatrix}1\\ \sqrt{2}x'_1\\ \sqrt{2}x'_2\\ x_1'^2\\ \sqrt{2}x'_1x'_2\\ x_2'^2\end{pmatrix}}_{\Phi(\vec{x}')}$$

16.2 Valid kernels

16.2.1 Dot product in feature space

A valid kernel is a similarity function defined in the Cartesian product of the input space: $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, It corresponds to a dot product in a certain feature space: $k(\vec{x},\vec{x}' = \Phi(\vec{x})^T\Phi(\vec{x}')$. The kernel generalizes the notion of dot product to arbitrary input space and it can be seen as a measure of similarity between objects.

16.2.2 Gram matrix

Given examples $\{\vec{x}_1,\ldots,\vec{x}_m\}$ and a kernel function k the Gram matrix K is the symmetric matrix of pairwise kernels between examples:

$$K_{ij} = k(\vec{x}_i,\vec{x}_j)\forall i,j$$

16.2.2.1 Positive definite matrix

A symmetric $m \times m$ matrix K is positive definite if:

$$\sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0, \forall \vec{c} \in \mathbb{R}^m$$

If this equality holds for $\vec{c} = \vec{0}$ then the matrix is strictly positive definite. A matrix is also positive definite if all eigenvalues are non-negative or there exists a matrix B such that $K = B^T B$.

16.2.3 positive definite kernels

A positive definite kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ giving rise to a positive definite Gram matrix for any m and $\{\vec{x}_1, \ldots, \vec{x}_m\}$. Positive definiteness is necessary and sufficient condition for a kernel to correspond to a dot product of some feature map Φ.

16.2.4 Verifying kernel validity

To verify kernel validity one can prove its positive definiteness, find out a corresponding feature map or use kernel combination properties.

16.3 Support vector regression

16.3.1 Dual problem

$$\max_{\alpha \in \mathbb{R}^m} -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) \underbrace{\Phi(\vec{x}_i)^T \Phi(\vec{x}_j)}_{k(\vec{x}_i, \vec{x}_j)} - \epsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i (\alpha_i^* - \alpha_i)$$

Subject to $\sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0$ with $\alpha_i, \alpha_i^* \in [0, C] \forall i \in [1, m]$. The regression function is:

$$f(\vec{x}) = \vec{w}^T \Phi(\vec{x}) + w_0 = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \underbrace{\Phi(\vec{x}_i)^T \Phi(\vec{x})}_{k(\vec{x}_i, \vec{x})} + w_0$$

16.3.2 Stochastic Perceptron

The function is:

$$f(\vec{x}) = \vec{w}^T \vec{x}$$

1. Initialize $\vec{w} = \vec{0}$.
2. Iterate until all examples are correctly classified and for each incorrectly classified training example $(\vec{x}_i, y_i)$:

$$\vec{w} \leftarrow \vec{w} + \eta y_i \vec{x}_i$$

16.3.3 Kernel Perceptron

The function is:

$$f(\vec{x}) = \sum_{i=1}^{m} \alpha_i k(\vec{x}_i, \vec{x})$$

1. Initialize $\alpha_i = 0 \forall i$.
2. Iterate until all examples are correctly classified and for each incorrectly classified training example $(\vec{x}_i, y_i)$:

$$\alpha_i \leftarrow \alpha_i + \eta y_i$$

16.4 Kernels

16.4.1 Basic kernels

16.4.1.1 Linear kernel

$$k(\vec{x}, \vec{x}') = \vec{x}^T \vec{x}'$$

16.4.1.2 Polynomial kernel

$$k_{d,c}(\vec{x}, \vec{x}') = (\vec{x}^T \vec{x}' + c)^d$$

16.4.2 Gaussian kernel

$$k_\sigma(\vec{x}, \vec{x}' = e^{-\frac{||\vec{x}-\vec{x}'||^2}{2\sigma^2}} = e^{-\frac{\vec{x}^T\vec{x} - 2\vec{x}^T\vec{x}' + \vec{x}'^T\vec{x}}{2\sigma^2}}$$

The Gaussian kernel depends on a width parameter σ. The smaller the width, the more prediction on a point only depends on its nearest neighbours. This is an example of an universal kernel: it can uniformly approximate any arbitrary continuous target function.

16.4.3 Kernels on structured data

Kernels are a generalization of dot products to arbitrary domains. It is possible to design kernels over structured objects like sequences, trees or graphs, the idea is to design a pairwise function measuring the similarity of two objects. This measure has to satisfy the positive definite condition to be a valid kernel.

16.4.3.1 Match or delta kernel

The match or delta kernel it is the simplest kernel on structures and x does not need to be a vector:

$$k_\delta(x, x') = \delta(x, x') = \begin{cases} 1 & if\ x = x' \\ 0 & otherwise \end{cases}$$

16.4.4 Kernels on sequences - spectrum kernel

When considering sequences the feature space is all possible k-grams or subsequences. An efficient procedure based on suffix trees allows to compute the kernel without explicitly building feature maps.

16.4.5 Kernel combination

Simpler kernels can be combined using operators like sum and product. A kernel combination allows to design complex kernels on structures from simpler ones. Correctly using the combination operators guarantees that the complex kernels are positive definite. This is the simplest constructive approach to build valid kernels.

16.4.5.1 Kernel sum

The sum of two kernels corresponds to the concatenation of their respective feature spaces:

$$\begin{aligned}(k_1 + k_2)(\vec{x}, \vec{x}') &= k_1(\vec{x}, \vec{x}') + k_2(\vec{x}, \vec{x}') = \\ &= \Phi_1(\vec{x})^T \Phi(\vec{x}') + \Phi_2(\vec{x})^T \Phi(\vec{x}') = \\ &= \begin{pmatrix}\phi_1(\vec{x}) & \Phi_2(\vec{x})\end{pmatrix} \begin{pmatrix}\Phi_1(\vec{x}') \\ \Phi_2(\vec{x}')\end{pmatrix}\end{aligned}$$

The two kernels can be defined on different spaces.

16.4.5.2 Kernel product

The product of two kernels corresponds to the Cartesian products of their features:

$$\begin{aligned}(k_1 \times k_2)(\vec{x}, \vec{x}') &= k_1(\vec{x}, \vec{x}') k_2(\vec{x}, \vec{x}') = \\ &= \sum_{i=1}^{n} \Phi_{1i}(\vec{x}) \Phi_{i1}(\vec{x}') \sum_{j=1}^{m} \Phi_{2j}(\vec{x}) \Phi_{2j}(\vec{x}') = \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} (\Phi_{1i}(\vec{x}) \Phi_{2j}(\vec{x}))(\Phi_{1i}(\vec{x}') \Phi_{2j}(\vec{x}')) = \\ &= \sum_{k=1}^{nm} \Phi_{12k}(\vec{x}) \Phi_{12k}(\vec{x}') = \Phi_{12}(\vec{x})^T \Phi_{12}(\vec{x}')\end{aligned}$$

Where $\Phi_{12}(\vec{x}) = \Phi_1(\vec{x}) \times \Phi_2(\vec{x})$ and is the Cartesian product. The product can be between kernels in different spaces and is the tensor product.

16.4.5.3 Linear combination

A kernel can be rescaled by an arbitrary positive constant: $k_\beta(\vec{x}, \vec{x}') = \beta k(\vec{x}, \vec{x}')$. A linear combination of kernels can be defined, each rescaled by the desired weight:

$$k_{sum}(\vec{x}, \vec{x}') = \sum_{k=1}^{K} \beta_k k_k(\vec{x}, \vec{x}')$$

The weights of the linear combination can be learned simultaneously to the predictor weights. This is performing kernel learning.

16.4.5.4 Kernel normalization

Kernel values can be influenced by the dimension of objects. This effect can be reduced normalizing the kernel. One way to do so is the cosine normalization, that computes the cosine of the dot product in feature space:

$$\hat{k}(\vec{x}, \vec{x}') = \frac{k(\vec{x}, \vec{x}')}{\sqrt{k(\vec{x}, \vec{x})k(\vec{x}', \vec{x}')}}$$

16.4.5.5 Kernel composition

Given a kernel over structured data $k(\vec{x}, \vec{x}')$ it is always possible to use a basic kernel on top of it:

$$(k_{d,c} \circ k)(\vec{x}, \vec{x}') = (k(\vec{x}, \vec{x}') + c)^d$$

$$(k_\sigma \circ k)(\vec{x}, \vec{x}') = e^{-\frac{k(\vec{x},\vec{x})-2k(\vec{x},\vec{x}')+k(\vec{x}',\vec{x}')}{2\sigma^2}}$$

16.4.6 Kernels on graphs

16.4.6.1 Weistfeiler-Lehman graph kernel

The Weistfeiler-Lehman graph kernel is an efficient graph kernel for large graphs. It relies on an approximation of the Weistfeiler-Lehman test of graph isomorphism. It defines a family of graph kernels.

16.4.6.1.1 Weistfeiler-Lehman isomorphism test Given $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}', \mathcal{E}')$ with $n = |\mathcal{V}| = |mathcalV'|$, let $L(G) = \{l(v)|v \in \mathcal{V}\}$ be the set of labels in G end let $L(G) = L(G')$. Let $label(s)$ be a function assigning a unique label to a string.

1. Set $l_0(v) = l(v) \forall v$.
2. For $i \in [1, n-1]$:
   (a) For each node v in G and G'.
   (b) Let $M_i(v) = \{l_{i-1}(u)|u \in neigh(v)\}$.
   (c) Concatenate the sorted labels of $M_i(v)$ into $s_i(v)$.
   (d) Let $l_i(v) = label(l_{i-1}(v) \circ s_i(v))$ with $\circ$ concatenation.
   (e) If $L_i(G) \neq L_i(G')$.
   (f) Return fail.
3. Return pass.

16.4.6.1.2 Weistfeiler-Lehman graph kernel Let $\{G_0, ;G_h\} = \{(\mathcal{V}, \mathcal{E}, l_0), \ldots, (\mathcal{V}, \mathcal{E}, l_h)\}$ be a sequence of graphs made from G, where l_i is the node labelling of the $i - th$ WL iteration. Let $k : G \times G' \to \mathbb{R}$ be any kernel on graphs. The Weistfeiler-Lehman graph kernel is defined as:

$$k_{WL}^{h}(G, G') = \sum_{i=1}^{h} k(G_i, G_i')$$

Chapter 17

Deep networks

17.1 Need for deep networks

The perceptron can only model linear function. Kernel machines can model non linear function thanks to the non-linearity provided by kernels. There is the need to design appropriate kernels, possibly selecting them from a set like in kernel learning. The solution is typically a linear combination of kernels.

17.2 Multilayer perceptron

The multilayer perceptron is a network of interconnected neurons. It has a layered architecture: neurons from one layer send outputs to the following one. The input layer is typically at the bottom and it is composed of the input features. The output layer is typically at the top and makes the prediction.

17.2.1 Activation function

17.2.1.1 Perceptron

The activation function of the perceptron is a threshold activation:

$$f(\vec{x}) = sign(\vec{w}^T \vec{x})$$

The derivative is zero apart from zero where the function is not differentiable, so it is impossible to run gradient-based optimization.

17.2.1.2 Sigmoid

The sigmoid is a smooth version of a threshold:

$$f(\vec{x}) = \sigma(\vec{w}^T \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$$

It is approximately linear around zero and saturates for large and small values.

17.2.2 Output layer

17.2.2.1 Binary classification

In binary classification there is one output neuron $\sigma(\vec{x})$. It has a sigmoid activation function:

$$f(\vec{x}) = \sigma(o(\vec{x})) = \frac{1}{1 + e^{-o(\vec{x}}}$$

Its decision threshold is at 0.5:

$$y^* = sign(f(\vec{x}) - 0.5$$

17.2.2.2 Multiclass classification

In multiclass classification there is one output neuron per class, together they form the logits layer:

$$[o_1(\vec{x}), \ldots, o_c(\vec{x})]$$

They use the softmax activation function:

$$f_i(\vec{x}) = \frac{e^{o_i(\vec{x})}}{\sum_{j=1}^{c} e^{o_j(\vec{x})}}$$

The decision is the highest scoring class:

$$y^* = \arg \max_{i \in [1,c]} f_i(\vec{x})$$

17.2.2.3 Regression

In regression there is one output neuron $o(\vec{x})$ with a linear activation function. The decision is the value of the output neuron:

$$f(\vec{x}) = o(\vec{x})$$

17.2.3 Representational power of a multilayer perceptron

The function representable by a perceptron are:

- Boolean functions: any boolean function can be represented by some network with two layers of units.
- Continuous functions: every bounded continuous function can be approximated with an arbitrary small error by a network with two layers of unit with a sigmoid hidden activation and linear output activation.
- Arbitrary functions: any function can be approximated to arbitrary accuracy by a network with three layers of units with sigmoid hidden activation and a linear output activation.

17.2.4 Shallow and deep structures for boolean functions

17.2.4.1 Conjunctive normal form

For a conjunctive normal form or CNF there is one neuron for each clause representing OR gates with negative weights for negated literals. Moreover there is one neuron at the top representing the and gate.

17.2.4.2 Number of gates

Some functions require an exponential number of gates like the parity function. This can be expressed with a linear number of gates through a deep network representing a combination of xor gates.

17.2.5 Training MLP

17.2.5.1 Common choices for loss functions

17.2.5.1.1 Cross entropy Minimizing cross entropy corresponds to maximizing likelihood.

17.2.5.1.1.1 For binary classification Let $y \in \{0, 1\}$:

$$l(y, f)\vec{x})) = -(u \log f(\vec{x}) + (1 - y) \log(1 - f(\vec{x})))$$

17.2.5.1.1.2 For multiclass classification Let $y \in [1, c]$:

$$l(y, f(\vec{x})) = -\log f_y(\vec{x})$$

17.2.5.1.2 Mean squared error Mean squared error is used for regression:

$$l(y, f(\vec{x})) = (y - f(\vec{x}))^2$$

17.2.5.2 Stochastic gradient descent

The training error for example $(\vec{x}, y)$ is, for example with regression:

$$E(W) = \frac{1}{2}(y - f(\vec{x}))^2$$

Considering η learning rate the gradient update is:

$$w_{ij} = w_{ij} - \eta \frac{\partial E(W)}{\partial w_{ij}}$$

17.2.5.3 Backpropagation

The backpropagation is used to update the weights in all the layer and the chain rule is used for derivation:

$$\frac{\partial E(W)}{\partial w_{ij}} = \underbrace{\frac{\partial E(W)}{\partial a_i}}_{\delta_i} \frac{\partial a_i}{\partial w_{ij}} = \delta_i \phi_j$$

17.2.5.4 Output units

The δ is easy to compute on output units. For example for regression with linear output:

$$\begin{aligned}\delta_o = \frac{\partial E(W)}{\partial a_o} = \frac{\partial \frac{1}{2}(y - f(\vec{x}))^2}{\partial a_o} = \\ = \frac{\partial \frac{1}{2}(y - a_o)^2}{\partial a_o} = -(y - a_o)\end{aligned}$$

17.2.5.5 Hidden units

To change the weights on the hidden units the error contribution is considered through all outer connections with sigmoid activation:

$$\begin{aligned}\delta_i = \frac{\partial E(W)}{\partial a_i} = \sum_{k \in ch[i]} \frac{\partial E(W)}{\partial a_k} \frac{\partial a_k}{\partial a_i} = \\ = \sum_{k \in ch[i]} \delta_k \frac{\partial a_k}{\partial \phi_i} \frac{\partial \phi_i}{\partial a_l} = \\ = \sum_{k \in ch[i]} \delta_k w_{ki} \frac{\partial \sigma(a_i)}{\partial a_i} = \\ = \sum_{k \in ch[i]} \delta_k w_{ki} \sigma(a_i)(1 - \sigma(a_i))\end{aligned}$$

17.2.5.5.1 Derivative of sigmoid

$$\begin{aligned}\frac{\partial \sigma(x)}{\partial x} = \frac{\partial}{\partial x} \frac{1}{1 + e^{-x}} = \\ = -(1 + e^{-x})^{-2} \frac{\partial}{\partial x}(1 + e^{-x}) = \\ = -(1 + e^{-x})^{-2} - e^{-2x} \frac{\partial e^x}{\partial x} = \\ = (1 + e^{-x})^{-2} e^{-2x} e^x = \\ = \frac{1}{1 + e^x} \frac{e^{-x}}{1 + e^{-x}} = \\ = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) = \\ = \sigma(x)(1 - \sigma(x))\end{aligned}$$

17.2.5.6 Modular nature of deep architectures

The modular structure of deep architectures becomes clear when considering the loss function: For any neuron at layer j: $\phi_j = F_j(\phi_{j-1}, W_j)$, then:

$$\frac{\partial E}{\partial W_j} = \frac{\partial E}{\partial \phi_j} \frac{\partial \phi_j}{\partial W_j} = \frac{\partial E}{\partial \phi_j} \frac{\partial F_j(\phi_{j-1}, W_j)}{\partial W_j}$$

Also:

$$\frac{\partial E}{\partial \phi_{j-1}} = \frac{\partial E}{\partial \phi_j}\frac{\partial \phi_j}{\partial \phi_{j-1}} = \frac{\partial E}{\partial \phi_j}\frac{\partial F_j(\phi_{j-1}, W_j)}{\partial \phi_{j-1}}$$

17.2.5.7 Remarks on backpropagation - local minima

The error surface of a multilayer neural network can contain several minima and backpropagation is only guaranteed to converge to a local one. There are heuristic that attempts to address the problem using stochastic instead of true gradient descent, training multiple networks with different random weights and others. Because kernel machines' training requires solving a quadratic optimization problem the global optimum is guaranteed. Deep networks are more expressive in principle, but they are harder to train.

17.2.6 Stopping criterion and generalization

The training termination condition is a crucial factor: overtraining the network increases the possibility of overfitting the training data. The network is initialized with small random weights so there is a simple decision surface. Overfitting occurs at later iteration, when increasingly complex surfaces are being generated. To choose when to stop a separate validation set is used to estimate performance of the network and choose when to stop training.

17.2.7 Vanishing gradient

The error gradient is backpropagated through layers and at each step it is multiplied by the derivative of the sigmoid that is very small for saturated units. It happens that the gradient vanishes in lower layer, making it difficult to train deep networks.

17.3 Trick of the trade

17.3.1 Introduction

Do not initialize weights to zero, but to small random values around zero. Standardize inputs $\vec{x} = \frac{\vec{x}-\vec{\mu}_{\vec{x}}}{\sigma_{\vec{x}}}$ to avoid the saturation of the hidden layers. Randomly shuffle training examples before each training epoch.

17.3.2 Activation functions

Linearity is nice for learning and saturation is bad for learning, so the rectifier activation has been introduced:

$$f(\vec{x}) = \max(0, \vec{w}^T\vec{x})$$

Neurons employing the rectifier activation are called rectified linear units *ReLU*.

17.3.3 Regularization

17.3.3.1 2-norm regularization

2-norm regularization penalizes weights by their Euclidean norm and weights with less influence on error get smaller values:

$$J(W) = E(W) + \lambda ||W||_2$$

17.3.3.2 1-norm regularization

1-norm regularization penalizes weights by the sum of their absolute values. It encourages less relevant weights to be exactly zero, inducing sparsity:

$$J(W) = E(W) + \lambda |W|$$

17.3.4 Initialization

Randomly initializing weights breaks symmetries between neurons. There is a need to carefully set an initialization range to preserve forward and backward variance:

$$W_{ij} \sim U\left(-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\right)$$

Where n and m are the number of inputs and outputs. A sparse initialization enforce a fraction of weights to be non-zero encouraging diversity between neurons.

17.3.5 Gradient descent

17.3.5.1 Batch vs stochastic

Batch gradient descent updates parameters after seeing all the examples and it is too slow for large dataset. A fully stochastic gradient descent updates parameters after seeing each examples and the objective is too different from the true one. In minibatch gradient descents the parameters are updated after seeing a minibatch m of examples.

17.3.5.2 Momentum

$$v_{ji} = \alpha v_{ji} - \eta \frac{\partial E(W)}{\partial w_{lj}}$$

$$\vec{w}_{ji} = w_{ji} + v_{ji}$$

$0 \leq \alpha \leq 1$ is the momentum, which tends to keep updating weights in the same direction. Thinking of a ball rolling on the error surface it can roll through local minima without stopping, it can traverse flat surfaces instead of stopping and increase the step size of search in region of constant gradient.

17.3.6 Adaptive gradient

17.3.6.1 Decreasing learning rate

$$\eta_t = \begin{cases} \left(1 - \frac{t}{\tau}\right)\eta_0 + \frac{t}{\tau}\eta_\tau & if\ t < \tau \\ \eta_\tau & otherwise \end{cases}$$

Larger learning rate at the beginning for faster convergence towards the attraction basin and smaller learning rate at the end to avoid oscillations close to the minimum.

17.3.6.2 Adagrad

Adagrad reduces the learning rate in steep directions and increase it in gentler ones.

$$r_{ij} = r_{ij} + \left(\frac{\partial E(W)}{\partial w_{lj}}\right)^2$$

$$\vec{w}_{ji} = \vec{w}_{ji} - \frac{\eta}{\sqrt{r_{ji}}}\frac{\partial E(W)}{\partial w_{lj}}$$

The square gradient is accumulated over all iterations and for non-convex problem this learning rate reduction can be excessive or premature.

17.3.6.3 RMSProp

RMSProp makes an exponentially decaying accumulation of squared gradient $0 < \rho < 1$. It avoid the premature reduction of Adagrad and has Adagrad-like behaviour when reaching the convex bowl.

$$r_{ij} = \rho r_{ij} + (1 - \rho)\left(\frac{\partial E(W)}{\partial w_{lj}}\right)^2$$

$$\vec{w}_{ji} = \vec{w}_{ji} - \frac{\eta}{\sqrt{r_{ji}}}\frac{\partial E(W)}{\partial w_{lj}}$$

17.3.7 Batch normalization

17.3.7.1 Covariate shift problem

The covariate shift problem happens when the input distribution to the model changes over time and the model does not adapt to it. In very deep networks internal covariate shift takes place among layers when they get updated by backpropagation. To solve this problem each node activation need to be normalized by its batch statistics:

$$\bar{x}_i = \frac{x_i - \mu_B}{\sigma_B}$$

Where x is the activation of an arbitrary layer, $B = \{x_1, \ldots, x_m\}$ is a batch of values for that activation and μ_B and σ_B^2 are the batch mean and variance. Scaling and shifting each activation with adjustable parameters γ and β that becomes part of the network parameters yields:

$$y_i = \gamma\bar{x}_i + \beta$$

This allows for more robustness to parameteactivation and μ_B and σ_B^2 are the batch mean and variance. Scaling and shifting each activation with adjustable parameters γ and β that becomes part of the network parameters yields:

$$y_i = \gamma \bar{x}_i + \beta$$

This allows for more robustness to parameter initialization, it allows for faster learning rates without divergence, keeps the activations in a non-saturated region and regularizes the model.

17.3.8 Pre-training

17.3.8.1 Layerwise pre-training

In layerwise pre-training each layer is trained by itself with actual labels.

17.3.8.2 Transfer learning

In transfer learning the network is first trained on a similar task, then the last layer is discarded and a new one is added and the resulting network is retrained on the target task.

17.3.8.3 Multi-level supervision

In multi-level supervision auxiliary output nodes are put at intermediate layers to speed up learning.

17.4 Popular deep architectures

There are many different possible architectures for deep neural networks:

- Convolutional networks for exploiting local correlations.
- Recurrent and recursive networks for collective predictions.
- Deep Boltzmann machines as probabilistic generative models.
- Generative adversarial networks to generate new instances as a game between discriminator and generator.

17.4.1 Autoencoders

Autoencoders are networks that perform unsupervised representation learning. The network is trained to reproduce the input in the output. It learns to map inputs into a sensible hidden representation and it can be done with unlabelled examples.

17.4.2 Convolutional networks

Convolutional networks CNN use location invariance and compositionality: convolution filters extract local features, pooling provide invariance to local variations, a hierarchy of filters compose complex features from simpler ones and there is a fully connected layer for final classification.

17.4.3 Long Short-Term Memory networks

Long short-term memory networks or LSMT perform recurrent computation with selective memory: a cell state is propagated along a chain, the forget gate selectively forgets parts of the cell state, the input gate selectively chooses parts of the candidate for cell update and the output gate selectively chooses parts of the cell state for output.

17.4.4 Generative adversarial networks

Generative adversarial networks or GANs perform generative learning as an adversarial game. A generator network learns to generate items from random noise. A discriminator network learns to distinguish between real items and generated ones. The two networks are jointly learned as an adversarial game. There is no need of human supervision.

17.4.4.1 Transformers

Transformers use an attention mechanisms to learn input words encodings that depend on other words in the sentence. The attention mechanism is used to learn output words encoding that thepend on input wird encodings and previously generated output words.

17.4.5 Graph neural networks

Graph neural networks allow to learn with graph convolution. They allow to learn feature representations for node, to propagate information between neighbouring nodes and efficient training with respect to graph kernels.

Chapter 18

Unsupervised learning

18.1 Setting

Supervised learning requires the availability of labelled examples. Labelling examples can be an extremely expensive process and sometimes the labels are not known. Unsupervised techniques can be employed to group examples into clusters.

18.2 K-means clustering

18.2.1 Setting

K-means clustering assumes that examples should be grouped into k cluster, with each cluster i represented by its measn $\vec{\mu}_i$.

18.2.2 Algorithm

1. Initialize cluster means $\vec{\mu}_1, \ldots, \vec{\mu}_k$.
2. Iterate until no mean changes:
   (a) Assign each example to the cluster with the nearest mean.
   (b) Update cluster means according to assigned examples.

18.2.3 Defining similarity

18.2.3.1 Similarity measures

18.2.3.1.1 Standard euclidean distance The standard euclidean distance in $\mathbb{R}^D$ is:

$$d(\vec{x}, \vec{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$$

18.2.3.1.2 Generic Minkowski metric The generic Minkowski metric for $p \geq 1$ is:

$$d(\vec{x}, \vec{x}') = \left(\sum_{i=1}^{d} |x_i - x_i'| \right)^{\frac{1}{p}}$$

18.2.3.1.3 Cosine similarity The cosine similarity is the cosine of the angle between vector and is:

$$s(\vec{x}, \vec{x}') = \frac{\vec{x}^T \vec{x}'}{||\vec{x}||||\vec{x};||}$$

18.2.3.2 Define the quality of the obtained clusters

The sum-of-squared error criterion is used to define the quality of the obtained clusters. Let n_i be the number of samples in cluster $\mathcal{D}_i$ and $\vec{\mu}_i$ the cluster sample mean:

$$\vec{\mu}_i = \frac{1}{n_i} \sum_{\vec{x} \in \mathcal{D}_i} \vec{x}$$

The sum-of-squared errors is defined as:

$$E = \sum_{i=1}^{k} \sum_{\vec{x} \in \mathcal{D}_i} ||\vec{x} - \vec{\mu}_i||^2$$

18.3 Gaussian Mixture model GMM

In the Gaussian mixture model GMM the examples are clustered using a mixture of Gaussian distributions. Assuming that a number of Gaussians is given it estimates mean and variance of each Gaussian.

18.3.1 Parameter estimation

Maximum likelihood estimation cannot be applied as cluster assignment is unknown. So the expectation-maximisation approach is used:

1. Compute expected cluster assignment given current parameter setting.
2. Estimate parameters given cluster assignment.
3. Iterate

18.3.2 Estimating means of k univariate Gaussians

18.3.2.1 Setting

A dataset of $x_1, \ldots, x_n$ examples is observed. For each example x_i the cluster assignment is modelled as $z_{i1}, \ldots, z_{ik}$ binary latent variables. $z_{ij} = 1$ if Gaussian j generated x_i and 0 otherwise. Parameters to be estimated are $\mu_1, \ldots, \mu_k$ the Gaussians means. All Gaussians are assumed to have the same known variance σ^2.

18.3.2.2 Algorithm

1. Initialize $h = \langle \mu_1, \ldots, \mu_k \rangle$
2. Iterate until difference in maximum likelihood is below a certain threshold:

E-step Calculate expected value $\mathbb{E}[z_{ij}]$ for each latent variable assuming current hypothesis $h = \langle \mu_1, \ldots, \mu_k \rangle$ holds.

M-step Calculate a new maximum likelihood hypotheses $h' = \langle \mu'_1, \ldots, \mu'_k \rangle$ assuming values of latent variables are their expected values just composed. Replace $h \leftarrow h'$.

18.3.2.2.1 E-step The expected value of z_{ij} is the probability that x_i is generated by the Gaussian j assuming hypothesis $h = \langle \mu_1, \ldots, \mu_k \rangle$ holds:

$$\mathbb{E}[z_{ij}] = fracp(x_i|\mu_j)\sum_{l=1}^{k} p(x_i|\mu_l) = \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{l=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i-\mu_l)^2}}$$

18.3.2.2.2 M-step The maximum-likelihood mean μ_j is the weighted sample mean, each instance being weighted by its probability of being generated by Gaussian j:

$$\mu'_j = \frac{\sum_{i=1}^{n} \mathbb{E}[z_{ij}]x_i}{\sum_{i=1}^{n} \mathbb{E}[z_{ij}]}$$

18.4 Expectation-Maximization (EM)

18.4.1 Formal setting

Given a dataset made of an observed part X and an unobserved part Z, there is a need to estimate the hypothesis maximizing the expected log-likelihood for the data, with expectation taken over unobserved data:

$$h^* = \arg\max_h \mathbb{E}_z[\ln p(X, Z|h)]$$

The unobserved data Z should be treated as random variables governed by the distribution depending on X and h.

18.4.2 Generic algorithm

1. Initialize hypothesis h.
2. Iterate until convergence:

E-step Compute the expected likelihood of an hypothesis h' for the full data, where the unobserved data distribution is modelled according to the current hypothesis h and the observed data:

$$Q(h; h') = \mathbb{E}_z[\ln p(X, Z|h')|h, X]$$

M-step Replace the current hypothesis with the one maximizing $Q(h'; h)$:

$$h \leftarrow \arg\max_{h'} Q(h'; h)$$

18.4.3 Estimating means of k univariate Gaussians

18.4.3.1 Derivation

The likelihood of an example is:

$$p(x_i, z_{i1}, \ldots, z_{ij}|h') = \frac{1}{\sqrt{2\pi}\sigma} e^{-\sum_{j=1}^{k} z_{ij} \frac{(x_i - \mu_j)^2}{2\sigma^2}}$$

The dataset log-likelihood is:

$$\ln p(X, Z|h) = \sum_{i=1}^{n} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \sum_{j=1}^{k} z_{ij} \frac{x_i - \mu'_j)}{2\sigma^2} \right)$$

18.4.3.2 E-step

The expected log-likelihood, remembering the linearity of the expectation operator:

$$\begin{aligned} \mathbb{E}_z[\ln p(X, Z|h')] &= \mathbb{E}_z \left[\sum_{i=1}^{n} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \sum_{j=1}^{k} z_{ij} \frac{(x_i - \mu_j)^2}{2\sigma^2} \right) \right] = \\ &= \sum_{i=1}^{n} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \sum_{j=1}^{k} \mathbb{E}[z_{ij}] \frac{(x_i - \mu'_j)^2}{2\sigma^2} \right) \end{aligned}$$

The expectation given current hypothesis h and observed data X is computed as:

$$\mathbb{E}[z_{ij}] = \frac{p(x_i|\mu_j)}{\sum_{l=1}^{k} p(x_i|\mu_l)} = \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{l=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i - \mu_l)^2}}$$

18.4.3.3 M-step

The likelihood maximization gives:

$$\begin{aligned} \arg\max_{h'} Q(h', h) &= \arg\max_{h'} \sum_{i=1}^{n} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \sum_{j=1}^{k} \mathbb{E}[z_{ij}] \frac{(x_i - \mu'_j)^2}{2\sigma^2} \right) = \\ &= \arg\min_{h'} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{E}[z_{ij}] (x_i \mu'_j)^2 \end{aligned}$$

Zeroing the derivative with respect to each mean:

$$\frac{\partial}{\partial \mu_j} = -2 \sum_{i=1}^{n} \mathbb{E}[z_{ij}](x_i - \mu'_j) = 0$$

$$\mu'_j = \frac{\sum_{i=1}^{n} \mathbb{E}[z_{ij}] x_i}{\sum_{i=1}^{n} \mathbb{E}[z_{ij}]}$$

18.5 Choosing the number of clusters

18.5.1 Elbow method

18.5.1.1 Idea

Increasing the number of clusters allows for better modelling of data. There is a need to trade-off the quality of the cluster with their quantity so the number of clusters stop being increased when the advantage is limited. This method can be ambiguous with multiple candidate points.

18.5.1.2 Approach

1. Run clustering algorithm for increasing number of clusters.
2. Plot clustering evaluation metric like the sum of squared errors for different k.
3. Choose k when there is an angle making an angle in the plot, or a drop in gain.

18.5.2 Average silhouette method

18.5.2.1 Idea

Increasing the number of clusters makes each cluster more homogeneous and make different clusters more similar. A quality metric that trades-off intra-cluster similarity and inter-cluster dissimilarity can be used.

18.5.2.2 Silhouette coefficient for example i

1. Compute the average dissimilarity between i and examples of its cluster C:

$$a_i = d(i, C) = \frac{1}{|C|} \sum_{j \in C} d(i, j)$$

2. Compute the average dissimilarity between i and examples of each clusters $C' \neq C$ and take the minimum:

$$b_i = \min_{C' \neq C} d(i, C')$$

3. The silhouette coefficient is:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

18.5.2.3 Approach

1. Run a cluster algorithm for increasing number of clusters.
2. Plot the average over the examples silhouette coefficient for different k.
3. Choose k where the average silhouette coefficient is maximal.

18.6 Hierarchical clustering

18.6.1 Setting

Clustering does not need to be flat: natural grouping of data is often hierarchical and a hierarchy of clusters can be built on examples.

18.6.2 Top-down approach

1. Start from a single cluster with all examples.
2. Recursively split clusters into subclusters.

18.6.3 Bottom-up approach

1. Start with n clusters of individual examples or singletons.
2. Recursively aggregate pairs of clusters.

18.6.4 Dendograms

A dendogram is a representation of hierarchical clusters. It is a tree where the root is the total dataset and each child represent a subcluster. The leaves are the examples.

18.6.5 Agglomeritive hierarchical clustering

18.6.5.1 Algorithm

1. Initialize: final cluster number k, initial cluster number $\hat{k} = n$ and initial clusters $\mathcal{D}_i = \{x_i\}, i \in \{1, \ldots, n\}$.
2. While $\hat{k} > k$:
   (a) Find pairwise nearest clusters $\mathcal{D}_i$, $\mathcal{D}_j$.
   (b) Merge $\mathcal{D}_i$ and $\mathcal{D}_j$.
   (c) Update $\hat{k} - = 1$.

The stopping criterion can be a threshold on pairwise similarity.

18.6.6 Measuring cluster similarities

- Nearest-neighbour:

$$d_{\min}(\mathcal{D}_i, \mathcal{D}_j) = \min_{\vec{x} \in \mathcal{D}_i, \vec{x}' \in \mathcal{D}_j} ||\vec{x} - \vec{x}'||$$

- Farthest-neighbour:

$$d_{\max}(\mathcal{D}_i, \mathcal{D}_j) = \max_{\vec{x} \in \mathcal{D}_i, \vec{x}' \in \mathcal{D}_j} ||\vec{x} - \vec{x}'||$$

- Average distance:

$$d_{avg}(\mathcal{D}_i, \mathcal{D}_j) = \frac{1}{n_i n_j} \sum_{\vec{x} \in \mathcal{D}_i} \sum_{\vec{x}' \in \mathcal{D}_j} ||\vec{x} - \vec{x}'||$$

- Distance between means:

$$d_{means}(\mathcal{D}_i, \mathcal{D}_j) = ||\vec{\mu}_i - \vec{\mu}_j||$$

Nearest-neighbour and farthest-neighbour are more sensitive to outliers.

18.6.7 Stepwise optimal hierarchical clustering

18.6.7.1 Algorithm

1. Initialize: final cluster number k, initial cluster number $\hat{k} = n$ and initial clusters $\mathcal{D}_i = \{x_i\}, i \in \{1, \ldots, n\}$.
2. While $\hat{k} > k$:
   (a) Find best clusters $\mathcal{D}_i$, $\mathcal{D}_j$ to merge according to the evaluation criterion.
   (b) Merge $\mathcal{D}_i$ and $\mathcal{D}_j$.
   (c) Update $\hat{k} -= 1$.

Chapter 19

Reinforcement learning

19.1 Introduction

19.1.1 Setting

The learner is provided a set of possible states $\mathcal{S}$ and, for each state, a set of possible actions, $\mathcal{A}$ moving it to a new state. In performing action a from state s, the learner is provided an immediate reward $r(s, a)$. The task is to learn a policy allowing to choose for each state s the action a maximizing the overall reward including future moves. The learner has to deal with problems of delayed rewards coming from future moves and trade-off between exploitation and exploration. Typical application include moving policies for robots and sequential scheduling problems in general.

19.1.2 Sequential decision making

An agent needs to take a sequence of decisions and should maximise some utility function. There is uncertainty in the result of a decision.

19.2 Markov decision process MPD

A Markov decision process is defined by:

- A set of states $\mathcal{S}$ in which the agent can be at each time instant,
- A possibly empty set of terminal states $\mathcal{S}_G \subset \mathcal{S}$,
- A set of actions $\mathcal{A}$ that the agent can make,
- A transition model providing the probability of going to a state s' with action a from state s:

$$p(s'|s, a) \qquad s, s' \in \mathcal{S}, a \in \mathcal{A}$$

- A reward $R(s, a, s')$ for making action a in state s and reaching state s'

19.2.1 Utilities over time

An environment history is defined as a sequence of states. Utilities are defined over environment histories. An infinite horizon with no constraint on the number of time steps is assumed. A stationary preference where if one history is preferred to another at time t, the same should hold at time t' provided they start from the same date is assumed. With these assumptions two sensible way to define utilities are:

Additive rewards

$$U([s_0, \ldots, s_2]) = R(s_0) + \cdots + R(s_n)$$

Discounted rewards for $\gamma \in [0, 1]$:

$$U([s_0, \ldots, s_2]) = \gamma^0 R(s_0) + \cdots + \gamma^{n-1} R(s_n)$$

Only rewards that only depend on the destination state are considered, in a more general case: $R(s_t, a_t, s_{t+1})$

19.2.2 Taking decisions

19.2.2.1 Optimal policy

A policy π is a full specification of what action to take at each state. The expected utility of a policy is the utility of an environment history taken in expectation over all possible histories generated with that policy. An optimal policy π^* is a policy maximizing expected utility. For infinite horizons optimal policies are stationary and depends only on the current state.

19.2.2.1.1 Utility of state The utility of a state given a policy π is:

$$U^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^t R(S_{t+k+1}) | S_t = s \right]$$

Where S_{i+k+1} is the state reached after k steps using policy π starting from $S_t = s$. The true utility of a state is its utility under an optimal policy:

$$U(s) = U^{\pi^*}(s)$$

Given the true utility, an optimal policy is:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s, a) U(s')$$

19.2.3 Computing an optimal policy

The utility of a state is its immediate reward plus the expected discounted utility of the next state, assuming that the agent chooses an optimal action.

19.2.3.1 Bellman equation

$$U(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)U(s')$$

There is a Bellman equation for each state $s \in \mathcal{S}$ where utilities of states are solutions of the set of Bellman equations. The solutions to the set of Bellman equations are unique. Directly solving the set of equation is hard and because of the non-linearities introduced by the max operation.

19.2.3.2 Value iteration

1. Initialize $U_0(s)$ to zero for all s.
2. Repeat:
   (a) Do a Bellman update for each state s:
   $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)U_i(s')$$
   (b) $i+ = 1$
3. Until max utility difference is below a threshold.
4. Return U.

The optimal policy can be set as:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)U(s')$$

19.2.3.3 Policy iteration

1. Initialize π_0 randomly.
2. Repeat:
   (a) Policy evaluation, solve the set of linear equations:
   $$U_i(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi_i(s))U_i(s') \qquad \forall s \in \mathcal{S}$$
   (b) Policy improvement:
   $$\pi_{i+1}(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)U_i(s')quad\forall s \in \mathcal{S}$$
3. Until there is no policy improvement.
4. Return π

19.3 Dealing with partial knowledge

Value and policy iteration assume perfect knowledge of environment, transition model and rewards. In most cases some of these aspects are not known. Reinforcement learning aims at learning policies by space exploration. With policy evaluation a policy is given and the environment is learned, with policy improvement both policy and environment are learned.

19.3.1 Policy evaluation in an unknown environment

19.3.1.1 Adaptive dynamic programming

19.3.1.1.1 Algorithm

1. Loop.
   (a) Initialize s.
      i. Receive reward r, set $R(s) = r$.
      ii. Choose next action $a \leftarrow \pi(s)$.
      iii. Take action a, reach step s'.
      iv. Update counts:

$$N_{sa} \leftarrow N_{sa} + 1 \qquad N_{s'|sa} \leftarrow N_{s'|sa} + 1$$

      v. Update transition model:

$$p(s''|s,a) \leftarrow \frac{N_{s''|sa}}{N_{sa}} \qquad \forall s'' \in \mathcal{S}$$

      vi. Update utility estimate:

$$U \leftarrow PolicyEvaluation(\pi, U, p, R, \gamma)$$

   (b) Until s is terminal.

19.3.1.1.2 Characteristics The algorithm performs maximum likelihood estimation of transition probabilities and upon updating the transition model it calls a standard policy evaluation to update the utility estimate (U is initially empty). Each step is expensive as it runs policy evaluation.

19.3.1.2 Temporal-difference policy evaluation

19.3.1.2.1 Rationale Temporal-difference (TD) policy evaluation avoids running the policy evaluation at each iteration locally updating the utility. If transition from s to s' is observed if s' was always the successor of s, its utility should be:

$$U(s) = R(s) + \gamma U(s')$$

The temporal-difference update rule updates the utility to get closer to that situation:

$$U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

19.3.1.2.2 Algorithm

1. Loop.
   (a) Repeat.
      i. Receive reward r.
      ii. Choose next action a and reach step s'.
      iii. Update the local utility estimate:

$$U(s) \leftarrow U(s) + \alpha(r + \gamma U(s') - U(s))$$

   (b) Until s is terminal

19.3.1.2.3 Characteristics There is no need for a transition model for the utility update and each step is faster than ADP, without loosing anything on the long run. It takes longer to converge and can e seen as a rough efficient approximation of ADP.

19.3.2 Policy learning in an unknown environment

19.3.2.1 Setting

Policy learning requires to combine the learning of the environment and the learning of the optimal policy for it. A simple option consist of replacing the policy evaluation in ADP with optimal policy computation given the current knowledge of the environment creating a greedy agent:

$$U(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a) U(s')$$

Because the knowledge of the environment is incomplete a greedy agent usually learns a suboptimal policy because of lack of exploration.

19.3.2.2 Exploration-exploitation trade-off

Exploitation consists in following promising directions given the current knowledge and exploration consists in trying novel directions looking for better unknown alternatives. A reasonable trade-off should be used in defining the search scheme. For example an ϵ-greedy strategy chooses a random move with probability ϵ and is greedy otherwise. Moreover a higher utility estimate can be assigned to relatively unexplored state-action pairs:

$$U^+(s) = R(s) + \gamma \max_{a \in \mathcal{A}} f\left(\sum_{s' \in \mathcal{S}} p(s'|s,a) U^+(s'), N_{sa}\right)$$

With f increasing over the first argument and decreasing over the second.

19.3.2.3 Temporal-difference learning - learning utilities of actions

The TD policy evaluation can also be adapted to learn an optimal policy. If TD is used to learn a state utility function it needs to estimate a transition model to derive a policy. TD can instead be applied to learn an action utility function $Q(s,a)$ and the optimal policy corresponds to:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

19.3.2.4 SARSA - on-policy temporal-difference learning

1. Loop.
   (a) Initialize s.
   (b) Repeat.
      i. Receive reward r.
      ii. Choose next action $a \leftarrow \pi^\epsilon(s)$.
      iii. Take action a and reach step s'.
      iv. Choose action $a' \leftarrow \pi^\epsilon(s')$.
      v. Update local utility estimate:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

   (c) Until s is terminal.

π^ϵ is an ϵ-greedy or some other form of non-greedy policy based on Q.

19.3.2.5 Q-learning - off-policy temporal-difference learning

1. Loop.
   (a) Initialize s.
   (b) Repeat.
      i. Receive reward r.
      ii. Choose next action $a \leftarrow \pi^\epsilon(s)$.
      iii. Take action a and reach step s'.
      iv. Update local utility estimate:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

   (c) Until s is terminal.

19.3.2.6 SARSA vs Q-learning

SARSA is on-policy: it updates Q using the current policy;s action. Q-learning is off-policy: it updates Q using the greedy policy's action. Off-policy methods are more flexible and they can learn from traces generated with an unknown policy. On-policy methods tend to converge faster and are easier to use for continuous-state spaces and linear function approximators.

19.4 Scaling to large state spaces

19.4.1 Function approximation

All techniques seen so far assume a tabular representation of utility functions. This type of representation doesn't scale to large state spaces. The solution is to rely on function approximation: $U(s)$ or $Q(s,a)$ are approximated with a parametrized function. The function takes a state representation as input and allows to generalize to unseen states.

19.4.2 Learning the approximation function

19.4.2.1 Temporal-difference learning - state utility

The TD error is:

$$E(s, s') = \frac{1}{2}(R(s) + \gamma U_{\vec{\theta}}(s') - U_{\vec{\theta}}(s))^2$$

The error gradient with respect to the function's parameter is:

$$\nabla_{\vec{\theta}} E(s, s') = (R(s) + \gamma U_{\vec{\theta}}(s') - U_{\vec{\theta}}(s))(-\nabla_{\vec{\theta}} U_{\vec{\theta}}(s))$$

So the stochastic gradient update rule is:

$$\begin{aligned}\vec{\theta} &= \vec{\theta} - \alpha \nabla_{\vec{\theta}} E(s, s') = \\ &= \vec{\theta} + \alpha(R(s) + \gamma U_{\vec{\theta}}(s') - U_{\vec{\theta}}(s))(\nabla_{\vec{\theta}} U_{\vec{\theta}}(s))\end{aligned}$$

19.4.2.2 Temporal-difference learning - action utility Q-learning

The TD error is:

$$E((s, a), s') = \frac{1}{2}(R(s) + \gamma \max_{a' \in \mathcal{A}} Q_{\vec{\theta}}(s', a') - Q_{\vec{\theta}}(s, a))^2$$

The error gradient with respect to the function's parameter is:

$$\nabla_{\vec{\theta}} E((s, a), s') = (R(s) + \gamma \max_{a' \in \mathcal{A}} Q_{\vec{\theta}}(s', a') - Q_{\vec{\theta}}(s, a))(-\nabla_{\vec{\theta}} Q_{\vec{\theta}}(s, a))$$

So the stochastic gradient update rule is:

$$\begin{aligned}\vec{\theta} &= \vec{\theta} - \alpha \nabla_{\vec{\theta}} E((s, a), s') = \\ &= \vec{\theta} + \alpha(R(s) + \gamma \max_{a' \in \mathcal{A}} Q_{\vec{\theta}}(s', a') - Q_{\vec{\theta}}(s, a))(\nabla_{\vec{\theta}} Q_{\vec{\theta}}(s, a))\end{aligned}$$