17s1: COMP9417 Machine Learning and Data Mining

Final Exam: Assessable learning objectives
Topic: Summary of and guide to learning objectives
Last revision: Sun Jun 18 22:51:37 AEST 2017

1 Overview

All course material is potentially assessable in the final exam. This means you should revise the content of all lectures. However, the final exam will have more emphasis on the course material that was not assessable in the mid-session exam.

As a general guide to the final exam, recall the overall course learning objectives. By the end of the subject, students should be able to:

- set up a well-defined learning problem for a given task
- select and define a representation for data to be used as input to a machine learning algorithm
- select and define a representation for the model to be output by a machine learning algorithm
- compare different algorithms according to the properties of their inputs and outputs
- compare different algorithms in terms of similarities and differences in the computational methods used
- develop and describe algorithms to solve a learning problem in terms of the inputs, outputs and computational methods used
- express key concepts from the foundations of computational and statistical learning theory and demonstrate their applicability
- express knowledge of general capabilities and limitations of machine learning from computational and statistical theory
- use or extend or invent algorithms in applications to real-world data sets and collect results to enable evaluation and comparison of their performance

This note re-states aims of each lecture as assessable learning objectives and gives brief extra guidance on what you should know for each.

2 Supervised Learning – Regression

The Assessable Learning Objectives (or Aims) for this lecture are to be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

(1) the supervised learning task of numeric prediction
(2) how linear regression solves the problem of numeric prediction
(3) fitting linear regression by least squares error criterion
(4) non-linear regression via linear-in-the-parameters models

(5) parameter estimation for regression
(6) local (nearest-neighbour) regression

The aims in this lecture should be fairly self-explanatory. Aim (1) is on slides 4–10. Aim (2) is on slides 11–15, and the derivation of the solution for linear regression that solves the problem of numeric prediction is on slides 46–47. It is also important to know the additional properties of linear regression on slides 48–52, as these will support your understanding of the approach.

Aim (3) is covered in several places: on slide 14 and slides 38–45; however it is a good idea to make sure you are somewhat familiar with the additional supporting material on slides 16–18 which covers some of the probabilistic and statistical background. For example, the bias-variance decomposition makes its first appearance in this course on slides 29–31.

Aim (4) is on slides 70–71. Note that the closely related idea of training by gradient descent appears on slide 60. These ideas return when we look at neural networks.

The core of aim (5) on parameter estimation is covered with least squares fitting under aim (3), but further issues to do with multiple regression appear on slides 61–62, and on model selection and regularization (to prevent overfitting) appear on slides 72–77.

Aim (6) is on slides 81–84. This is closely related to nearest neighbour classification, which is the topic at the start of the next lecture.

3 Supervised Learning – Classification (1)

Please note: slide numberings refer to the updated version of the slides dated June 12th, 2017. Aims for this lecture are as for the previous lecture for the following topics.

(1) describe distance measures and how they are used in classification
(2) outline the basic k-nearest neighbour classification method
(3) define MAP and ML inference using Bayes theorem
(4) define the Bayes optimal classification rule in terms of MAP inference
(5) outline the Naive Bayes classification algorithm
(6) outline the Perceptron classification algorithm

In this lecture the first method we consider is (k) nearest neighbour classification (kNN), which relies on a distance measure. Many distance measures are used in machine learning; Euclidean distance is widely-used, and sometimes Manhattan distance (slide 7). The distance measures used are (aim 1) are versions of the general statement of the $p-norm$ or Minkowski distance defined on slides 6–9. Additional details on distance measures for machine learning can be treated as background only (slides 10–22). The main aspects of k-nearest neighbour classification (aim 2) are described on slides 23–33. Further issues concerning kNN, such as why you would use distance weighting, and the Curse of Dimensionality, are on slides 34–40.

The second method covered is the simplest form of Bayesian modelling for classification – the popular Naive Bayes classifier. Although the Naive Bayes Classifier is not a complex algorithm it relies on some foundational concepts in machine learning that are introduced before we the algorithm and its application can be discussed. Bayes Theorem for the posterior probability of a hypothesis (e.g., as output by a machine learning algorithm) given a dataset is on slides 45–52. Key aspects of this idea are finding the maximum posterior (MAP) probability or maximum likelihood (ML) hypotheses. For instance, we see that learning a real-valued function of one variable (the linear regression setting of the previous lecture) can be seen (with the assumption of Gaussian noise) as a version of maximum likelihood on slides 63–66.

An important contrast can be made between two classes of machine learning approach in terms of the probability distribution underlying the models that are used. This is shown on slide 67, where we contrast discriminative and discriminative approaches. Discriminative approaches, like linear regression, model the learning problem as determining the conditional probability of the dependent variable Y (which could be a class value) given the feature values X. However, if we have the full joint probability for X and Y we can obtain the conditional probability and do classification, in addition to other probabilistic inferences. Modelling the full joint probability of the variables in our problem is usually termed a generative approach, since any dataset can be thought of as having been *generated* according to such a distribution. The Naive Bayes classifier is a generative model, where strong conditional independence assumptions simplify the probabilities so that learning can be done efficiently. This is outlined and two of the most common ways in which the algorithm is applied to text classfication on slides 81–117.

The third main type of linear classifier we covered is the linear discriminant (slide 118). We focused on the Perceptron, covered in slides 120–129.

4 Supervised Learning – Neural Networks (1)

Please note: slide numberings refer to the updated version of the slides dated June 14th, 2017. Aims for this lecture are a continuation of the previous two lectures, on regression and classification, since Neural Networks can be used for both types of learning. So the first two aims on slide 3 were covered in the previous lecture on linear classification, and the third aim (the dual version of perceptron training) was held over to the lecture on Kernel Methods (discussed below). Therefore here we cover just the last three aims (4–6), focusing on the classic method of error back-propagation to train the multi-layer perceptron.

(1) define the perceptron
(2) describe the perceptron training algorithm
(3) describe the dual version of perceptron training
(4) reproduce the method of gradient descent for linear models
(5) define the problem of non-linear models for classification
(6) describe back-propagation training of a multi-layer perceptron neural network for non-linear classification

In this lecture we start with the so-called *linear unit* which is essentially a linear model as we saw in the lecture on linear regression, although trained differently, and it is the basic building block of multi-layer perceptrons. You should satisfy yourself that you get each of the steps in the derivation of gradient descent training for a linear unit on slides 11–14. Be clear also on the how this fits into the algorithm on slide 15, and how a small change in this algorithm changes learning from batch mode to incremental or stochastic gradient descent (slides 17–18).

The field of neural networks is vast and a full treatment would take (at least) one whole course. In this lecture we simply develop the core idea behind the classic method of error back-propagation to train the multi-layer perceptron, which is to replace the thresholding *activation function* of the perceptron with a *differentiable* output, in this case the sigmoid function (other functions are more widely used in practice). This is presented on slides 22–23 (an example of the non-linear decision boundaries that can be learned by a multi-layer network of sigmoid units is on slides 20–21). A gradient descent training rule for each unit such networks can be derived by taking partial derivatives in a similar way to what was done to obtain a gradient descent training rule for a linear unit. This is shown in outline on slides 24-25, although some details of the full derivation to obtain the back-propagation training algorithm on slide 26 for the entire network are omitted.

5 Tree Learning

Please note: slide numberings refer to the updated version of the slides dated June 15th, 2017. Aims for this lecture are a continuation of the previous two lectures, on regression and classification, since Neural Networks can be used for both types of learning. So the first two aims on slide 3 were covered in the previous lecture on linear classification, and the third aim (the dual version of perceptron training) was helpd over to the lecture on Kernel Methods (discussed below). Therefore here we cover just the last three aims (4–6), focusing on the classic method of error back-propagation to train the multi-layer perceptron.

(1) define the decision tree representation
(2) list representation properties of data and models for which decision trees are appropriate
(3) reproduce the basic top-down algorithm for decision tree induction (TDIDT)
(4) define entropy in the context of learning a Boolean classifier from examples
(5) describe the inductive bias of the basic TDIDT algorithm
(6) define overfitting of a training set by a hypothesis
(7) describe developments of the basic TDIDT algorithm: pruning, rule generation, numerical attributes, many-valued attributes, costs, missing values
(8) describe regression and model trees

For aim (1) we see on slides 9–13 that in decision tree learning the representation is quite powerful; for discrete-valued attributes it is a disjunction of conjunctions, and for continuous attributes it is a recursive sub-division into decision regions. For aim (2) such a model class is appropriate for problems with certain characteristics – see slide 14. You should know (aim 3) the TDIDT algorithm on slide 15 which is the core decision tree learning algorithm, also known as "ID3", and used as the basis of nearly all tree learning algorithms, e.g., C4.5, CART, J48, C5/SEE5, etc. You will need to understand (aim 4) entropy and its use in the information gain measure for selecting attributes to split on – see slides 16-31.

The inductive bias of decision tree learners is captured in the description of hypothesis space search by the TDIDT algorithm on slides 35-39. You will need to know this for aim (5) since it applies to tree learning in general. It's basis in the Occam's Razor principle is discussed on slides 40-41; this is an often-used idea in many areas of machine learning; we saw a version of this idea in the context of Bayesian learning with the Minimum Description Length (MDL) principle in the lecture on linear classifiers. Overfitting (aim 6) is described for decision trees and machine learners in general on slides 42–46.

For aim (7) you should cover: pre- and post-pruning and their issues(slides 47-50), the general method of pruning or model regularisation using a validation set as shown by reduced-error pruning (slides 51–53), error-based pruning for sub-tree replacement (slides 54–55 and 57–61) but you don't have to memorize the formula on slide 59, although you should understand the ideas behind it). and rule-based pruning (slides 51–52 and 61–62). The remainder of points for aim (7) can be regarded as background (i.e., how in decision tree learning to handle: continuous-values attributes (slides 66–67), many-valued attributes (slides 69–71), missing (i.e., unknown) attribute values on slide 75). **Note**: this re-statement differs slightly from the aims at the start of the lecture notes in that it does not include *costs*.

For aim (8) on regression trees and model trees, the key aspects are covered on slides 78–91. In particular, you should know the basic representation and properties of both (slides 79–82 for regression trees and 88–90 for model trees). The remainder of the slides covering smoothing the prediction, etc., can be treated as additional background on regression and model tree learning.

6 Supervised Learning – Kernel Methods

The aims of this lecture are to be able to:

(1) outline non-linear classification with kernel methods
(2) describe learning with the dual perceptron
(3) outline the idea of learning in a dual space
(4) describe the concept of maximising the margin in linear classification
(5) describe the method of support vector machines (SVMs)
(6) describe the concept of kernel functions
(7) outline the idea of using a kernel in a learning algorithm

For aim (1) the kernels of the lecture title mainly refer to Support Vector Machines (SVMs), a type of learning algorithm based on defining a kernel which implicitly defines a model class or "hypothesis space". These have a number of interesting and useful properties, but the most well-known is their ability to extend linear classifiers to non-linear classification simply by a change of kernel function. We start by reviewing linear classifiers then motivate the need for SVMs on slides 4–11. The ability for SVMs to do non-linear classification via kernels using a function on dot products is called the “kernel trick” and is introduced on slides 15–16.

That we can obtain a learning algorithm able to exploit the kernel trick follows from the observation that some learning algorithms, such as the Perceptron, can be seen as operating in a “dual space”, since all weight updates that occur during learning are expressible in terms of adding or subtracting feature values from misclassified training instances. Thus we can derive a “dual form” of Perceptron training on slides 12–13 which covers aim (2). For aim (3) it suffices to see that in this dual space the Perceptron (and any linear classifier of the same form) can be expressed solely in terms of dot products between an instance to be classified and the training examples – this is the idea of learning in a dual space, where all the dot product information is captured in the *Gram matrix* (slide 12).

The question is, can we do better than perceptron learning ? SVMs are formulated as an answer to that question, where the approach is to set up the learning objective of finding a separating hyperplane that maximises the *margin*. How this is set up is aim (4), which is covered on slides 14 and 17–22, showing the “geometric perspective” on finding the weight vector that maximises the margin.

In terms of support vector machines (aim 5), the learning problem is then defined as a quadratic constrained optimization that can be solved using the method of Lagrange multipliers, outlined on slides 22–25. The interesting thing here is that this optimization is solved in its *dual* form, and we can observe in the expression on slide 25 that this essentially requires only the information from the data captured in the form of dot products, as for the dual Perceptron. Although in a real SVM this would be implemented by calling an optimization package, it is possible to set up the Gram matrix, determine the expression for the parameters α and find the solution by taking the partial derivatives and setting to zero. You should be familiar with the steps in this method, shown for the example on slides 26–28.

Aim (6) on the use of kernel function with dot products is explained with reference to some commonly used kernels on slides 41–52.

We leave as background issues such as handling noise using soft margins, sparse data (mentioned on slides 30–40), and the applications referred to on slides 53–54.

7 Unsupervised Learning – Dimensionality Reduction and Clustering

The slides for this lecture were divided into two parts: on dimensionality reduction, and clustering. However, we will treat them in this note as a single set and use the prefix ’DR’ to refer to slides on dimensionality reduction and ’CL’ for slides on clustering.
The aims of this lecture are to be able to:

(1) describe the problem of dimensionality reduction
(2) outline the method of Principal Component Analysis
(3) describe the problem of unsupervised learning
(4) outline a number of frameworks for unsupervised learning

(5) describe k-means clustering
(6) describe the role of the EM algorithm in k-means clustering
(7) describe hierarchical clustering
(8) outline methods of evaluation for unsupervised learning

Aim (1) is covered on slide DR 7. The motivation for dimensionality reduction is that although a dataset may be high-dimensional, like text documents with a large vocabulary of words or images with many pixels, the actual underlying structure of the data may be of lower dimensionality, like a small set of topics in documents.

For aim (2) Principal Component Analysis (PCA) is the best-known method of dimensionality reduction and is outlined on slides DR 8–12.

Aim (3) is covered on slides DR 4–6 and CL 2–3.

The main framework for clustering for aim (4) is covered on slides CL 4–9. Some variant frameworks are on slides CL 48–60, including frameworks such as conceptual clustering which is a form of Bayesian approach to clustering, and semi-supervised learning, but since these were not given sufficient time during the lecture they are left as background.

The k-means clustering algorithm of aim (5) is covered in slides CL 10–20. The figure on slide CL 11 shows an example where the data is for a single variable and there are two (normal) distributions, A and B, for which the parameters need to be identified. The data is shown above the graph. You should satisfy yourself that you can see how the parameters relate to the data, particularly in this case finding the two means ($k = 2$). Slide CL 12 just gives the notation for the algorithm on slide CL 13, and slides CL 14–15 are a representation of the algorithm applied to a 3-means problem. Slides CL 16–19 relate to the importance of supplying the "correct" value of k and what can happen if the assumptions of the algorithm are not met, particularly that a form of "overfitting" can occur. By the way, make sure you don't confuse k-means (clustering) with k-nearest neighbour (classification) !

Aim (5) covers the use of the Expectation Maximisation (EM) algorithm in k-means clustering (slides CL 21–29). The EM algorithm was developed to enable values to be estimated for "hidden" or missing variables — in this, there are k missing variables, i.e., the centroids of the k clusters. However, since coverage of this during the lecture was limited it should also be treated as background.

The second main approach to clustering we looked at is the subject of aim (7). Hierarchical clustering is covered in slides CL 36–47. A visual representation of different versions of the cluster distance formula is on slide CL 39. The output of hierarchical clustering is a cluster *tree* or dendrogram (slides CL 41 and CL 45–46). Since hierarchical clustering works from a dissimilarity matrix, you can choose to cluster either on the rows (instances) *or* columns (attributes) of a data set. This is shown on slide CL 47 for some biological (gene expression microarray) data.

Due to lack of time in the lecture, aim (8) on how the quality of clustering can be evaluated was only mentioned on slide CL 36, so is left as background.

8 Algorithm-Independent Aspects of Machine Learning

The aims of this lecture are to be able to:

(1) describe a basic theoretical framework for sample complexity of learning
(2) define the Probably Approximately Correct (PAC) learning framework
(3) define the Vapnik-Chervonenkis (VC) dimension framework
(4) define the Mistake Bounds framework and apply the WINNOW algorithm
(5) apply basic analysis of learning in each of PAC, VC & Mistake Bounds frameworks
(6) outline the "No Free Lunch" and "Ugly Duckling" Theorems

This lecture is on theoretical aspects of machine learning and, unfortunately, there are some definitions and results which should be retained. However, there will only be a few of these, and in any case, the best way to recall these is always to ensure you understand the basic concepts underlying the theory. That way you will always be on the right track, even if you forget some of the minor details, like the constants involved. More importantly you will know the *point* of the theory and how it may be applied.

The area of computational learning theory is *algorithm-independent* in the sense of characterising learning in terms of its general properties (some of the possible ones are on slides 6–8). One of these is sample complexity — slides 9–10 give the basic properties of this framework within the setting of concept learning — which covers aim (1) and is the setting for the PAC approach of aim (2). First we recall the definition of true error (slides 11–13), which we saw before when discussing overfitting, in particular the difference between training error (sometimes referred to as sample error) and true error. This links to aim (2), since PAC learning is investigated in this course as a method of bounding the true error given training error. However, first we need to describe the framework of concept learning within which PAC learning is introduced (slides 14–18). The theorem bounding sample complexity in the PAC sense is worked through on slides 19–22. You should be able to recall the formula on slide 22 for bounding the sample complexity of a consistent learner in terms of ϵ, δ and the size of the hypothesis space, even if you just remember the role of the terms and their proportionalities. Examples of applying this to two hypothesis spaces are on slides 23–26. The definition of PAC learning is on slide 27 – you don't need to memorise this, just understand the general principle, i.e., be clear on the components of the definition and what is actually being defined ! Lastly, it is easy to derive the result of exponential sample complexity for unbiased learning of Boolean function (slide 29) which quantifies the need for an inductive bias of some kind for a learning algorithm to be able to generalise effectively from a training dataset.

For aim (3) we introduced the VC dimension as a way of measuring hypothesis space size, addressing the issue of model complexity (slides 30-40). As mentioned in the lecture this is a complicated issue, although the VC dimension can be understood in terms of *dichotomies* and *shattering*. This gives a method to work out a lower bound on the VC dimension. Note the Venn diagram on slide 33 in which *all* subsets of instances are represented. A hypothesis space corresponding to this diagram would be one in which there was a hypothesis for each subset of instances, i.e., no *restriction* (language) bias. We explained that the VC dimension of the class of linear decision surfaces (slides 38–40) is $n - 1$ where n is the number of dimensions (features or attributes). Following this working out we see that the VC dimension of the hypothesis space can be used in a version of the PAC learning result on sample complexity of ϵ-exhausting the version space, although you are not required to memorise the formula on slide 41 !

Another way to attack the idea of complexity or "how hard is it to learn a concept class" is to investigate the learning curve or prediction ability of a learning algorithm as it receives training examples.

To do this we concentrated on the mistake bounds analysis of concept learning algorithms.

For aim (4) we took the actual treatment of mistakes in the context of concept learning and derived some bounds for algorithms. The setting for mistake bounds analysis in the lecture is on slide 42.

WINNOW, on slides 47–49, is a linear classification algorithm that, like the Perceptron, learns in a mistake-driven way. However, unlike the Perceptron the WINNOW algorithm is able to quickly drive down the weights of irrelevant features or attributes, i.e., those that do not affect the classification. In fact, it's mistake bound is only at worst logarithmic in the number of irrelevant attributes.

Two simpler concept learning algorithms that also learn in a mistake-driven way are FIND-S and the HALVING ALGORITHM on slides 50–56. However, it easier to derive mistake bounds for these algorithms than WINNOW and the Perceptron easy to derive. Ideas from both WINNOW and the HALVING ALGORITHM are combined in the influential WEIGHTED MAJORITY algorithm on slides 58–60. The importance of this algorithm is that it is an ensemble learning algorithm which learns the best *weighted combination* of individual classifier learning algorithms with a Winnow-type logarithmic mistake bound.

Aim (5) was covered in the preceding discussion of aims (2)-(4).

Aim (6) represents a more statistical view on algorithm independent learning. The "No Free Lunch" theorem is a part of learning theory in that it addresses the question of whether we can say one learning method is generally better than another. Slides 63–71 cover this, and you just need to be clear on what the "No Free Lunch" results mean (they don't mean that learning can't work !). The related result known as the "Ugly Duckling Theorem" is more subtle, and is left for background.

9 Ensemble Learning

Assessable learning objectives:

1. describe the framework of bias-variance decomposition
2. define the method of bagging
3. define the method of boosting

The aims of this lecture are to be able to:

(1) describe the framework of the bias-variance decomposition and some of its practical implications
(2) describe how ensembles might be used to address the bias and variance components of error
(3) outline the concept of the stability of a learning algorithm
(4) describe the ensemble methods of bagging, random forests and boosting
(5) compare the operation of these methods in terms of the bias and variance components of error

The bias-variance decomposition (aim 1) extends the analysis from the previous lecture of how the representation and algorithm selected for a learner, i.e, the class of models or hypotheses it can learn, can affect its expected error. The bias-variance decomposition is reviewed on slide 5 and a diagram showing the 4 cases of low/high bias/variance is on slide 6. An example for regression-type models is shown on slides 7–9, where it was shown how issues of model complexity can be analysed in these terms. The trade-off of bias and variance is described by way of an example on slides 11–14. This shows that more data is required for low bias learners than for high bias learners (essentially, to avoid overfitting) but that this come with the penalty of more computation, since low bias learners, needing to fit more complex models, typically require greater runtimes (and also memory).

Following from this idea is the concept of stability (aim 3) which is closely related to the bias-variance error properties of an algorithm. Some learners are unstable (i.e., small changes in the training set cause large changes in the off-training set error) which is reflected in a high variance component of their error. This is discussed on slides 15–21, where it is noted that it is not sufficient simply to characterise an *algorithm* as stable or not, without taking into account also the effect of varying some of its parameters. This can be seen even when varying the single parameter of k in kNN (slide 20).

Aim (2) addresses how the bias-variance decomposition can be exploited to reduce bias or variance (or both) using ensemble methods. The ensemble method of bagging (aim 4) was introduced to reduce the problem of variance (slides 24–36). Recall that bagging is based on the statistical method of bootstrap sampling with replacement from the training data. The use of decision trees or other *unstable* methods as the base learners is important to bagging as these can have a relatively high variance component in their prediction error. Random forests (Aim 4) extend the idea of bagging with randomization (slides 37–39). Boosting (aim 4) is also an ensemble method but this derives from PAC learning rather than statistics. The algorithm AdaBoost.M1 applies *weighted* voting of the base classifiers in the ensemble. The weights are adjusted by multiplying by a factor based on the error, with the effect that correctly classified instances are gradually down-weighted leading the ensemble to iteratively re-focus on learning the remaining “hard-to-classify” instances. Boosting is covered on slides 40–56. For each of these ensemble methods the affect on bias-variance (aim 5) is covered in their presentation.

The remainder of the lecture covers more ensemble methods — these are left as background and will not be assessable.

10 Remaining Lectures

The lecture on probabilistic graphical models in Week 12 and the guest lecture in Week 13 are not assessable.