Machine Learning (COMP30027) Summary Notes
Akira Wang

1 Basics of Machine Learning

1.1 Terminology

1. Instances (exemplars) are rows of a data set
2. Attribute (features) are the columns of a data set
3. Concepts (class labels) are things we aim to learn or predict from a data set
4. N training instances, C classes, D attributes

1.2 Quantities

1. Nominal quantities (categorical or discrete) have no relation between labels. *Example: sunny, hot, rainy.*
2. Ordinal quantities (categorical or discrete) have an implied ordering between labels. *Example: (cold, mild, hot) have an ordering (cold < mild < hot).*
3. Continuous quantities are real-valued attributes with a defined zero point, and have no explicit upper bound

1.3 Methods

1. **Supervised** methods have prior knowledge to a closed set of classes, and us that to predict new values.
2. Unsupervised methods will dynamically discover classes without any knowledge of class labels, and will train itself to categorize instances mathematically.

1.4 Entropy (Information Theory)

Entropy is a measure of **unpredictability** on the information required to predict an event. The entropy *in bits* of a discrete random variable X is defined as:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \Pr(x_i)\log_2(\Pr(x_i)) \tag{1}$$

where $\mathrm{H}(X) \in [0, \log_2(n)]$

1. A **high** entropy value means X is **unpredictable**. Each outcome gives *one bit* of information.
2. A **low** entropy value means X is more **predictable**. We don't learn anything once we see the outcome.

2 Naive Bayes

2.0.1 Hyperparameters

1. The choice of smoothing method
2. *Optional* (The choice of distribution to model the features).

2.0.2 Parameters

1. The prior and posterior probabilities.
2. Size $= \mathcal{O}(|C|+|C||FV|)$, where C is the set of classes and FV is the set of feature-value pairs.

2.0.3 Interpretation

1. Based on the most *positively* weighted features associated with a given instance.

2.1 The Naive Bayes Implementation

The Naive Bayes assumes that **all** probabilities are independent. This means that for every class j:

$$\Pr(x_1, x_2, \ldots, x_n | c_j) \approx \prod_{i=1} \Pr(x_i | c_j). \tag{2}$$

This is called the **conditional independence assumption**, and makes Naive Bayes a tractable method.

Naive Bayes works by calculating a set of prior probabilities of classes $\Pr(\text{class} = c)$, and posterior probabilities of attributes given a class $\Pr(\text{attribute} = a | \text{class} = c)$. Then for every test instance, it will calculate the posterior and assign it the largest corresponding prior probability.

$$\text{Prediction} = \Pr(\text{class} = c) \prod_{i=1} \Pr(\text{attribute} = a_i | c) \tag{3}$$

2.2 Probabilistic Smoothing

Since Naive Bayes predicts using a product of probabilities, any probability of 0 results in an overall 0. This means that any unseen event becomes *impossible*, which is untrue.

2.2.1 Epsilon

One method to combat this is to assume that **no event is impossible**, i.e. every probability is greater than 0. This is implemented by assigning a 0 probability with some value $\epsilon \to 0$, where the condition $1 + \epsilon \approx 1$.

2.2.2 Laplace Smoothing / add-one or add-k

The better alternative is add-k. Laplace smoothing essentially gives any unseen events a count of 1 (or a value $k \in (0, 1)$). Then, all counts are increased to ensure that the monotonicity is maintained.

Let $V =$ Number of attributes. Then for every class j, and attribute value i:

$$\hat{Pr}(a_i, c_j) = \frac{1 + freq(a_i, c_j)}{V + freq(c_j)}. \tag{4}$$

2.3 Missing Values

1. If a value is missing in a **training** instance, then it is possible to simply have it not contribute to the attribute-value count estimates for that attribute.
2. If a value is missing in a **test** instance, then you may just ignore the attribute for the purposes of classification.

2.4 Analysis of NB

1. Works quite fast for most data sets (provides an adequate benchmark)
2. Complexity: $\mathcal{O}(CD + C)$

3 Evaluation

3.1 Accuracy

The basic evaluation metric is *Accuracy*, given as:

$$\text{Accuracy} = \frac{\text{Number of correctly predicted labels}}{\text{Total number of test instances}}. \tag{5}$$

3.2 Evaluation Strategies

3.2.1 Holdout

1. Each instance is randomly assigned as either a training instance or test instance.
2. The data set is partitioned with **no** overlap between data sets
3. You build a model based on the training instances, and evaluate the trained model with the test instances.
4. Popular train-test splits are: 50-50, 80-20, or 90-10

Advantages:

1. Simple to work with and to implement.

2. High reproducibility.

Disadvantages:

1. The split ratio affects the estimate of the classifiers behaviour.
2. Lots of test instances with few training instances may leave the model to build an inaccurate model.
3. Lots of training instances with few test instances leaves the model to be accurate, but the test instances may not be representative of future test instances.

3.2.2 Repeated Random Sub-sampling

Works similar to holdout, but over several iterations.

1. New train-test sets are randomly chosen each iteration.
2. The size of train-test split is fixed across all iterations.
3. A new model and evaluation is done every iteration.

Advantages:

1. Average holdout method tends to produce more reliable results.

Disadvantages:

1. Slower than the holdout method (by a constant factor of number of iterations)
2. A wrong choice of the train-test split ratio can lead to highly misleading results

3.3 k-Fold Cross-Validation

The usual preferred method of evaluation. The data set is progressively split into a number of m partitions, where:

1. One partition is used as test instances.

2. The other $m - 1$ partitions are used as training instances.
3. The evaluation metric is aggregated across the m partitions by taking the average.

This is much better than holdout / repeated random sub-sampling since:

1. Every instance is a test instance (for some partition)
2. Takes roughly the **same time** as repeated random sub-sampling
3. Can be shown to **minimise the bias and variance** of our estimates of the classifier's performance

3.3.1 Choosing values of m

1. Small m: More instances per partition, but **more variance** in performance estimates
2. Large m: Fewer instances per partition, but **less variance** at the sacrifice of time complexity
3. *A usual default value is to use* $m = 10$ *which mimics a 90-10 holdout strategy.*

3.3.2 Stratification

Stratification is a type of inductive learning hypothesis. Any hypothesis found to approximate the target function over a large training data set will also approximate the target function well over any **unseen** test examples.

However, machine learning suffers from **inductive bias** meaning assumptions must be made about the data to build a model and make predictions.

Stratification assumes that the class distribution of unseen instances will share the same distribution of seen instances. **This is not true for any Machine Learning problem**.

3.4 Classification Accuracy

3.4.1 Terminology

1. True Positive (TP) are instances where we predicted a label to be A, and the actual label was A.

2. False Positive (FP) are instances where we predicted a label to be B, but the actual label was A.

3. False Negative (FN) are instances where we predicted a label to be A, but the actual label was undefined ?.

4. True Negative (TN) are instances where we predicted a label to be undefined ?, and the actual label was undefined ?.

We only want **True Negative** and **True Positives**.

A classifier may then classify:

1. Interesting Instance as I if it is TP

2. Interesting Instance as U if it is FN

3. Uninteresting Instance as I if it is FP

4. Uninteresting Instance as U if it is TN

3.4.2 Classification Accuracy

Not to be mixed up with Accuracy. Classification Accuracy is the proportion of instances for which we have correctly predicted the label, given as

$$\text{Classification Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}. \tag{6}$$

3.4.3 Error Rate

The Error Rate of a classifier is given as:

$$\text{Error Rate} = 1 - \text{Classification Accuracy}. \tag{7}$$

3.4.4 Precision and Recall

1. **Precision**: How often are we correct when we predict a label. This is given as:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{8}$$

2. **Recall**: What proportion of the predictions have we correctly predicted. This is given as:

$$\text{Precision} = \frac{TP}{TP + FN}. \tag{9}$$

High precision gives low recall, and high recall gives low precision. Since we want both precision and recall to be high, we can evaluate this using an **F-Score**.

$$F_\beta = \frac{(1 + \beta^2)2 \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{10}$$

$$F_1 = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{11}$$

3.4.5 Macro and Micro Averaging

Over multi-class data sets, you must calculate the Precision / Recall / F-Score **per class**, and must be averaged across c classes.

1. **Macro-Averaging** takes the mean of all Precision / Recall:

$$\text{Precision}_M = \frac{\sum_{i=1}^{c} \text{Precision(i)}}{c} \tag{12}$$

$$\text{Recall}_M = \frac{\sum_{i=1}^{c} \text{Recall(i)}}{c} \tag{13}$$

2. **Micro-Averaging** combines all test instances into a single pooled Precision / Recall:

$$\text{Precision}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FP + i} \tag{14}$$

$$\text{Recall}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FN + i} \tag{15}$$

3.5 Baseline and Benchmarks

3.5.1 Random Baseline

1. Randomly assign a class to each test instance.

2. randomly assign a class c_k to each test instance and weight the class assignment according to $\Pr(c_k)$ (this assumes we know class prior probabilities).

3.5.2 Zero-R / Majority Class

Essentially classifies all test instances as the most common class in the training set.

Complexity: $\mathcal{O}(0)$

3.5.3 One-R / Decision Stump

Creates a single rule for each attribute in the training data, then selects the rule with the **smallest error rate** as its "one rule".

Complexity: $\mathcal{O}(1)$

4 Decision Trees

Decision trees are created in a recursive divide-and-conquer fashion. We want the **smallest tree** which minimizes the errors.

4.0.1 Hyperparameters

1. The choice of function used for attribute selection
2. The convergence criterion

4.0.2 Parameters

1. The decision tree itself
2. Size = $\mathcal{O}(|FV|)$, where FV is the set of feature-value pairs.

4.0.3 Interpretation

1. Based directly on the path through the decision tree.

4.1 Information Gain

Information Gain is the expected **reduction** in entropy caused by knowing the value of an attribute. Compare:

1. The entropy before splitting the tree using the attribute's values
2. The weighted average of the entropy over the children after the split (referred to as **Mean Information**)

If the entropy **decreases**, then we have a better tree that is **more predictable**.

Information Gain also tend to prefer highly-branching attributes:

1. A subset of instances is more likely to be homogeneous (all of a single class) if there are only a few instances.
2. Attributes with many values will have fewer instances at each child node.

These factors may result in **over-fitting or fragmentation**.

4.1.1 Mean Information

We can calculate the mean information for a decision tree stump with m attribute values as:

$$\text{Mean Info}(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{m} \Pr(x_i)\mathrm{H}(x_i). \tag{16}$$

4.1.2 Gain Ratio

The Gain Ration reduces the bias for Information Gain towards highly branching attributes by normalizing relative to the Split Information. The Gain Ratio can be calculated as:

$$\mathrm{GR}(R_A, |R) = \frac{\mathrm{IG}(R_A|R)}{\mathrm{H}(R_A)}. \tag{17}$$

4.1.3 Split Ratio

The Split Ratio or **intrinsic value** is the *entropy* of a given split (evenness of the distribution of instances to attribute values).

4.2 Stopping Criteria

The ID3 DT algorithm is defined in a way such that:

1. The Info Gain / Gain Ratio allows us to choose the seemingly better attribute at a given node.
2. It is an approximate indication of how much absolute improvement we expect from partitioning the data according to the values of a given attribute.
3. An Info Gain of 0 or close to 0 means that there is no improvement and can often be pruned.
4. A typical modification of ID3 is to choose the best attribute given it is greater than some threshold τ.

4.3 Analysis of ID3 DT

1. Highly regarded among **basic** supervised learners.
2. Fast training and testing time complexities.
3. Susceptible to the effects of irrelevant attributes.
4. Complexity: $\mathcal{O}(D)$

There are also some alternative DTs:

1. **Oblivious Decision Tree**: Requires the same attribute at every node in a layer.
2. **Random Tree**: Uses samples of the possible attributes at any given node. This helps to reduce the effects of irrelevant attributes a basis of a DT variant called the **Random Forest**.

5 Instance Based Learning

5.1 Feature Vectors

A feature vector is an n dimensional vector of features that represent an object.

Since vector coordinates are a point in an orthogonal n-space, the angle of the vector in that n-space is determined by the relative weight of each m attributes.

Similarity:

1. A numerical measure of how similar two vectors are.
2. Higher measure for closer similarity.

Dissimilarity:

1. A numerical measure of how different two vectors are.
2. **Lower** measure for **closer** similarity, likewise higher measure for more difference.
3. Minimum dissimilarity is often 0, with a varying upper limit

5.2 Distance Metrics

5.2.1 Hamming Distance

The Hamming Distance is given as

$$d_{\mathrm{H}}(A, B) = \sum_{i=1}^{n} [0 \text{ if } a_i == b_i \text{ else } 1]. \tag{18}$$

5.2.2 Euclidean Distance

The Euclidean Distance is given as

$$d_E(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}. \tag{19}$$

5.2.3 Manhattan Distance

Given two items A and B, and their feature vectors $\mathbf{a}$ and $\mathbf{b}$, we can calculate their similarity via their distance d based on the absolute differences of their Cartesian coordinates

$$d_M(A, B) = \sum_{i=1}^{n} |a_i - b_i|. \tag{20}$$

5.2.4 Cosine Similarity

Given two items P and Q, and their feature vectors $\mathbf{p}$ and $\mathbf{q}$, we can calculate their similarity via their vector cosine:

$$cos(P, Q) = \frac{\mathbf{p} \cdot \mathbf{q}}{|\mathbf{p}||\mathbf{q}|}. \tag{21}$$

5.3 k-Nearest Neighbours

The nearest neighbour is defined as the closest object from your object, using a specified distance metric. In classification, we give class assignments of existing data points, and classify them according to the k nearest neighbours.

5.3.1 Hyperparameters

1. k (The neighbourhood size)
2. Distance / Similarity metric
3. Feature weighting / Selection

5.3.2 Parameters

1. None. k-NN is lazy and doesn't abstract away from the training instances.

5.3.3 Interpretation

1. Relative to the training instances that give rise to a given classification and their geometric distribution.

5.3.4 1-NN

Classifies the test input according to the class of the closest training instance.

5.3.5 k-NN

Classifies the test input according to the **majority** class of the k nearest training instance.

A **weighted variant** of k-NN will classify the test instances according to the weighted accumulative class of the k nearest training instances, where the weights are based on similarity of the input to the each of k neighbours.

5.4 Weight Strategies

5.4.1 Majority Class

Gives each neighbour equal weighting and classifies according to the majority class of set of neighbours.

5.4.2 Inverse Distance

Weights the vote of each instance by taking the inverse and adding some value ϵ.

$$w_j = \frac{1}{d_j + \epsilon} \tag{22}$$

5.4.3 Inverse Linear Distance

Weights the vote of each instance by taking the inverse and its "ranking" into account.

$$w_j = \frac{d_k - d_j}{d_k - d_1} \tag{23}$$

where d_1 is 1st the nearest neighbour, and k is the furthest neighbour.

5.4.4 Analysis of k-NN

A typical implementation involves a brute-force computation of distances between a test instance, and to every other training instance.

For N training instances, and D dimensions, we end up with $\mathcal{O}(DN)$ performance. This means for small data sets, the performance is fast but becomes infeasible as N grows.

This is because

1. The model build by NB or a DT is generally much smaller than the number of training instances in the data set.

2. The model built by k-NN is also the data set itself, all calculations are done during run time.

We also try to to use odd values for k since the distribution of classes could be split equally. Complexity: $\mathcal{O}(ND + k)$

6 Support Vector Machines

6.0.1 Hyperparameters

1. The penalty term C/ϵ for soft-margin SVMs.
2. Feature value scaling.
3. The kernel function used.

6.0.2 Parameters

1. Vector of feature weights + bias term.
2. Size = $\mathcal{O}(|C||F|)$ assuming a one-vs-rest SVM, where C is the set of classes and F is the set of features.

6.0.3 Interpretation

1. The absolute value of the weight associated with each non-zero feature in a given instance provides an indication of its relative importance in classification.

6.1 Nearest Prototype Classification

6.1.1 Hyperparameters

1. Distance / Similarity metric used to calculate the Prototype and distance to each prototype in classification
2. Feature weighting / Selection

6.1.2 Parameters

1. The prototype for each class
2. Size = $\mathcal{O}(|C||F|)$, where C is the set of classes and F is the set of features.

6.1.3 Interpretation

1. Relative to the geometric distribution of the prototypes and the distance to each prototype for a given test instance.

To find the Nearest Prototype, we calculate the centroid (mean of all train instances) of each class.

TO classify, we find calculate a distance metric between the test instance and prototype, and assign the label of the closest prototype as our prediction.

Complexity: $\mathcal{O}(CD)$

6.2 Margins and Kernel Functions

6.2.1 Maximum Margin

How can we rate the different decision boundaries to work out which is *best*?

1. For a given training set, we would like to find a decision boundary that allows us to make all correct and confident predictions on the training examples.
2. Some methods find a separating hyper-plane, but not an optimal one. SVM's **do** converge to an optimal solution.
3. SVM's maximize the distance between the hyper-plane and the *difficult fringe points* which are close to the decision boundary.

6.2.2 Soft Margin

A possibly large margin solution is *usually* better even though the constraints are being violated. This permits some points to be on the *wrong* side of the hyper-plane.

If we suspect the data is *nearly* linearly separable, then using a soft margin is better.

Slack variables can be introduced to allow some variables to be on the wrong side of the hyper-place at some cost (there's a modified function that we don't need to know about).

6.2.3 Kernel Function

To obtain a non-linear classifier, we can transform our data by applying a mapping function Φ such that a linear classifier can be used on the new feature vectors.

If a data set is deemed to be non linearly separable, a kernel function is better than a soft margin since they will end up producing a spurious (non related) margin. A kernel function can transform exponential data sets into a linear data set by applying a log transform.

6.3 Lagrange Multipliers are not in the scope of this subject lmao

6.4 Multiple Class SVM

Since SVM's are inherently two-class classifiers most common approaches to extending to multiple classes include:

1. **One vs All**: Classification chooses one class which classifies test data points with the greatest margin.
2. **One vs One**: Classification which chooses class selected by the most number of classifiers.

6.4.1 Analysis of SVM

1. SVM's are a high-accuracy *margin* classifier.
2. Learning a model means finding the best separating hyper-plane.
3. Classification is built on projection of a point onto a hyper-plane normal.
4. If the data set is non linear then a kernel function may be used to transform it into a linear data set.
5. SVM's have several parameters that need to optimised and may end up being slow.

6. Complexity: $\mathcal{O}(C^2D + C^2)$

Geometric classifiers assume all attributes are equally important and calculate some similarity / distance metric. For example, if the distance metric between useful attributes are small but the distance metric between non-similar attributes are large, they will outweigh the classification and deem them *non similar*.

SVMs solve this issue by finding optimal parameters (weights) for each attribute so that the basis of prediction is more meaningful.

7 Discrete and Continuous Data

7.1 Discretisation

Discretisation is the translation of continuous attributes onto nominal attributes (also known as *binning*).

The process is performed in two steps

1. Decide how many values to map the feature onto (equal-width, equal-frequency, k-Means).
2. Map the features such that they are in their respective ranges

$$\{(x_0, x_1], (x_1, x_2], \ldots, (x_{n-1}, x_n)\} \tag{24}$$

(this is the ML notation for binning!)

Although it is simple to discretise attributes and in most cases, a viable solution, we end up with the loss of generality (no sense of *numeric proximity* or *ordering* which may result in over-fitting).

7.1.1 Unsupervised Discretisation

There are three main methods of Unsupervised Discretisation:

1. **Equal-Width**: Partition the range into n intervals and allocate each attribute value into its respective interval. Suffers from data sets with a tight group of values with outliers.

2. **Equal-Frequency**: Partition the number of values n into m groups and allocate each interval until there are m attribute values. For new instances added, use the median as a dividing point.

3. k**-Means**: Apply an unsupervised clustering algorithm.

   (a) Select k points at random to act as seed clusters.

   (b) Assign each instance to the cluster with the nearest centroid.

   (c) Compute the centroids of each cluster by taking the mean.

   (d) Repeat until the assignment of instances of the clusters converge to a stable state.

7.1.2 Supervised Discretisation

The idea is to *group* values into class-contiguous intervals.

1. Sort values and identify breakpoints in class memberships.

64	65	68	69	70	71	72	72	75
yes	no	yes	yes	yes	no	no	yes	yes

   The groups then become:

$$(64), (65), (68, 69, 70), (71, 72, 72), (75) \tag{25}$$

2. If there any ties, re-position any breakpoints where there is *no change* in numerical value.

$$(64), (65), (68, 69, 70), (71, 72, 72), (75) \tag{26}$$

Although it is very simple to implement, it usually creates *too* many categories which leads to over-fitting. To combat this, we apply the **group values approach**, where each category must have at least n instances of a *unique class*.

8 Feature Selection

8.1 Wrapper Methods

8.1.1 Naive Approach

A naive approach to wrapper methods is to choose subset of attributes that give the best performance on the development data (with respect to a single learner).

1. Feature sets will have optimal performance on development data.
2. Takes a very long time and therefore only practical for very small data sets.
3. Complexity: $\mathcal{O}(\frac{2^m}{6})$

8.1.2 Greedy Approach

A greedy approach would be to train and evaluate the model on each single attribute. Then, we choose the best attribute for each train-test until it converges to a model.

1. Converges mush quicker compared to the Naive Approach.
2. Usually converges to a **sub-optimal** model.
3. Complexity: $\mathcal{O}(\frac{m^2}{2})$

8.1.3 Ablation Approach

The ablation approach starts with all attributes, then removes an attribute each iteration until it diverges to a model.

1. Assumes all attributes are independent.
2. Complexity: $\mathcal{O}(m^2)$

Refer to Forward Selection, Backward Selection, or Step-wise Selection for Regression models.

8.2 Embedded Methods

To some degree, learners such as SVM, Logistic Regression, and DT's perform some form of feature selection within the training process. This is what we refer to as *embedded* methods.

However, it should be noted that these learners will still benefit with feature selection prior to building the model.

8.3 Filtering Methods

Exam! If there is a question like this, create a Contingency table for the attributes required to answer the problem for full marks.

8.3.1 Pointwise Mutual Information

Pointwise Mutual Information is one of several (retarded) metrics designed to assign a numerical value to its relevance to the model. (Just use AIC - Yaoban)

Attributes with the greatest PMI are most correlated with a class, and attributes with low PMI are more uncorrelated with a class.

Shorthand Notation:

$$A \to A = Y \tag{27}$$

where Y is an interesting value of a binary attribute.

The PMI can be calculated by

$$\text{PMI}(\text{attribute} = a, \text{class} = c) = \log_2\left(\frac{\Pr(a \cap c)}{\Pr(a)\Pr(c)}\right). \tag{28}$$

Think! Pointwise refers to a single point (i.e. an interesting attribute value to an interesting class label).

8.3.2 Mutual Information

Think! Mutual Information is the sum of all Pointwise Mutual Information for every attribute value to every class label.

(ML Notation: N training instances, C classes, D attributes)

$$\begin{aligned}\text{MI(Attribute, Class)} &= \sum_{i \in D}\sum_{j \in C} \Pr(i \cap j)\log_2\left(\frac{\Pr(a \cap c)}{\Pr(a)\Pr(c)}\right) \\ &= \sum_{i \in \{D\}}\sum_{j \in \{C\}} \Pr(i \cap j)\text{PMI}(i, j).\end{aligned}$$

8.3.3 Chi Squared is taught but d.o.f is outside the scope of this subject lmao

8.3.4 Mean and Variance

By finding the sample mean $\bar{x}$ of an random variable X, and its sample variance s^2, we can approximate a Normal distribution $X \sim N(\mu, \sigma^2)$. Specifically for Naive Bayes, we can estimate the probability of observing any given value with pointwise estimation (number of standard deviations away from the sample mean).

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{29}$$

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2. \tag{30}$$

9 Gradient Descent and Regression

9.1 Hyper-Parameter Optimisation

9.1.1 Grid Search / Exhaustive Search

We can iteratively search the parameter space for optimal values:

1. For each numerical θ_k:
2. Identify boundaries of the parameter range
3. Divide the range into linear or log steps

1. Hyper-parameter tuning is usually done during the final stage in an attempt to get higher accuracy with respect to the development data.
2. Since we are evaluating several models, there is also a risk of *over-tuning*. (The best choice of hyper-parameters for the development data may not be the best choice of hyper-parameters on the test data.)
3. When comparing models, you should use the same number of hyper-parameter search iterations for both.

9.2 Gradient Descent

Steps:

1. Make a prediction.
2. Compare prediction with true value.
3. Multiply with the corresponding attribute value.
4. Update all weights.

The logic behind Gradient Descent algorithms is to essentially measure the slope of the error function Error(θ) and minimize it.

If the gradient points up-hill, we will follow it down-hill. Each update will reduce the error term slightly.

With respect to the learning rate α (refer to AI definition of Gradient Descent), choose α which is not too small (slow) or too large (may miss the minimum).

Note that Gradient Descent is non-optimal since it may converge to a local minimum, but not the Error Function's true minimum. However, for Linear Regression, SS_{Res} is convex and therefore will only have one minimum.

9.3 Logistic Regression

Steps:

1. Take one class as a pivot (binary).
2. For every other class (binary), build the model compared to the pivot class.
3. (Resulting in $|C| - 1$ regression models)
4. Predict according to the highest scored model.

Logistic Regression is a type of probabilistic classification for binary data sets. The Logistic Function is defined as

$$\frac{1}{1 + e^{-(X\beta)}} \tag{31}$$

where β are the parameters of the model, and X is the design matrix of the model.

We choose a β that maximise the likelihood of observing our training instances (through Gradient Descent).

1. Positive $X\beta$ means the class is Y
2. Negative $X\beta$ means the class is N
3. Having $X\beta \approx 0$ means that it is very uncertain

In Logistic Regression, we model for every class j

$$\begin{aligned}\Pr(c_j|x_1, x_2, \ldots, x_D; \beta) &= \text{logistic}(X\beta) \\ &= \frac{1}{1 + e^{-(X\beta)}} \\ &= h_\beta(X).\end{aligned}$$

9.3.1 Analysis of Logistic Regression

You don't need to know how the math works.

1. Vast improvements on its counterpart Naive Bayes.
2. Suited to frequency based features (problems such as Neuro_linguistic Problems)
3. Slow to train and required feature scaling
4. Requires large N in order to be effective

10 Model Interpretability

Model interpretability is the means by which we can interpret the basis of a given model classifying an instance the way it does.

10.1 Error Analysis

Error analysis is the manual analysis of the sorts of errors that a given model makes.

1. Identify different *classes* of error that the system makes (relative to the predicted vs actual labels).
2. Hypothesising as to what has caused the different errors, and testing those hypothesis against the actual data.
3. Quantifying whether it is a question of data quantity / sparsity, or something more fundamental than that.

The starting point for error analysis is usually a **confusion matrix** and a **random sub-sample** of miss-classified instances. A good starting assumption is that a given *cell* in the confusion matrix forms a single error class.

Ensure that you test your hypotheses against your data since things which intuitively make sense, may not actually have any impact on the classification.

10.2 Hyperparameters and Parameters

A *hyperparameter* of a model is the set of parameters which can be adjusted to define the bias / constraints in the learning process.

A *parameter* of a classifier is the result of a given set of hyperparamters applied onto a particular training data set, and is then used to classify the test instances.

10.3 Dimension Reduction

Any **dimension reduction** method will be unable to faithfully reproduce to original data into a 2 or 3 dimensional reduced version.

Feature selection is one form of dimension reduction, but ideally we want to be able to reduce the dimensions in a way which is more faithful to the full feature set.

10.3.1 Principal Component Analysis

PCA will reduce the dimensions of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables *"the principal components (PCs)"* which are uncorrelated and ordered so that the first few retain most of the variation present in all of the original variables. PCA is generally performed using an eigenvalue solver.

11 Classifier Combination / Ensemble Learning

Classifier combination constructs a set of *base classifiers* from a given set of training data and aggregates the outputs into a single *meta-classifier*.

1. The combination of several *weak* classifiers can be at least as good as one strong classifier.
2. The combination of selection of *strong* classifiers is usually at least as good as the *best of the* base classifiers.

11.1 Voting

The simplest means of classification over multiple base classifiers is *voting*.

Essentially run the multiple base classifiers over the test data and select the class predicted by the most number of base classifiers (kind of like the majority class).

For a continuous class set, take the average over the numeric predictions of our base classifiers.

11.1.1 Instance Manipulation

Generate multiple training data sets through sampling, and train the base classifier over each train set.

11.1.2 Feature Manipulation

Generate multiple training data sets through different feature subsets, and train the base classifier over each train set.

11.1.3 Class Label Manipulation

Generate multiple training data sets by manipulating the class labels in a reversible manner.

11.1.4 Algorithm Manipulation

Semi-randomly adjust internal parameters within a given algorithm to generate multiple base classifiers over a given data set.

11.2 Stacking

Intuition: Smooth the errors over a range of algorithms with different biases.

1. Can use simple voting but assumes the classifiers have equal performance.
2. Train a classifier over the outputs of the base classifiers by using nested cross-validation to reduce bias.

Example:
Given a training data set (X, y):

1. Train the combination of base classifiers.
2. Predict the probability / label of the training instance.
3. Train the meta-classifier (Logistic Regression) over the predicted probability / label.

11.2.1 Analysis of Stacking

1. Mathematically simple but computationally expensive.
2. Able to combine classifiers with varying performance.
3. Generally yields in better results than the best of the base classifiers.

11.3 Bagging

Intuition: The more data we have, the lower the variance (by the CLT) and therefore the better the performance.

By constructing *novel* data sets through a combination of random sampling and replacement, we can create more training data sets for our combined classifier to use.

1. Randomly sample the original data set N times with replacement.
2. The new data set has the same size, where any instance is absent with probability $(1 - \frac{1}{N})^N$.
3. Construct k random data sets for k base classifiers and evaluate performance.

- Original training dataset:

1	2	3	4	5	6	7	8	9	10

- Bootstrap samples:

7	2	6	7	5	4	8	8	1	10

1	3	8	10	3	5	8	10	1	9

2	9	4	2	7	9	3	10	1	10

⋮

11.3.1 Analysis of Bagging

1. Simple method based on sampling and voting.
2. Possible to use parallel computing for each individual base classifier.
3. Highly effective over noisy data sets (outliers **may** vanish).
4. Performance is generally *significantly* better than the base classifiers but may be substantially worse.

11.4 Random Forests

11.4.1 Hyperparameters

1. Number of trees B.
2. Feature sub-sample size.

11.4.2 Interpretation

1. Logic behind predictions on individual instances can be tediously followed through the various trees.

A Random Tree is a variant of a Decision Tree, but

1. At each node, only *some* of the possible attributes are considered (i.e. a fixed proportion τ of all the attributes are used).
2. Attempts to control the number of unhelpful attributes in the feature set.
3. Much faster to build than a *deterministic* DT **but** increases model variance.

The structure of a Random Forest is an ensemble of Random Trees which is built using a different *bagged* training data set and classified via *voting*.

The idea behind Random Forests is to minimise the overall model variance **without** introducing combined model bias.

11.4.3 Analysis of Random Forests

1. Robust to over-fitting at the sacrifice of interpretability.
2. Generally performs well and super efficient.

11.5 Boosting

Intuition: Tune base classifiers to focus on the instances which are *hard to classify*.

1. Iteratively change the distribution and weights of training instances to reflect the performance of the classifier on the previous iteration.

2. Start with each training instance having a probability of $\frac{1}{N}$ to be included in the sample.

3. Over T iterations, train the classifier and update the weight of each instance according to whether it is correctly classified.

4. Combine the base classifiers via weighted voting

- Original training dataset:

1	2	3	4	5	6	7	8	9	10

- Boosting samples:

Iteration 1:	7	2	6	7	5	4	8	8	1	10
Iteration 2:	1	3	8	4	3	5	4	10	1	4
Iteration 3:	4	9	4	2	4	4	3	10	1	4

⋮

11.5.1 AdaBoost

An improvement to the default Boosting. Essentially re-initialises the instance weights if the classifier error rate ϵ is over 0.5

11.5.2 Analysis of Boosting

1. Mathematically complicated but computationally cheap.

2. Based on iterative sampling and weighted voting, but still computationally more expensive than bagging.

3. Guaranteed performance in the form of error bounds over the training data.

4. Has the tendency to overfit (saturating instances).

12 Semi-Supervised Learning

12.1 "Soft" k-means Clustering

It is possible to have a probabilistic version of k-means, where each instance is probabilistically assigned to each of the k clusters.

1. Set $t = 0$ and randomly initialise the centroids $\mu_1^0, \mu_2^0, \ldots, \mu_k^0$
2. Soft assign each instance x_j to a cluster based on:

$$z_{ij} = \frac{\exp(-\beta||x_j - \mu_i^t||)}{\sum_l \exp(-\beta||x_j - \mu_l^t||)}, \tag{32}$$

   where $\beta > 0$ and is called the **stiffness parameter**.

3. Update each of the centroids:

$$\mu_i^{t+1} = \frac{\sum_j z_{ij} x_j}{\sum_j z_{ij}} \tag{33}$$

   Think! It's the probability of this centroid occuring, given that the instances belong to this centroid.

4. Set $t+ = 1$ and repeat until centroids stabilise.

This is a combination of Overlapping, Probabilistic, Partitioning and Batch Clustering methods.

12.2 Clustering via Finite Mixtures

A *finite mixture* is a mixed distribution with k component distributions.

12.2.1 The Expectation Maximisation (EM) Algorithm

An algorithm with guaranteed *positive* hill-climbing characteristic and primarily used to estimate hidden parameter values or to cluster memberships.

Essentially a generalised "soft" k-means:

1. Based on the current estimate of the parameters, calculate the expected log-likelihood (**Expectation Step**)
2. Compute the new parameter distribution that maximises the log-likelihood (**Maximisation Step**)

Examples of usage: Estimate the values of missing values based on features with known values.

12.2.2 Convergence

The log-likelihood gives us an estimate of how "good" the cluster model is.

Convergence can be found by finding the differences between log-likelihoods between iterations. Once it falls below a predefined level ϵ, we can say that the cluster model has converged.

12.2.3 Analysis of the EM Algorithm

Advantages:

1. Guaranteed to have a positive hill climbing behaviour
2. Converges fast
3. Results in a probabilistic cluster assignment

Disadvantages:

1. Possibility of getting stuck in a local maximum (if it is not convex)
2. Relies on an arbitrary k which needs to be estimated
3. Tends to over-fit the data

12.3 Clustering Evaluation

Clustering can be evaluated with two metrics:

- **Unsupervised**: how cohesive are individual clusters/how separate is one cluster from other clusters?
- **Supervised**: how well do cluster labels match externally supplied class labels?

12.3.1 Unsupervised Evaluation

A good cluster "analysis" will have one or both of:

- High cluster cohesion (similar instances are all grouped closely)
- High cluster separation (instances from different clusters are distinctly separated)

An alternative is to use SS_{Res} of proximity using Euclidean Distance as its metric.

12.3.2 Supervised Evaluation

Cluster "validity" measures the degree of predicted labels to the true labels.

12.4 Semi-Supervised Learning

Learn from both unlabelled and labelled data. This is useful for clustering if we have some domain knowledge indicating inter-cluster compatibility.

12.5 Active Learning

Active learning builds off the hypothesis that a classifier can achieve higher accuracy with fewer training instances if it is allowed to have some say in the selection of the training instances.

The underlying assumption is that labelling is a finite resource, which should be expended in a way which optimises machine learning effectiveness.

Active learners pose *queries* (unlabelled instances) for labelling by an *oracle* (a human annotator).

12.5.1 Multiple Classifiers

Use a **query-by-committee** (QBC), where a suite of classifiers is trained. Then, the instance with the highest disagreement is selected for querying.

13 Evaluation

13.1 Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large training data set will also approximate the target function well over held-out test examples.

1. **Overfitting**: has the classifier tuned itself to the idiosyncrasies of the training data, rather than learning its generalisable properties?
2. **Consistency**: is the classifier able to flawlessly predict the class of all training instances?
3. **Generalisation**: how well does the classifier generalise from the specifics of the training examples to predict the target function?

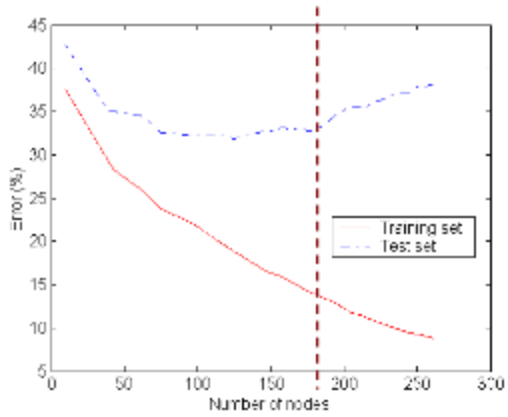
13.2 Overfitting

More training instances usually result in a better model, likewise more evaluation instances result in a more reliable estimate of effectiveness.

A good model should fit the training data well, and generalise well to unseen data.

The expectation is that training and test data are randomly selected from the same population, but neither are the entire population.

Possible evidence of overfitting:



Reasons:

1. Lack of coverage of population sample could lead to a poor model.
2. May be due to small number of examples, or due to non-randomness in training sample (known as *sampling bias*).

13.3 Terminology for Bias

1. **Model Bias**: the tendency of our classifier to make systematically wrong predictions.
2. **Evaluation Bias**: the tendency of our evaluation strategy to over / under estimate the effectiveness of our classifier.
3. **Sampling Bias**: If our training or evaluation data set isn't representative of the population (breaking the Inductive Learning Hypothesis)

We don't want high bias and low variance. (always makes the same prediction mistake regardless of the data set)

Having a low bias but high variance is acceptable in some circumstances.

14 Deep Learning

14.1 Neural Networks

Neural networks are composed of single neurons.

14.1.1 Neurons

Neurons are defined as

- an vector of numeric inputs
- a scalar output
- An activation function f (the hyperparameter)

Training a neural network entails identifying the weights $\mathbf{w}$ which try and minimise the errors on the training data.

A classic method of training for neural networks is **perceptron algorithm**, where each iteration is termed an **epoch**

14.2 Perceptron

The Perceptron algorithm guarantees convergence for *linearly separable* data, but

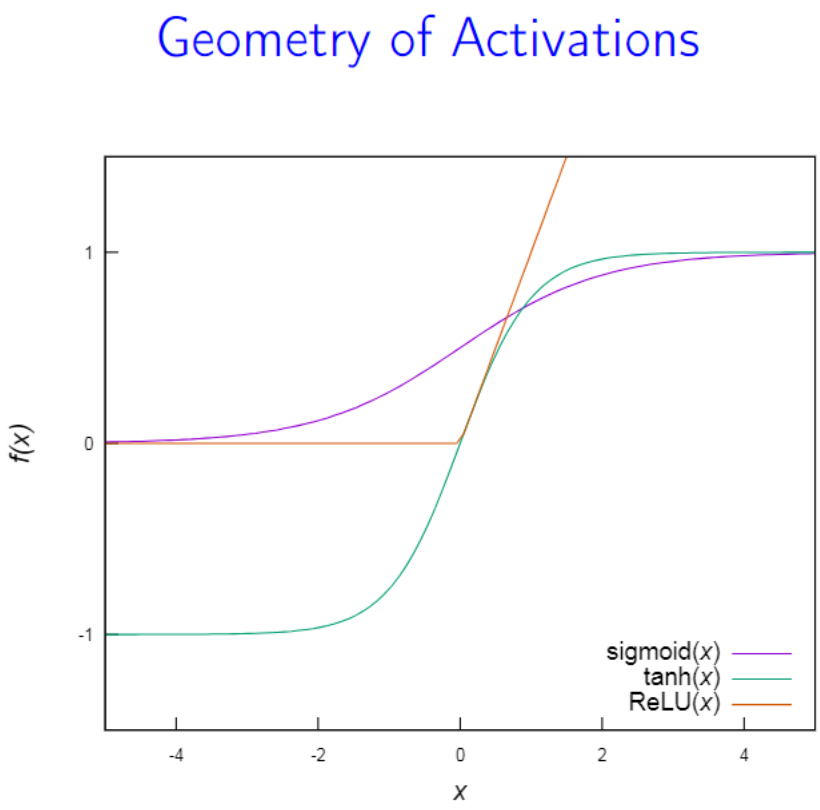
1. the convergence point depends on the initialisation
2. the convergence point will also depend on the learning rate

For non linearly separable data, there is no guarantee of convergence. It is however, able to extend to multi-class problems in a similar manner to logistic regression.

14.2.1 Common Activation Functions

There are a few non-linear activation functions that are used commonly:

1. logistic sigmoid function
2. hyperbolic tan
3. rectified linear unit



Actually, a neural network with a single neuron with a sigmoid activation is ***equivalent*** *to logistic regression.*

14.3 Stacking Neurons

The power of neural nets come from "stacking" several neurons in different ways such as layering (parallel neurons of varying size) and then creating extra layers of varying sizes.

Consider a **fully-connected feed-forward neural network**, this takes the following form

- **input layer** is made up of the individual features
- **hidden layer** is made up of of an arbitrary number of neurons, each of which is connected to all neurons in the preceding layer, and all neurons in the following layer
- **output layer** which combines the inputs from the preceding layer into the output

We denote this as a **multi-layer perceptron** ("MLP"), a fully connected feed-forward neural network *with at least* one hidden layer.

14.4 Representational Power of Neural Nets

The **universal approximation theorem** states that a feed-forward neural network with a single hidden layer is able to approximate any continuous function. In other words, it is possible for a **feed-forward neural net** with a **non-linear** activation function to *learn* any continuous basis function *dynamically*, unlike SVM's where their kernel function is a *hyperparameter*.

14.5 Regularisation

Neural nets are prone to chronic overfitting due to the large number of parameters, which means regularisation is critical.

14.6 Advanced Neural Networks (deep learning)

Deep learning is the combination of multiple hidden layers with sufficient data to train the models (a very large data set - think millions of instances).

14.6.1 Documentation Representation

- A "bag of words" representation of a document is a sparse vector of "term frequencies"
- "token n-grams" maintain more ifnromation from the document, but are much more sparse
- A token/n-gram can be *suggestive* of class, but no more than suggestive.

14.6.2 Image Representation

- An individual pixel can only be suggestive of some image property
- Only reasonable in combination (sequences of pixels can define a shape or feature)
- Feature engineering for image representation is difficult

14.7 Representation Learning

In deep learning, each layer of model extracts out a dense real-valued vector representation of a test instances (term: **embedding**).

This is generally the approach to deep representation learning.

14.7.1 Embedding Advantages

1. We have fewer features which means a faster train / test
2. Feature engineering is free (done on the fly)
3. With careful choice of representation, we can represent an instance by combinations of embedding(s), allowing us to use the trained network for problems it was never designed for

14.7.2 word2vec

One common example of **representation learning** in language analysis is **word2vec**.

The basic approach is that for each word in a corpus, we attempt to predict it from its context words (ignoring sequence).

Since it is framed as a classification task, we can synthetically generate negative instances through negative sampling, by randomly selecting words from the vocabulary

The model used to learn these word embeddings are a simple feed-forward neural network (with no hidden layers) using logistic regression as its **objective** function.

14.8 Convolutional Neural Networks

Convnets are crucial to computer vision applications. They are made up of the following layers:

- convolutional layers
- max pooling layers
- fully-connected layers

and two components:

- a kernel in the form of a matrix, which is overlaid on different sub-regions of the image and combined through an **element-wise product**
- a **stride** which defines how many positions in the image to advance the kernel on each iteration.

- For example, assuming the following 3×4 image and $s = 1$:

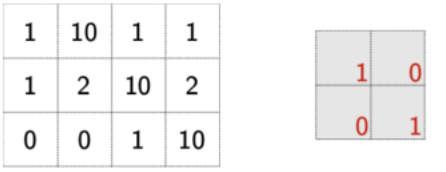
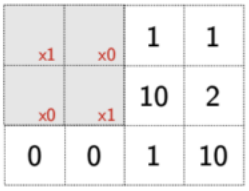


image 2x2 kernel

the first convolution would be:



convolution

$1 \times 10 + 10 \times 0 + 1 \times 0 + 2 \times 1 = 3$

- The full convolutional output would be:

$$\begin{bmatrix} 3 & 20 & 3 \\ 1 & 3 & 20 \end{bmatrix}$$

14.8.1 Convnet: Pooling

Pooling is comprimised of three components:

- a kernel in the form of a matrix, which is overlaid on different sub-regions of the image to determine the extent of the "pool"
- a **stride** which defines how many positions in the image to advance the kernel on each iteration
- the pooling basis using either **max** (1-max or k-max) or **average**

14.9 Deep Learning - Advantages

1. Massive improvements in empirical accuracy over standard data sets for vision and speech recognition tasks
2. Possible to model very large contexts/neighbourhoods compared to conventional models
3. Easy to combine different input modalities
4. Easy to use using TensorFlow, PyTorch, Keras

14.10 Deep Learning - Disadvantages

1. Any savings relating to feature engineering are **outweighed** by costs in terms of **architecture** engineering
2. Very expensive to train over large data sets (think about the implications this has on hyperparameter tuning)
3. Probably full of overblown claims about the capabilities of deep learning

15 Hidden Markov Chains

Markov chains assume that the likelihood of transitioning into a given state depends only on the current state, and not the previous state(s)

HMM's take the form $\mu(A, B, \Pi)$:

- A is the transition probability matrix.
- B is the output probability matrix (omission / observed)
- Π is the initial state distribution.

15.0.1 Forward Algorithm

Starting from the initial state Π, and calculates the next state.

15.0.2 Viterbi Algorithm

Uses dynamic programming to find the most likely sequence of states.

15.0.3 Applications of HMM

Text classification, Automated Speech Recognition or Optical Character Recognition.