**INTRODUCTION**

**What is learning? –** Fitting a function to data, allowing mapping of inputs to outputs. Lets us generalize to unseen inputs. Don’t need to learn if input set if finite, can just memorise input->output. Also can’t learn if there is no rule connecting input and output. Can’t use memorization for real world problems because real world problems usually have infinite inputs or we want to make predictions.

**Machine Learning Tasks –** Classification (predict **discrete** class labels based on features), Regression (predict **continuous** outcomes based on features), Predict relationships between features / outcomes (sequence learning, association rule mining). Understand and reconstruct the processes that produced the features / outcomes (model fitting, probability estimation).

**Machine learning methods –** Naïve Bayes classifiers, Decision trees, Support vector machines, Linear Regression, Logistic Regression, Gaussian mixture models, Hidden Markov models, Perceptron, Deep neural networks.

**Machine learning competence –** Training models, evaluating models, interpreting model performance, choosing the right model for a task (understanding pros and cons of each model).

**MACHINE LEARNING BASICS**

**Terminology –** Input consists of **instances** (a.k.a **exemplars or observations**) Individual, independent samples of the world.

Instances are composed of **attributes / features**, measured aspects of each instance, and **concepts (usually labels)**: things we wish to learn about the instances.

To generalize, we must learn a function that maps **attributes** to **concepts**, concept = f(attributes). We should be able to return the concept for any set of attributes, even attribute values the model hasn’t seen before.

**Concepts –** Discrete class labels (classification), numeric output (regression), clusters (when we don’t know the labels), probability of an event, the most likely order of events, a sequence of commands, a complex model, etc…

**Supervised –** methods receive labelled instances during training to assist in learning the association between attributes and concepts.

**Unsupervised –** methods receive unlabeled data, must learn from attributes only. Try to discover structure (correlated features, groups, sequences, etc.), discover **latent variables** that explain patterns in the observed instances (not observable but can be inferred from other observable variables), **reduce dimensionality** for a supervised learner.

**Supervised train & test:** Model learns mapping from attributes to concepts by seeing many examples, model is then tested on unseen / new set of attributes and tries to predict concept. Compare ground truth labels to predictions for evaluation.

**Unsupervised train & test:** Model learns a function that produces a useful concept from attributes (e.g. probability distribution, clusters). Can manually evaluate via human judgement or can test to see if future samples from the same distribution are well predicted by the model. More difficult to evaluate than supervised models.

**Association Learning:** Detect useful patterns, associations, correlations, or causal relations between attributes or between attributes and concepts. A good pattern is a combination of attribute values which are strongly correlated with other attribute values. Any kind of structure could be interesting and there is no “right” answer. Evaluation is difficult as there could be potentially many possible association rules in one dataset.

**Marr’s levels of analysis – Computational:** What structure does this machine learning model expect to see in the world? What rule / pattern / model etc. explains this data? **Algorithmic:** Given a model, what’s the best fit for this data? Usually involves minimizing and error or loss function. **Implementational:** How to find that best fit in finite time? Not always possible to solve exactly.

**Even when models have the same goal (find clusters) they make very different assumptions which leads to different results, fewer assumptions does not necessarily result in a better model, simplifying assumptions may allow a model to find a better result.**

**PROBABILITY**

**Variable Types:** Attributes can be various types – **Nominal** (categorical, discrete categories with no natural ordering), **ordinal** (discrete values with natural order, not real numeric values, mathematical operations may not make sense, distances may be inconsistent, can’t add or subtract, thresholds are still meaningful), **numerical** (continuous, real-valued with a defined zero point and no explicit bound, intervals are consistent and mathematical operations make sense). **Important as some models can only work with nominal or continuous data.**

P(X|Y) = P(X,Y) / P(Y),

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

P(Y|X) = (P(X|Y) P(Y)) / P(X) Bayes’ Rule

Independence -> P(X|Y) = P(X), P(X,Y) = P(X)P(Y)

Bernoulli trials – independent events with only two possible outcomes, Binomial distribution is a series of Bernoulli trials.

Multinomial distribution – results from a series of independent trials with more than two outcomes.

Empirical density functions – learn the probability distribution from the data instead of assuming a specific theoretical distribution. Kernel density estimation (KDE) estimates a continuous pdf from a set of observed data points. KDE represents the distribution as a sum of normal distributions with means at each of the observed data points.

**Probability concepts are key for machine learning, knowing the likelihood that outcomes (features, etc.) will occur together lets you predict one from the other. Modelling probability distributions (or density functions) is central to many models (Naïve Bayes, generative models, unsupervised models).**

**NAÏVE BAYES**

People can grasp a “class” concept from very limited data: few examples, noisy data, diverse data set, ambiguity about feature importance. To get this generalization performance from machines, we can use probabilistic models.

**Probabilistic models:** provide a framework for modelling systems that are noisy/uncertain. Rules that let you generalize from limited observations. Based on laws of probability, so gives an optimal prediction given the available data and assumptions built into the model. Goal is classification so we build a supervised model.

Given an instance T, which class c is the most likely: c^ = arg max C P(c|T). (i.e. the class that maximises the probability given instance T). For each class c: find examples of c in the training data, count the number of times T has been observed, then choose class c^ with the greatest frequency of observed T. **Requires a massive amount of data. Need to have seen every possible combination of attributes in the training set, ideally at least a few times per class, to have a good estimate of frequency.** Naïve Bayes solves this problem by assuming different attributes are statistically independent within the same class (conditional on class). Compute P(c|T) from P(T|c).

P(C|X) = P(X|C) P(C) / P(X) where P(X) = P(X|a) P(a) + P(X|b) P(b)…

Simply find the class that maximises P(C|T) = P(T|C) P(C) (exclude denominator of Bayes’ Rule as all c in C are divided by the same thing so don’t need to include it for comparison.

= arg max Cj in C P(x1,x2,…,xn|Cj) P(Cj) = argmax cj in C P(cj) product (P(xi|cj)). Intuition is each class generates, attributes with some likelihood, which class is most likely to have generated these attributes. The prior P(cj) can be estimated from the frequency of classes in the training set (maximum likelihood estimate). Naïve Bayes learns the priors from the training set and uses them in prediction. **Good if the training set correctly reflects the real-world / test set distribution of the classes (or is close), but potentially a problem if you want to apply the classifier to a new situation with new priors.**

Problem, if any P(xi|cj) = 0, the final value will be zero. Solution, treat unobserved events as possible but unlikely (all probabilities must be > 0) which isn’t likely wrong if we didn’t observe a particular attribute value within a class.

**Probabilistic Smoothing I –** Replace 0 with small positive constant e, should be much less than (1/N). Since e is tiny, can assume all other probabilities still sum to 1 and don’t need to scale other values. Tends to reduce to comparisons to whichever class has the fewest e.

**Probabilistic Smoothing II –** Increase all counts by 1 (**Laplace smoothing**) unseen events get a count of 1, increase all other counts by 1. Can generalise to increase all counts by alpha. Probabilities are changed drastically when there are few instances, but changers are smaller with more instances (mimics confidence). Add-one smoothing is known to overestimate the likelihood of rare events, but e or lower values of alpha can underestimate. Hard to choose right alpha in practice.

**Probabilistic Smoothing III –** Good-Turing estimation uses the observed counts of different events to estimate how likely you are to see a never-before-seen event.

**Missing Values –** Missing values can be ignored during training and testing.

**PROS AND CONS:**

**We don’t need a perfect estimate of P(c|T) for every class, we just need to know which class is most likely, ignoring correlation between attributes typically doesn’t change the rank of class probabilities. This extends to Naïve Bayes being robust to small errors in estimating P(xi|cj) – again class rank typically doesn’t change.**

**Simple to build, fast. Computations scale well to high-dimensional datasets. Explainable – generally easy to understand why the model makes the decision it does.**

**Inaccurate when there are many missing P(xi|cj) values. Conditional independence assumption becomes problematic for complex systems.**

Multiplying a lot of numbers (0, 1] can lead to underflow. Can take the log transformation of each probability and sum instead of multiplying.

**DATA TYPE CONVERSION**

Naïve Bayes assumes nominal data, what happens if we have continuous data? Can convert data types. In general,if we have attributes of the wrong type for a given model, we can discard those attributes, convert the attributes, or change the model.

**Nominal -> numeric:** Convert category names to numbers, creates artificial ordering when no order exists, makes some categories seem more/less similar to each other.

One-hot encoding, if an attribute has m possible values, encode it with m bits with a single bit indicating which value it takes. Best way to represent nominal values in a continuous space but highly increases dimensionality of the data.

**A picture containing text, screenshot, font, line

Description automatically generatedComputing distances:**

**Numeric -> nominal: Discretisation** is the translation of continuous numeric attributes to discrete nominal attributes. Generally a two-step process, decide how many nominal values (=intervals) onto which you will map the numeric values and decide where to place the boundaries for these intervals. **Equal-width discretization** find min/max of range, partition into n bins of width (max-min)/n. **Equal-frequency discretization** sort values, find breakpoints that produce n bins with (approximately) equal numbers of items. **Arbitrary group boundaries, equal-width is sensitive to outliers, equal frequency is sensitive to sample bias, user must still choose n.**

Can also use some clustering method to discover natural breakpoints within your data (e.g. k-means clustering). **More complicated, if data doesn’t have natural “groups”, k-means result is same as equal-width discretization, sensitive to outliers, user must choose n.**

**Supervised discretization:** Group values into class-contiguous intervals (Sort values, and identify breakpoints in class membership. Reposition breakpoints if the numeric value is the same. Set the breakpoints midway between the neighbouring values. **May help you find “groups” that are most relevant for your classification task, but arbitrary group boundaries, arbitrary number of groups (tends to produce too many groups), tends to overfit training set.**

Overall various options with pros and cons, discretization can sometimes help simplify data to make it easier for the model to learn, but you can potentially lose details that would have been useful.

For Naïve Bayes, if xi is a numeric attribute, then P(xi|cj) can be computed using a probability density function. Can use Bernoulli/Binomial (binary attributes), Multivariate (nominal), Multinomial (natural numbers corresponding to frequency), Gaussian distributions. Gaussian is a reasonable approximation for many events and its easy to compute mean and standard deviation from data to use in the model, so Gaussian distributions are a good baseline distribution to use for numeric attributes. KDE is also a way to try and build a probability distribution for use in Naïve Bayes classification for numeric attributes.

**Naïve Bayes works with various data types, just need to compute P(xi|cj) differently. Therefore Naïve Bayes gives the ability to mix numeric and nominal attributes in one model.**

**MODEL EVALUATION**

**Accuracy –** Basic evaluation metric, measures percent of time classifier is correct. (number of correct labels / total number of test instances).

**Random holdout –** Randomly partition data into “training” and “test” sets. Train on training set, test on unseen “holdout” data / test data. Trade-off between having enough data for training and a representative test set, generally prefer having more training data though for better model performance.

**Repeated random subsampling:** Random holdout repeated multiple times, average over all iterations, slower but more accurate evaluation.

**Cross-validation:** preferred method, split data into m partitions, train on m-1 partitions, test on 1 partition, use each partition as a test set. Aggregate evaluation metric across m partitions, can average or concatenate. This guarantees each item appears as a test item exactly once. Leave-one-out is ideal but extremely slow unless N is tiny.

**Stratification:** Random sampling may produce different class proportions / distributions. Stratification or vertical sampling ensures training data and test data have the same class distributions as the whole dataset.

**Inductive Learning Hypothesis:** Any model which approximates the target function well over a large training set will also generalise to unseen examples. However, machine learners also suffer from **inductive bias** – assumptions made about the data to build the model and make predictions -> if assumptions don’t hold on unseen examples then predictions suffer, different assumptions lead to different predictions.

**Validation set:** sometimes a validation set is used as a “test” set for training data to optimise parameters and check if the model converged to a good solution. Using a validation set assists in the goal of avoiding overfitting and judging how well a model generalises.

A picture containing text, screenshot, font, number

Description automatically generated

Note that certain types of errors may be more important than others depending on the machine learning problem. (e.g. False Negatives are extremely important to minimise in cancer screening).

A picture containing text, font, screenshot, white

Description automatically generated**Precision:** How often is the model correct when it predicts a positive case. **Recall/Sensitivity:** What proportion of true positives did the model detect.

A picture containing font, text, number, line

Description automatically generatedHigh precision + low recall: model needs a lot of evidence to say “positive”, low precision + high recall: model doesn’t need a lot of evidence to say “positive”. Ideally we want both to be high, F-score combines the two metrics.

**Specificity** is the proportion of true negative cases the model was able to detect. (TN / TN + FP)

Confusion matrcies show ground truths vs what the model has classified and shows the pattern of errors in a multiclass classification task.

In multiclass evaluation you can evaluate **one-vs-rest** where you treat one class as positive and all others as negative. (doesn’t care if negative classes are misclassified).

Total accuracy is sum of diagonal cells over sum of entire table. Precision/recall/F-score are computed per class (OvR) and averaged across classes. Can also macro, micro and weighted average.

**Baseline:** Simple Naïve method that we would expect any machine learning method to beat (e.g. random guessing).

**Benchmark:** Established rival technique to which we are comparing our method (e.g. current best-performing algorithm on a leaderboard). Usage of terms isn’t so strict in practice, baseline can be used for both.

**Common baselines:** Random baseline (guess uniformly or guess based on class distribution in the training set). **0R:** always guess the most common class. **Regression:** always guess mean value, **Object Detection:** guess middle of image, **1R:** Assign each level of each attribute to the most likely class, use attribute that has lowest error rate (i.e. most correctly classifies) as the “one rule”.

**We cannot guarantee optimal performance on a test set without knowing what unseen data will look like (not possible). Therefore test data is our best way of evaluating generalisation.**

**DECISION TREES**

1R is a decision stump, we can construct decision trees of arbitrary depth to capture more information (e.g. complex feature interaction).

A screenshot of a computer program

Description automatically generated with low confidence**ID3:** Construct decision trees in recursive divide-and-conquer fashion.

Classify new instances by traversing down tree and classifying according to majority class at deepest reachable node. Complications include unobserved attribute-value pairs and missing values. Can use disjunctive descriptions to fully represent in a formula which conditions relate to a particular class (e.g. outlook = normal ^ humidity = normal V outlook = rain for class C).

A picture containing text, font, receipt, screenshot

Description automatically generated**Attribute selection: Entropy** (measure of unpredictability), given a probability distribution the information (in bits) required to predict an event is the distribution’s **entropy** or **information value.**

If most of the probability mass is assigned to a single event -> event is predictable -> entropy is low, and if probability mass is spread between many events then the reverse is true. In the context of Decision Trees, we are looking at the class distribution at a node. We want leaves with **low entropy** (skewed towards a particular class i.e. we gain useful information about a particular class or classes from this attribute).

**Information Gain:** the expected reduction in entropy caused by knowing the value of an attribute. Compare entropy before splitting with weighted average of the entropy over children nodes after splitting (=mean information). If entropy decreases, then we have better tree as it is more predictable.

A picture containing text, font, screenshot, white

Description automatically generated

**IG = H(root) – Mean Info(children)**

IG tends to prefer highly-branching attributes, subsets are more likely to be pure (above purity threshold) if there is a large number of attribute values. May result in overfitting.

**Gain Ratio:** reduces the bias for information gain towards highly-branching attributes

**Split Info:** the entropy of a given split (evenness of split) aka entropy of the attribute values, not with respect to target classes.

GR = IG / SI.

ID3 performs a simple-to-complex, hill-climbing search through a space of hypotheses for one that best fits the training examples. No backtracking. We want to get the smallest tree or hypothesis that fits the data (a short tree that fits the data is unlikely to be a coincidence whereas a large data might be and therefore will generalise poorly)

**ID3 Decision Trees are highly regarded among supervised learners, fast to train and classify, and has some techniques to account for missing/continuous feature values. HOWEVER they are susceptible to the effects of irrelevant features, and are prone to overfitting.**

**Random Trees:** use a sample of the possible attributes at a given node, helps to account for irrelevant attributes, basis for random forests.

**INSTANCE-BASED LEARNING**

Requires labelled examples stored in memory, learns directly by example. We need attributes to draw comparisons between instances. Location of instances in a vector space is dependent on which attributes are used.

**Similarity:** Numerical measure of how alike two data objects are, higher when objects are more alike, often [0,1]

A picture containing text, font, screenshot, number

Description automatically generated**Dissimilarity:** Numerical measure of how different two data objects are, lower when objects are more alike, minimum is often 0, upper limit varies.

**Distance Measures:** function that takes two points in a space as arguments, no negative distances, distance is zero from point to itself, distance is symmetric, d(x,y) <= d(x,z) + d(z,y)

A picture containing font, white, diagram, text

Description automatically generated**Manhattan Distance:**

A picture containing text, handwriting, font, line

Description automatically generated**Cosine Similarity:**

**Nearest Neighbour Classification:** Nearest Neighbour is the point with max similarity or min distance to the current point. Induces a decision boundary based on where the instances lie in the feature space.

**K Nearest Neighbours:** compare to K nearest neighbours, smaller K tends to lead to lower performance due to noise (overfitting), larger values of K drive classifier performance toward 0R performance (as you essentially emulate 0R guessing). Trial and error is generally only way to get suitable K. K is generally set to odd value to always have a majority class. Can use random tie breaking, prior probabilities of classes, adding 1 extra neighbour to resolve ties. Weighted KNN: classify test input according to the weighted accumulative class of the K nearest training instances, where weights are based on similarity of the test input to each of the K neighbours.

**Weighting Strategies: Equal weight**, classify according to **majority class** of K neighbours (regular KNN). **By the inverse linear distance** from test instance to instance j (normalised). **By the inverse distance** from the test instance to instance j.

Typical implementation involves the brute-force computation of distances between a test instance and every training instance. Competitive for small datasets, becomes infeasible as number of samples N grows, O(DN) complexity where D is dimensionality (number of features).

**Model built is the dataset itself, KNN is lazy. The time we save in training is lost if we have to make many predictions. Naïve Bayes and Decision Trees are generally much smaller than the dataset (O(CD) calculations, C classes D attributes for NB, O(D) node traversals for DT).**

**Simple, instance based, model free. Can produce flexible decision boundaries. Incremental (can add extra data on the fly). HOWEVER, requires a useful distance function. Arbitrary K value, Lazy learner: everything done at run time, Prone to noise and the curse of high dimensionality.**

**Nearest Prototype Classification:** Parametric variant of nearest neighbour classification. Calculate the centroid of each class and classify test instances according to closest centroid (by Euclidean distance). Centroid is calculated by averaging numeric values along each attribute axis.

**SUPPORT VECTOR MACHINE**

Find a straight line/hyperplane that separates two classes. Classes need to be linearly separable for this to be possible. Separating hyperplane in D dimensions can be defined by a **normal w** and an **intercept** b.

A picture containing text, font, screenshot, diagram

Description automatically generatedf(x) = w^T x + b

Straight line in 2D space, plane in 3D space

**Optimal Solution:** maximise the distance between the hyperplane and the difficult points close to decision boundary. Most stable under alterations to input.

**Margin:** 2 x the distance between closest data point and boundary

Associate one class as positive (+1), and one as negative (-1).

Find the best hyperplane **w** and *b* which maximises the margin between the positive and negative training instances.

A black text on a white background

Description automatically generated with low confidenceHow to learn **w** and *b*?

A picture containing text, font, screenshot, algebra

Description automatically generated

**Non-linearly Separable: Soft Margins,** allow some points to be incorrectly classified (i.e. be on wrong side of hyperplane) to form a better overall decision boundary.

**Non-linear SVM:** Make non-linearly separable problem separable by mapping data into better representation space. Apply mapping function to the feature vectors and perform linear classification on the new feature vectors.

**Multi-Class SVM:** Convert to two class problem, **one-versus-all:** one classifier to separate one class from the rest of classes, choose the class which classifies test data point with greatest margin (i.e. furthest away from decision boundary) (group other classes together on other side of decision boundary for one-vs-all). **One-versus-one:** one classifier per pair of classes, choose the class selected by most classifiers. **Training time can be a serious issue, as we need to build many SVMs.**

**SVM is a linear hyperplane-based classifier for a two-class classification problem. SVM selects the hyperplane with maximum margin. Soft margins allow some data points to violate the separating hyperplane. Non-linear SVM transforms data to a new feature space and finds a hyperplane separating two classes in the new space. Classification of new instances is efficient. SVMs can be applied to non-linearly-separable data with an appropriate kernel function.**

**Maybe revise maths behind SVM but don’t need to memorise, just understand whats happening and the terminology**

**INTERPRETATION AND VISUALISATION**

Interpret models in two ways: explain why a given model has misclassified an instance in the way it has (**error analysis**), and explain why a given model has classified an instance in the way it has (**model interpretability**).

**Error Analysis: identify** different classes of error that the model makes (predicted vs actual labels), **hypothesise** what has caused the different errors and test that, **quantify** whether data quantity/sparsity is to blame or something more fundamental, **feed** those hypotheses **back** into feature/model engineering to see if the model can be improved. Use confusion matrices as a starting point, use model assumptions to guide error analysis in terms of particular traits in the instance that are leading to misclassification.

**Model Interpretability:** Interpret the reasoning for a model classifying an instance the way it does, hyperparameters and parameters define a lot of the model and can be related to the reasoning. **Hyperparameters** are parameters which **define and constrain the learning process** and are **explicitly specified by the machine learning engineer before training the model.** Regular parameters are learned when a model with a given set of hyperparameters is trained, they are internal to the model and set by the model itself.

**KNN:** Hyperparameters: k, distance/similarity metric, feature weighting

Parameters: prototype for each class, size: O(|C||F|)

Interpretation: relative to the distribution of the prototypes in the space and distance to each prototype

**Naïve Bayes:** Hyperparameters: smoothing method, optionally choice of distribution used to model features

Parameters: class priors and conditional probability for each feature-value-class combination, size: O(|C|+|C||FV|)

Interpretation: based on the most positively-weighted features associated with a given instance

**Decision Trees:** Hyperparameters: attribute selection: e.g. information gain, gain ratio, etc. and Stopping Criterion.

Parameters: decision tree itself, typical size: O(|FV|) (feature-value pairs)

Interpretation: based on path through decision tree

**SVM:** Hyperparameters: penalty term C for soft-margin SVM, choice of kernel and any kernel hyperparameters, how to deal with multi-class problem.

Parameters: hyperplane: normal vector + bias, size: O(|C||F|)

Interpretation: 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.

**Tune Hyperparameters:** Understand the meaning of a hyperparameter, try different settings (manual tuning, grid search etc.), compare performance on validation set.

**Visualising Data:** Visualising data can be a valuable way of getting to know it (e.g. visually detect anomalies / outliers, check distribution of data, check decision boundary, etc.)

**Dimensionality Reduction:** If there are more than 3 attributes can reduce feature space down to 2 or 3 dimensions. Can use feature selection (removing features) or dimensionality reduction where features are compressed / converted / combined into lower dimensional features. PCA (Principal Component Analysis) is a popular form of dimensionality reduction. Central idea is that the principal components (new features) are linear combinations of the original features, are orthogonal to each other, capture the maximum amount of variation in the data.

**LINEAR REGRESSION**

A picture containing text, font, white, line

Description automatically generatedUsed for continuous output / concepts from continuous attributes. Assume a linear relationship between the attribute values ak and the concept c.

A picture containing text, font, handwriting, white

Description automatically generatedAt its most basic, the linear relationship can be expressed as a line.

A picture containing text, font, handwriting, diagram

Description automatically generated

Linear functions capture changes in one variable that correlate linearly with changes in another variable. This simple assumption permits simpler mathematical strategies. For some variables, this assumption makes sense. Find the optimal beta weights in training, and predict using these weights on a new set of features **x**.

A picture containing text, font, diagram, line

Description automatically generated**Least Squares Method:** find the line that minimises the sum of the squares of the vertical distances between predicted y^ and actual y. **Minimise** the Mean Squared Error (MSE) of N data points.

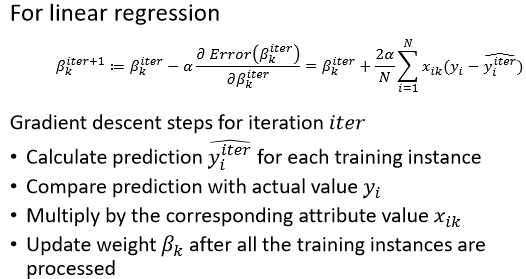
MSE is convex -> local minimum is a global minimum.

A picture containing font, text, white, diagram

Description automatically generated**Solution:** calculate partial derivatives of the loss function, with respect to **Beta**, for N instances and D attributes. Set all partial derivatives of each Beta to 0, and solve D+1 equations with D+1 unknowns.

**Parameter Estimation:** Given an evaluation metric M (like Accuracy or Error), a dataset T, a feature representation F(T), and a learner L with parameters p. Maximise or minimise M(p;L,F(T)), where learner L and dataset F(T) are fixed.

Learning => finding parameters that optimise some criterion M for model L and feature representation F(T).

**How to find?: Analytic Solution:** closed form, exact computation, requires solving a system of equations. (e.g. for linear regression, finding Beta). Challenges: derivatives can be undefined or not calculable. **Exhaustive Solution:** For M parameters taking N unique values, perform N^M train-evaluate cycles. Works for methods with a small number of hyperparameters, taking a small number of values, e.g. KNN (similarity / distance metric, voting strategy, K). **Grid Search:** Try different combinations of supplied parameters from a ‘parameter grid’ and see which combination produces best results. More samples: closer to optimal estimate, but more time required. **Iterative Approximation:** Guess the parameter values, compute error, decide whether to stop, compute parameters based on previous guess, continue until stop. **Gradient Descent:** Find the direction: a vector of partial derivatives which measure the slope (gradient) of the criterion / function. Each update reduces error slightly / increases performance slightly. Alpha is learning rate, which defines how big a step to update the parameters by, small alpha = slow algorithm, large alpha = might overstep the minimum.

**Evaluation:** Doesn’t make sense to evaluate numeric prediction tasks in same manner as classification tasks. There are many scoring metrics for regression tasks, all of which are based on the **absolute or relative difference** between the predicted value yi and the actual value yi of test instances.

A picture containing text, font, screenshot, diagram

Description automatically generatedCan use MSE, RMSE (Root Mean Squared Error), RRSE, Correlation Coefficient (Pearson’s correlation, statistical correlation between predicted and actual values).

Relative ranking of methods across different metrics is reasonably stable, such that actual choice of metric isn’t crucial. **Linear regression is an intuitive model, with relatively easy implementation. Can apply transformations to x variables when there are nonlinear relationships.**

**LOGISTIC REGRESSION**

**Regression -> Classification:** Discretisation: map continuous values onto discrete labels, e.g. predict age group 11-20, 21-30,… instead of the value of the age

**Classification -> Regression:** Binary-class, take class labels 0 and 1 as numeric values. Multi-class: build a set of regression tasks (OvOne or OvAll), approximates a numeric membership for each class.

**Probabilistic Classification:** Attempt to model P(c|**x**) directly, assuming a 2-class problem (Y/N). P(c=Y|**x**): probability of an instance with class Y, if class label = Y: P(c=Y|**x**) = 1, if class label = N: P(c=Y|**x**) = 0. If the numerical attributes of the instance are predictors, P(c=Y|**x**) is the target variable: P(c=Y|**x**) = **Beta \* X** = Beta0 + Beta1x1 + … + BetaDxD. Problem: Linear function, no guarantee that P is in [0,1], meaning of probabilities outside [0,1] is unclear.

**Logistic Function:** P is in [0,1] for all z values (z= **Beta \* x**). Curve is unbalanced, fine granularity of responses as P->0, coarse responses as P->1, P>1 when **Beta\* X** > 0.

Logit (log odds) results in same response behaviour for high and low P. Can use linear regression to model logit P. Sigmoid function (S shape).

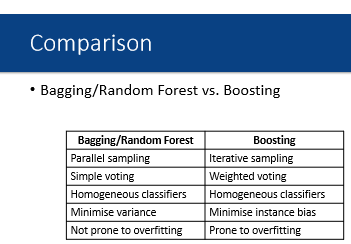
Again we try to find optimal Beta as in linear regression. P(c=Y|**x**) > 0.5-> predict **class = Y**, P(c=Y|**x**) < 0.5 -> **predict class = N**.

A picture containing text, font, line, handwriting

Description automatically generatedMaximise log-likelihood or likelihood function, preferred method is gradient ascent / iteration.

A picture containing text, screenshot, font, number

Description automatically generatedMultinomial logistic regression, take one class as pivot, for every other class cj, build a regression model, treat class cj as class Y, pivot class as class N (OvAll). End up with (|C|-1) different logistic regression models, predict the class label according to the one with the highest probability score.

**Forms a linear decision boundary, provides probabilities of the classification result, easy to interpret using the learnt parameters, good baseline for classification tasks before trying more complex models.**

**CLASSIFIER COMBINATION**

**Classifier combination / ensemble learning:** constructs a set of **base classifiers** from training data and performs classification by **aggregating** the outputs made by each base classifier.

**Intuitions:** Take into account the opinions of several experts rather than relying only on one. The combination of lots of weak classifiers can be at least as good as one strong classifier. The combination of a selection of strong classifiers is (usually) at least as good as the best of the base classifiers.

Combination works when **the base classifiers do not make the same mistakes** and **each base classifier is reasonably accurate.**

**Construct Base Classifiers: Instance Manipulation:** generate multiple training datasets through sampling, and train a base classifier over each (e.g. bagging). **Feature manipulation:** generate multiple training datasets through different feature subsets, and train a base classifier over each (e.g. random forest). **Algorithm manipulation:** semi-randomly tweak internal parameters within a given algorithm to generate multiple base classifiers over a given dataset.

**Voting:** For nominal classes, run multiple base classifiers over the test data and select the most predicted class. For continuous output, average over the numeric predictions of the base classifiers.

**Bias:** The tendency of a classifier to make systematically wrong predictions

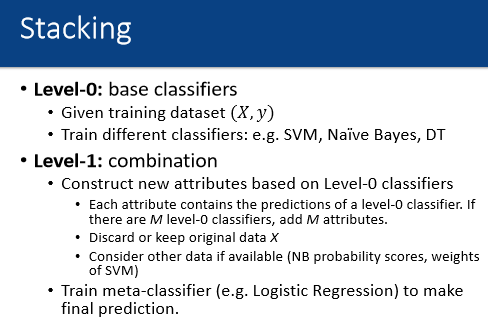
**Variance:** The tendency of producing different parameters or predictions for different training sets using the same learner. Lower bias and lower variance -> better generalization.

**Bagging:** Bootstrap aggregating: construct new datasets through a combination of **random sampling and replacement** [more data = lower variance (better performance)]. Construct k random datasets for k base classifiers and combine via voting. **Simple method based on sampling and voting, possibility to parallelize computation of individual base classifiers, effective over noisy datasets, as the outliers may vanish, performance is generally significantly better than the base classifiers and only occasionally substantially worse.**

**Random Tree:** A decision tree, but only some of the possible attributes are considered at each node (e.g. a fixed proportion t of all the attributes, faster to build than a deterministic Decision Tree, but increases model variance).

**Random Forests:** an ensemble of Random Trees, each tree is built using a different bagged training dataset, combined classification via voting. **Hyperparameters:** number of trees B, which can be optimized based on “out-of-bag” error, feature sub-sample size: as it increases, both the strength and the correlation increase. **Interpretation:** logic behind predictions on individual instances can be followed through the various trees (like in the case of decision trees). **Generally a very strong performer, efficient to construct. Parallelizable. Robust to overfitting. Interpretability sacrificed.**

**Boosting: Intuition:** tune base classifiers to focus on the hard-to-classify instances. **Method:** iteratively change the distribution and weights of training instances to reflect the performance of the classifier on the previous iteration. Start with sampling: each training instance having 1/N probability of being included in the sample. Over T iterations, train a classifier and update the weight of each instance according to whether it is correctly classified, combine the base classifiers via weighted voted. (i.e. put more of the weighted instances in the samples to train the classifiers on instances that they are getting wrong). **Base classifiers: decision stumps (1R) or decision trees. Mathematically complicated by computationally cheap method based on iterative sampling and weighted voting. The method has guaranteed performance in the form of error bounds over the training data. More computationally expensive than bagging. In practical applications, boosting has the tendency to overfit.**

**Stacking: Intuition:** smooth errors over **a range of algorithms** with different biases. Method 1: voting (which classifier to trust?). Method 2: train a meta-classifier (level-1 model) over the outputs of the base classifiers (level-0 models). Learn which classifiers are the reliable ones and combine the output of base classifiers, train using nested cross validation to reduce bias.

**Able to combine different heterogeneous classifiers with varying performance. Mathematically simple but computationally expensive method. Generally, stacking results in as good or better results than the best of the base classifiers.**

**FEATURE SELECTION**

Main goal is to identify good features and remove the redundant and irrelevant features to improve the performance. Also, seeing important features can suggest other important features. Fewer features = smaller model = faster answer (doesn’t necessarily = more accurate answer, which is more important than faster answers).

**Wrappers:** choose subset of attributes that give best performance on the validation data. **Can find the feature set with optimal performance on validation data for this learner, but takes a long time (not practical for large datasets).** Can use **sequential forward selection (Greedy Approach):** Train and evaluate model on each single attribute, choose the best attribute and add to the model, repeat until performance (e.g. accuracy) stops increasing. **Can converge to a sub-optimal (or even bad) solution and assumes independence of attributes).** Can also use **sequential backward selection (Ablation Approach)**: Train and evaluate model with all attributes, remove each attribute, train and evaluate model, choose attribute which causes least performance degradation and remove, repeat until performance (e.g. accuracy) starts to degrade by more than a threshold. **Removes most of the irrelevant attributes at the start, performs best when the optimal subset is large. But, running time is very slow with a lot of attributes, not feasible on large datasets.**

**Embedded Methods:** Models perform features selection as part of the algorithm, e.g. Decision Trees, Regression Model with regularization, e.g. linear regression with L1-norm regulariser (LASSO). However, the models can still benefit from other feature selection approaches.

**Filtering Methods:** Most popular strategy, evaluate “goodness” of each attribute, consider each attribute separately (linear time in number of attributes). Well correlated with interesting class, inversely correlated with interesting class, well correlated or inversely correlated with uninteresting class all make an attribute “good”.

**Pointwise Mutual Information:** Attributes with greatest PMI are most correlated with class.

**Mutual Information:** Consider the PMIs of all the combinations of attribute set a, not a, interesting class c, not c. Contingency Table is a compact representation of these frequency counts. Can compute P(a,c) based on the table, e.g. P(a,c) = sigma(a,c) / M (number of features and classes combined).

**Chi-Square:** Similar idea with MI, but different solution. Conduct statistical test to check the independence of an attribute and the class. Can also represent in a contingency table. Compare the value actually observed O(W) with the expected value E(W). Fit Chi-Square statistic to a chi-square distribution, **higher value of Chi-Square indicates the dependency between a feature and the class.**

**Common Issues: Nominal attributes of multiple values:** Treat as multiple binary attributes (difficult to interpret regarding the original feature). Expand formulae and contingency tables.

**Continuous Attributes:** Estimate probabilities P(a,c), P(a), P(c) etc. by fitting a distribution such as Gaussian and using pdf. Could also discretise values.

**Multi-class Classification:** More difficult than binary classification task. Certain features may be useful for predicting one class but not others, so have to be careful which ones are removed.

**Feature selection is necessary for distance-based models, e.g. KNN, and to a lesser extent NB and DTs. SVMs can work well without feature selection.**

**MODEL EVALUATION II**

**Evaluation of Supervised ML:** Start with a dataset of instances comprised of attributes and labels, build a classifier using a learner and the dataset, assess the effectiveness of the classifier. Comparing the predictions with the actual labels on unseen instances using metrics (accuracy, precision, recall, F1-score).

**Tensions in Classification: Generalisation, Consistency, Overfitting. Trying to balance consistency and generalization against overfitting.**

**Overfitting:** Learning Curves are a plot of learning performance over experience or time. **Y-axis:** performance metric, **X-axis:** conditions, e.g. sizes of training sets, model complexity, iterations…, **Training learning curve:** calculated from the training set, shows how well model is learning, **Test learning curve:** calculated from holdout/test set, shows how well model is generalizing. For different sizes of training set can evaluate holdout amount. For comparing models, could for example look at MSE of different polynomial orders used in linear regression. **Could reveal overfitting if the test performance is far worse than the training performance, which likely means that the model is learning overly specific patterns in the training data rather than the true relationship between attributes and class labels.**

**Reasons for overfitting:** Decision boundary could be distorted by noise (and overly fit around this noise instead of using a soft margin), a simpler decision boundary could generalize better. Limited training set: not fully represent patterns in the population, could be due to small number of samples or non-randomness in training samples (sample bias->not true representation of population).

**Solutions for overfitting:** In soft-margin SVM, the slack variables provide regularization. In linear regression, the norm of the parameters can be used as a regulariser. Mathematically, a norm is a total length of a vector, so norm of Beta. Linear regression with L2-norm regularization (ridge regression) adding the sum of squared beta values to the error term, encourages solutions where most parameter values are small.

A black text on a white background

Description automatically generated with low confidence

Linear regression with L1-norm regularization (LASSO), adding the sum of Beta’s absolute values to the error term, encourages solutions were few parameters are non zero.

**Bias: Model bias:** the tendency of a model to make systematically wrong predictions

**Evaluation bias:** the tendency of an evaluation strategy to over- or under-estimate the effectiveness of a model

**Sampling bias:** If the training or test dataset isn’t representative of the population, which breaks the **Inductive Learning Hypothesis.**

Variance refers to model variance and evaluation variance.

**Model bias** in **regression** context: For every evaluation instance, the signed error can be calculated. Assuming every instance is independent, bias is the average of these signed errors. sqrt(MSE)

A model is **biased** if: the predictions are systematically higher or lower than the true value

A model is **unbiased** if: the predictions are systematically correct or roughly equally vary around the true values.

**Model bias** in **classification** context: Label predictions can’t be “too high” or “too low”, “biased towards the majority class” means our model predicts too many instances as the majority class.

Typically compare the class distribution: **unbiased** if same distribution as true distribution, **biased** if model produces labels with different distribution from actual distribution.

**Model variance** relates to the tendency of different training sets to produce different models or predictions with the same type of learner: **high variance** if a different randomly sampled training set leads to very different predictions on the evaluation set and **low variance** if a different randomly sampled training set leads to similar predictions (regardless of correctness).

Lower bias and lower variance -> better generalisation.

**Evaluation:** The evaluation metric is also an estimator, desire to know the true error rate of a classifier, but only have an estimate of the error rate, subject to some particular set of test instances. The quality of estimation is independent of the trained model. For example, training error is one starting point if estimating the performance of a classifier on new cases. With unlimited samples, apparent error rate will approach the true error rate.

**Evaluation bias:** Our estimate of model effectiveness is systematically too high / low

**Evaluation variance:** Our estimate of the effectiveness of a model changes a lot as we alter the instances in the test set, can be hard to distinguish from model variance.

More training data: less model variance, more evaluation variance

Less training data but more test data: more model variance, less evaluation variance. Holdout->repeated random subsampling->K-fold Cross-Validation->Stratification->Leave-one-out Cross-Validation in terms of which evaluation methods control evaluation bias and variance the most.

**SEQUENTIAL MODEL**

In many tasks there is “structure” between instances. **Sequential Structure:** time series analysis, speech recognition, genomic data. **Hierarchical structure:** classifying web pages within a website. **Graph structure:** deriving an influence matrix for a social network. This calls for **structured classification** models which are able to capture the interaction between instances.

**Markov Chains:** A Markov Chain describes the system that transits from one state to another according to certain probabilistic rules

A picture containing text, font, diagram, screenshot

Description automatically generatedMarkov chain assumes that state qt only depends on the immediately preceding state qt-1. Probability of observing hot hot, cold

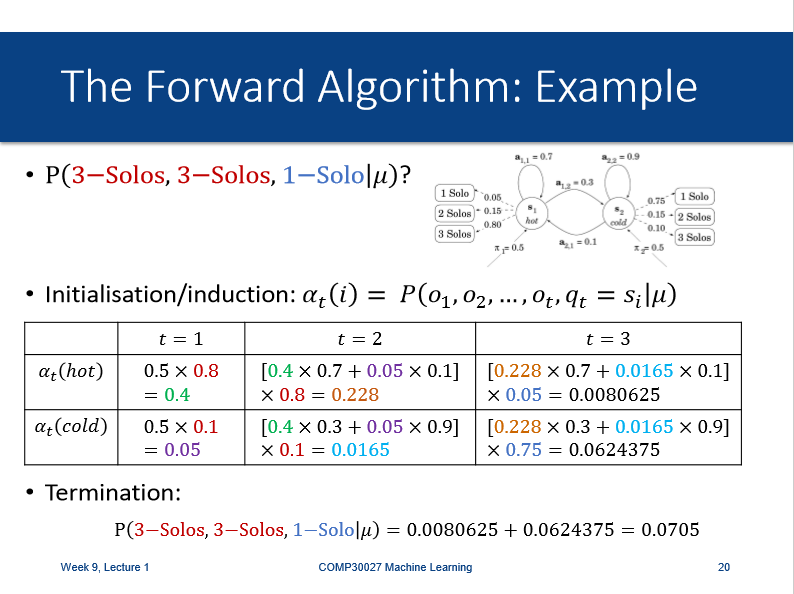
P(q1=hot,q2=hot,q3=cold)

= P(q3=cold|q2=hot)P(q2=hot|q1=hot)P(q1=hot)

**Hidden Markov Models:** Also have observations and an output probability matrix (probability of observing observation ot (yk at time t) given state i). P(ot|q1,…,qt-1,o1,…,ot-1) = P(ot|qt).

**Three Tasks: Evaluation:** estimate the likelihood of an observation sequence P(sequence|HMM model). **Decoding:** find the most probable state sequence (given a HMM model and observation sequence, determine the most probable hidden state sequence). **Learning:** estimate parameters of HMM (Transition probability matrix A, output probability matrix B, initial state distribution PI).

**Evaluation:** easy to calculate if we know the hidden state sequence, otherwise we must consider all possible state sequences. Can be done via the **Forward Algorithm** which uses dynamic programming and is more efficient.

**Decoding:** Cannot just use the most probable state for each step, as the transition probability from the most likely state at t-1 to the most likely state at t can be small -> unlikely sequence. **Brute-force:** Enumerate the probabilities of all hidden state sequences and sort. More efficient method: **Viterbi algorithm**.

A screenshot of a computer

Description automatically generated with low confidenceA screenshot of a graph

Description automatically generated with low confidence

**HMM Learning: Supervised case:** assuming we have labelled data, it is possible to use simple Maximum Likelihood Estimation to learn the parameters of the model.

No state labels? **Unsupervised case:** uses forward-backward algorithm (out of scope).

**Highly efficient approach to structured classification, but with limited representation of context (only observation and state sequences). HMM tends to suffer from floating point underflow (use scaling coefficients for forward algorithm, use logs for Viterbi Algorithm). HMM assumes state only depends on immediately preceding state, observation only depends on current state.**

**UNSUPERVISED LEARNING, MIXTURE MODELS**

**Clustering:** unsupervised learning, no explicit or implicit definition of class. Learn structure from data alone. But usually we bring our own assumptions about what kind of structure we expect in the data. **Deterministic clustering:** each instance is a member of one cluster = clusters can’t overlap. **Probabilistic clustering:** each instance has a weight in each class = clusters overlap. Choose depending on how distinct we expect categories of data to be, probabilistic clustering is slower, more complex as it has more parameters.

**K-means clustering:** Select k points at random as the initial cluster centroids. For each instance, compute the distance to each centroid. Assign each instance to the cluster with nearest centroid. Compute a new centroid for each cluster (centroid = mean of all instances in the cluster). Go to 2, repeat until no instances are reassigned.

**“Soft” k-means clustering:** Use **softmax function** (produces a vector that has the properties of a probability distribution, commonly used in many ML problems and can handle negative values. E.g. normalize the output of a multiclass classifier. Note that it doesn’t produce true probabilities.) Choose random centroids again, probability instance xi is in cluster j calculated by formula zij = …, Beta > 0 is the stiffness parameter and controls how peaky or flat distribution will be. Update each of the centroids with a weighted average of all instances. Result: overlapping clusters.

**Finite mixtures:** A distribution composed of k component distributions, used to represent subgroups or latent factors in a dataset. **Gaussian mixture model** (GMM) represents a distribution as composed of k Gaussian distributions.

**Expectation maximization (EM):** family of parameter estimation algorithms with guaranteed “positive” hill-climbing characteristics relative to the gradient of log-likelihood (each iteration improves state of algorithm). Used to estimate (hidden) parameter values, such as cluster membership.

**GMM with EM algorithm:** Assume this data is a mixture of two normal distributions, each instance is drawn from one of the two distributions: probability of drawing from distribution 1 is gamma and from 2 is (1 – gamma). Probability density is gamma \* pdf of distribution 1 + (1 – gamma) \* pdf of distribution 2. **Problem:** estimate the parameters gamma, mu1, sigma1, mu2, sigma2. Difficult to solve numerically, use iterative EM algorithm: **Expectation step:** calculate log-likelihood of instances given current parameters **Maximisation step:** update parameters to maximise calculated log-likelihood. **Guaranteed “positive” hill-climbing behaviour, fast to converge, results in probabilistic cluster assignment. HOWEVER, has an element of randomness – final model may differ depending on initial parameters. Can get stuck in local maximum; not guaranteed to find the global maximum-likelihood solution. The number of clusters (k) must be assumed.**

**Kernel Density Estimation:** Try to represent the probability distribution from which the data originated, Kernel density estimate of distribution is (Gaussian pdf(x – xi) for all instances I) / number of instances. **Model arbitrary probability distributions. No assumptions about the shape of the distribution (e.g. Gaussian). HOWEVER, need to choose a kernel bandwidth (std of Gaussian kernel which determines how jagged/smooth the final distribution will be). Need many parameters to represent the PDF; slow to compute probability at new points.**

**Why do unsupervised learning?:** Map high-dimensional data to a smaller set of clusters or latent factors, discover relationships / trends in data. Model probability density function for probabilistic models (e.g. for use In Naïve Bayes). Generate new samples from the modelled probability distribution.

**Unsupervised cluster evaluation:** No ground truth labels: need subjective evaluation, check similarity of clusters of multiple iterations, cross-validation, supervised evaluation if labels are available (check if clusters match the labels). **Cohesion:** instances in a given cluster should be closely related to each other. **Separation:** instances in different clusters should be distinct from each other. One way to evaluate cluster quality (especially for k-means) is sum of squared errors SSE (distance from each instance in a cluster to its centroid). Proximity measure is often Euclidean for numeric data, or Hamming for nominal.

**Supervised cluster evaluation: Purity:** how common the most common label is in a cluster. **Entropy:** how many different labels are occurring in the same cluster.

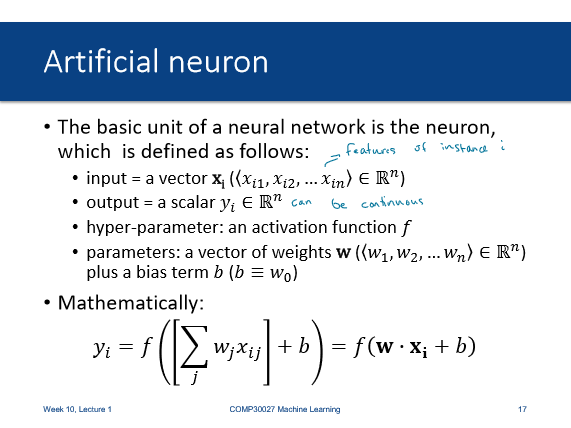
**NEURAL NETWORKS I: PERCEPTRON**

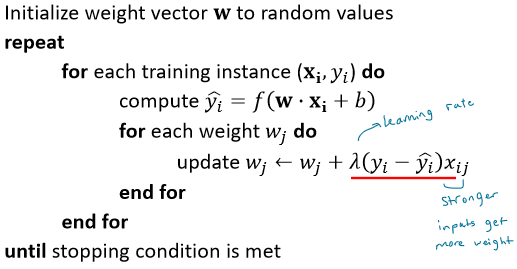
**Representation Learning:** Represent data in a format that a model can more easily use to discover relationships with. E.g. (bag of words for text, bag of pixels / rgb values for images). Representation learning is a common application for neural networks. Networks learn a feature hierarchy – from simple combinations of the input to more complex features (sometimes called **embeddings**).

**Embeddings:** Low-dimensional representations of the input, often useful for a range of tasks (not just the task on which the network was originally trained).

**Neural Networks:** Modelled after biological neurons, weighted inputs from input neurons are summed, if activation above a threshold, signal is sent to output neurons.

**A picture containing font, text, handwriting, white

Description automatically generatedPerceptron:** a neural network with just a single neuron. A perceptron is a binary linear classifier, which can be written as

**Training a perceptron:** finding the weights **w** which minimize errors on the training data. The classic way to train a perceptron is by iterating over examples in a training dataset. Each iteration over the whole dataset is called an **epoch. Perception algorithm:** For training examples, compute y based on current weights. Update weights based on the difference between predicted y and true label.

A picture containing text, diagram, line, font

Description automatically generated**Activation function:** In a perceptron: Maps a linear response to the range we want (class labels 0 or 1), in a multilayer perceptron / neural network: Adds a non-linearity after each linear operation.

**Perceptron algorithm is guaranteed to converge for linearly-separable data, but the convergence point (class boundary) will depend on: The initial values of the weights and bias. The learning rate. Not guaranteed to maximise the margin between classes. Not guaranteed to converge over non-linearly separable data (i.e. converges around small range). A perceptron with a sigmoid activation function (plus a binary step to convert the output to 0 or 1) is equivalent to logistic regression.**

**Multilayer Perceptron:** Layer = a group of neurons working in parallel on the same input. MLP: An **input layer**, at least one **hidden layer**. Each hidden layer receives the previous layer’s output as its input. An **output layer** which receives input from a hidden layer and outputs the class label. Hidden layers learn some features that are useful for the output layer (embeddings = representation learning). Neuron amount in practice determined by picking an arbitrary value in between the input and output size.

**Hidden layer:** There’s not just one “correct” output for the neurons in a hidden layer, like there is for the output layer. How can the hidden layer(s) be trained? **Backpropagation: Stochastic (random subset of input)** gradient descent. Find parameters (weights + biases) to minimize loss on the training dataset. Look at the derivative of error and update weights in the direction that would reduce error: compute errors at the output layer w.r.t each weight using partial differentiation, propagate errors back to each of the earlier layers, chain rule used to efficiently compute derivatives.

**Output Layer:** Structure depends on task. **Binary classification:** 1 output neuron with step activation function (0 or 1). **N-way classification:** N output neurons and softmax activation function. **Regression:** 1 output neuron with identity activation function (take weighted sum of inputs and adda bias, no threshold).

**Neural network/MLP properties:** Universal approximation theorem: a feed-forward neural network with a single hidden layer (with finite neurons) is able to approximate any continuous function in real numbers in n dimensions. Means the network can learn any (linear or non-linear) basis function dynamically, unlike many other methods (e.g., SVM where kernel is a hyperparameter). Note that this requires non-linearities. (If no non-linearities, multiple layers don’t accomplish anything - same computations can be done in one layer without non-linear activation functions).

**Can be adapted to many types of problems (classification, regression). Universal approximator – can model arbitrary basis functions. Representation learning in hidden layers. HOWEVER, very high number of parameters – slow to train, prone to overfitting, high memory requirements. Stochastic gradient descent – not guaranteed to converge to the same solution every time unless train on large dataset for days/weeks.**

**NEURAL NETWORKS II: DEEP LEARNING**

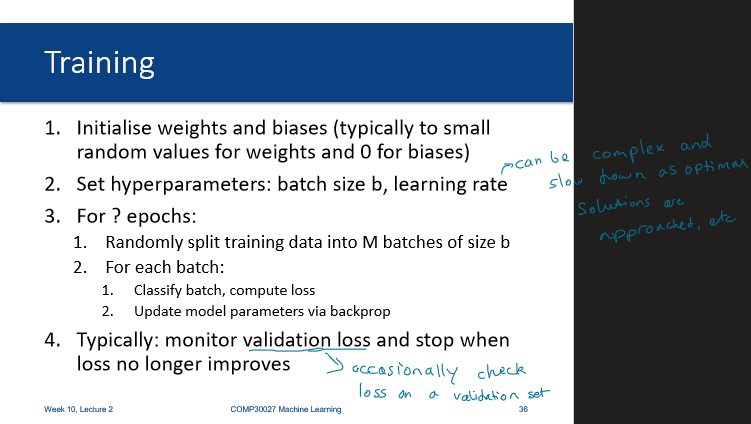
**Deep Learning:** Deep learning refers to neural networks with a larger number of hidden layers. Network **depth** = the number of layers. Network **width** = number of neurons per layer. Focus on **representation learning** i.e. transformation of raw inputs into a latent intermediate representation that is more amenable to learning.

**Convolutions:** A **kernel**, which is a matrix overlaid on the input and computes an element-wise product with the input. A **stride** which defines how many positions in the input to advance the kernel on each iteration (stride = 1 means the kernel will operate on every input), about balancing computation time vs finding all embeddings.

Layers learn a hierarchy of image features. **Fully-connected layer:** Each neuron is connected to every neuron in the last layer. The neuron learns some combination of the last layer’s responses. The output to the next layer is the neuron’s response. **Convolutional layer:** Each neuron is connected to a small patch of all the last layer’s outputs. The neuron learns a convolutional kernel. The output to the next layer is the input convolved with the neuron’s kernel. **Convolutional layers are efficient – just one neuron learns to recognize a feature like “vertical edge” everywhere in the input. Preserves spatial relations (output represents “where” feature are present, not just “what” features are present). HOWEVER, limited kernel size means model is limited to learning local features and misses many others.**

**Max pooling:** specific to images. Within a small window in the kernel’s output map, take the highest output and discard the rest. Adds **translation invariance** – network response is the same even if features are in a slightly different position in the image. Reduces computation time but discards some information.

**Dropout:** Randomly discards some neurons (set output = 0). Form of regularization: forces neurons to find useful features independently of each other, tries to prevent overfitting by ensuring that less important neurons get enough training -> better generalization.

**Training a neural network: N-way classification task = N output neurons:** Outputs are usually processed with softmax function to resemble a probability distribution. After softmax, response of each neuron is in the range 0-1, and the sum of responses to all N classes is 1. Loss = cross entropy loss (measure of the difference between the predicted class and ground-truth class). **Other outputs + loss for other tasks:** E.g. regression = 1 output neuron with identity (linear) activation function, MSE loss.

**Regularisation:** Due to the very high number of parameters, CNNs are prone to overfitting, even on large datasets. Regularisation is usually needed to reduce overfitting. Common options: L1 or L2 regularisation, dropout, early stopping. L1, L2 regularisation adds an extra term to the loss function. **L1 regularisation** adds a penalty based on the sum of the absolute values of all parameters. (Encourages many parameters = 0). **L2 regularisation** adds a penalty based on the sum of the squares of all parameters (encourages small (but non-zero) parameters). Early stopping helps keep parameters near their initial values (near 0).

**Embeddings from neural networks are good for a range of tasks.**

**Models are very sensitive to some types of noise and may not generalize to very similar tasks. In theory, neural networks make few assumptions, but in practice architecture decisions will constrain what the model can (or is likely to) learn.**

**Advantages: State-of-the-art performance, embeddings may be useful for a range of tasks (visualization for example, or feeding these embeddings into another model).** **Disadvantages: Difficult to interpret, requires large training datasets, sensitive to noise, dataset bias.**

**GENERATIVE MODELS**

**Discriminative Models:** Learn conditional probability of class Y given attributes X: P(Y|X=x)

**Generative Models:** Learn joint probability of attributes X and class Y: P(X,Y). Generative model contains discriminative model: you can use joint probability to get P(Y|X=x), AND generative can do the reverse: P(X|Y=y).

In theory, generative models have better reasoning because you have a complete model of the world. But in practice, if you only need to discriminate between some classes, a discriminative model often performs better. **Generative models can model uncertainty, detect out-of-distribution data (data significantly different from training instances, e.g. a totally new class). Can also generate new samples from a distribution.**

**Generative model approaches:** model the pdf of what you want to generate (e.g. images). Often hard to model the density function directly. Instead: Map to lower-dimensional “latent” space – autoencoders, variational autoencoders (VAE). Learn a function to convert samples from a simple PDF (e.g. Gaussian) into the target PDF, to allow sampling from the PDF – generative adversarial networks (GAN). Learn the gradient of the PDF and move along the gradient to generate more-probable samples – score-based models, diffusion models. Training is unsupervised (or self-supervised), requiring no human labels. Usually, hide part of the input and ask model to predict it. (e.g. Masked language modelling – predict missing words in sentences, image denoising – remove (Gaussian) noise from images.)

**Autoencoders:** Essentially, neural networks for unsupervised learning. Output of the network is whatever was passed to the network (e.g. an image). Hidden layer learns a lower-dimensional (latent) representation of the input. Different types – “variational autoencoder” adds constraints to the basic autoencoder so the latent layer can be used to generate new samples.

**Hidden layer:** “Bottleneck” layer: - smaller than the input. Represents the input in terms of **latent variables** (in the simplest case, one hidden layer with linear activation functions, this layer learns PCA.) The hidden layer is smaller as we are trying to reduce the dimensionality. **Output layer:** Unlike a standard NN, the output is not a class or regression value – it’s the same type as the input (e.g. an image). Activation function is chosen appropriately (tanh or sigmoid for binary image, linear activation for a grayscale / colour image). **Learns a smaller, latent variable representation of the input. Can learn this representation over complex features. Variational autoencoders can be used to generate new instances. HOWEVER, deeper versions can be difficult to train.**

**Generative Adversarial Networks (GANs):** neural networks that learn to generate instances from a particular distribution (e.g. images of faces). Actually consist of two neural networks: a **generator** and a **discriminator**. Training involves a sort of competition between the two networks. **Generato**r learns a probability distribution over inputs. It does this by sampling from a distribution (e.g. Gaussians) and learning a function to map from this distribution to the input Architecture looks like a reversed neural network (from distribution numbers scaling up to a full input). **Discriminator** learns to identify real inputs vs. fake inputs created by generator. Neural network classifier with two output classes (real, fake). Architecture depends on task (e.g. for images the discriminator might be a CNN with several convolutional layers. The networks are trained together on a combination of real data **x** and generator input **z.** Discriminator wants to maximise D(**x**) and minimize D(G(**z**)). Generator wants to maximise D(G(**z**)). Can treat as a zero-sum game with the goal of finding equilibrium between G and D. **If the discriminator is too good:** Easily rejects all fake inputs, not much information to train the generator. **If the discriminator is too bad:** Easily confused by fake inputs that don’t look real, generator will learn a poor solution. Hard to find a balance between discriminator and generator.

**Evaluating GANs:** Outputs should look like inputs (look “real” and not “fake”, REALISM). Outputs should not be identical to inputs (memorized training data, MEMORISATION). Outputs should be as diverse as real data (avoid **mode collapse** = the generator only creates one or a few outputs, DIVERSITY). First two are easier to evaluate. Evaluate realism manually by inspection, evaluate memorization by comparing outputs to nearest neighbours, take a sample of outputs and count duplicates to check diversity. Should have similar diversity as present in training set. **Advantages: Model, and generate samples from, complex probability distributions. Disadvantages: Can be unstable / hard to train. Difficult to evaluate. Even when the performance looks good, the learned probability distribution may not actually be correct.**

**“BIG DATA”, DATA AUGMENTATION, AND ETHICS**

**Importance of data:** simple models with a lot of data beat complex models with less data. (Labelled) data is a bottleneck for machine learning, model depth has increased dramatically but the size of “large-scale” datasets has not kept pace. Adding data is nearly as effective as adding layers, more parameters are not helpful unless you have more data to train them. Data is (often) abundant but labelling is expensive and slow.

**Data augmentation:** There are various ways to expand a labelled training dataset. General: resampling methods, Data-specific: add artificial variation to each instance, without changing ground truth label.

**Bootstrap sampling:** Create “new” datasets by resampling existing data, with or without replacement. Common in perceptron and neural network training (“mini-batch”, “batch size”), methods that involve stochastic gradient descent. Each “batch” has a slightly different distribution of instances, forces model to use different features and not get stuck in local minima.

**Data manipulation:** add a small amount of noise to each instance to create multiple variations. (E.g. change image brightness, resize, rotate, flip, shift, adjust audio volume or frequencies, synonym substitution for text, etc.). These perturbations should not change the instance’s label. Generally, they should be the same kind of variations you expect in real-world data.

**Data synthesis:** Create artificial data using another machine learning method: Train a probability distribution on labelled data, sample the probability distribution to produce new instances. Generative adversarial network: neural network trained to create samples from a distribution. Exploit algorithms designed for other tasks, e.g.: Computer-generated images, automatic translation.

**Advantages: More data nearly always improves learning. Most learning algorithms have some robustness to noise (e.g., from machine-translation errors). Disadvantages: Biased training data. May introduce features that don’t exist in the real world. May propagate errors. Increases problems with interpretability and transparency.**

**Semi-supervised learning:** learning from both labeled and unlabeled data. Training data consists of L labeled instances <xi,yi> and U unlabeled instances <xi>. Often U >> L. Learn a better classifier from L and U than is possible from L alone.

Simple approach: combine a supervised and unsupervised model. E.g. find clusters, choose a label for each (could be most common label) and apply it to the unlabeled cluster members.

**Self training:** Assume you have labelled data L and unlabeled data U. Train a model fi on L using supervised learning method. Apply fi to predict labels on each instance in U. Identify a subset of U, U’ with “high-confidence” labels. Remove U’ from the unlabeled dataset and add it to the labeled set with the classifier predictions as “ground truth” labels. Repeat until L does not change. Aka “bootstrapping”.

**Propagating labels requires some assumptions about the distribution of labels over instances: Points that are nearby are likely to have the same label. Not really creating data from nothing. Classification errors are also propagated (One option: move points back to the “unlabeled” pool if the classification confidence falls below a threshold.**

**Active Learning:** A classifier could achieve higher accuracy with fewer training instances if it were allowed to have some say in the selection of the training instances. Labeling is a finite resource, so instances should be labeled in a way that maximises learning. Active learners pose **queries** (unlabeled instances) for labeling by an **oracle** (e.g. a human annotator). Ideally, we’d use the instances that are most effective for distinguishing between competing models. To do this most efficiently, we should have some sense of the likelihood of different models, or knowledge of how labels are distributed over instances, which usually isn’t the case. In machine learning, querying generally focuses on instances with high uncertainty, e.g.: Instances near the boundaries between classes, instances in regions with few labels.

**Query strategies:** Which unlabeled instances will be most useful for learning? One simple strategy: query instances where the classifier is least confident of the classification. Alternatively, margin sampling selects queries where the classifier is least able to distinguish between two categories. Or better still, use entropy as an uncertainty measure. A more complex strategy, if you have multiple classifiers, **query by committee (QBC):** Train multiple classifiers on a labeled dataset, use each to predict on unlabeled data, and select instances with the highest disagreement between classifiers. Assumes that all the classifiers learn something different, so can provide different information, disagreement measured by entropy.

**Active Learning Practicalities: Empirically, seems to be a robust strategy to increase accuracy. HOWEVER, often difficult to justify these strategies theoretically, introduces bias, resulting in a dataset that might not be as useful for other machine learning tasks, may be sensitive to label noise.**

**Data considerations:** Does the training dataset accurately reflect the real world? Tendency to use data that is convenient, not necessarily fair or representative; e.g., movie transcripts, Twitter.

Do you want to accurately reflect the real world? – Under sample minority groups, replicate historical biases

Do you have the right to use this data to train an AI model? – Data ownership, copyright, privacy.

**Dataset bias:** What happens if some groups are less represented in a dataset? – Less contribution to loss function, so potentially more errors on this group. Poorer model fit in this group due to limited training data. Poorer generalization to new examples of this group due to lack of training diversity. Note that continuous learning methods can exacerbate this problem.

**Mitigating dataset bias:** Treats as an imbalanced dataset problem – Use data augmentation or oversample to increase samples from underrepresented group, Adjust loss function to put more penalty on errors in underrepresented group. Force model output to be indepdent of some variable (e.g., race or gender). **Requires you to know about the bias in your dataset, doesn’t solve the problem of low diversity.**

**Check your dataset for biases that you can think of, but be aware that datasets may be biased in many ways you haven’t thought of. The fact that something is on the internet doesn’t necessarily mean you can use it to train AI – there may be legal / ethical issues around using data in this way, even if it is “publicly available”**

**Labeled data is major bottleneck for machine learning. There are various strategies for making use of unlabeled data, or making more effective use of the data and labeling resources we already have. These strategies generally improve performance, but sometimes with a trade-off in terms of error propagation, bias, and interpretability.**

From my understanding, we should probably know the Logistic Reg derivation, MSE, RMSE, Weighting methods in k-NN, IG, GR (including split info), entropy, all eval metrics, distance metrics, Unsupervised learning metrics maybe.......we really did not spend too much time on this one. Kmeans I'd imagine we need to know for sure.

GMM formulas are hard and specific. EM the same. HMM probably hard to memorise but concept we should know. SVM maths no way. Maybe the optimisation functions for soft and hard margins.

If staff could confirm this that would be great.

That looks correct to me. We'd also expect you to know things like the perceptron equation and the perceptron training algorithm, basic probability that we covered in lecture, the Naive Bayes formula and how to do NB smoothing.

You should understand the concepts behind the more complex formulas (like what is done on the E and M step of E-M) but if we have a question specifically related to the formula we'll provide it.

Reply