|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ML Pipeline | | | | | Collection, Extraction (Feature Engineering), Understanding (visualisation), preprocessing, Model choice, training, validation, deployment | | | | | | | | |
| Learning algo | | | 2 components of the learning algo: 1) Hypothesis set/Representation for h, H = {h1, ..., h|H|} (can incur representation bias)  2) Learning algo A to select h H s.t. h ≈ f (target fn) w high accuracy (can incur procedural bias)  Both make up the learning model. We choose H, and learning algo A then select h. Each A has a diff inductive bias  Futility of Bias Free Learning: A learner that makes no a priori assumptions on f has no rational basis for classifying any unseen instances  Making prior assumptions - Inductive Bias (= representation + procedural) - is the only way to make learning feasible. | | | | | | | | | | |
| KNN | Distance Metrics: Euclidean dist/L2(**X**i, **X**j) = [sum(xi - yi)2]1/2. Any dist metric (non-negative, symmetric, obeys triangle inequality) would work  Possible issues: Discrete data: Non-ordinal -> one-hot encoding. Ordinal -> assume standard interval ranges or use one-hot encoding  Diff value ranges: some attributes may dominate the dist calculation -> use range normalization | | | | | | | | | | | | |
| Smaller k: complex surface. Larger k: smoother surface | | | | | | | | | | | | When k = 1, all points classified to their own label |
| Aggregation over k: majority vote (predict class based on most frequent among knn) | | | | | | | | | | | | Weighted vote: vote according to dist of neighbours |
| Decision Tree | | Need to choose: 1) how to pick feature to split on. 2) How to discretize cts feature. 3) Where to prune trees | | | | | | | | | | | |
| 1) To ChooseAttribute, use Entropy to measure purity, H(X) = , where K = num of labels in X, Pi = prob of class i  A chosen feature Vi divides the dataset, S into subsets S1, ..., SC according to the Ci distinct values for Vi  remainder(S, Vi) = . Information Gain, IG(S, Vi) = H(S) - remainder(S, Vi)  Choose Vi that gives the largest IG. (entropy and IG have unit bits). Note: entropy = 1: prob = 0.5 (most unpredictable) | | | | | | | | | | | |
| 2) To choose from an infinite num of split pts for a cts feature Vj: | | | | | | a) Observe that moving split pts btw 2 observed values w the same label has no effect on IG  b) Hence we only split btw examples from diff labels y, to improve IG | | | | | |
| 3) To decide where to prune: a) Don't let tree grow too deep  b) Don't let leaves get too small (require leaves to have at least i training data) | | | | | | | | | | c) validation -> remove node and replace by MODE of labels y if pruned tree performs better than original | |
| Efficient: both in training (greedy search) and testing  Instability: susceptible to small fluctuations (high var) | | | | | | | Hard decision boundaries: by default, no probabilistic interpretation of boundaries  Axis decisions: doesn't capture interaction btw features (feature & bug) | | | | |
| Evaluating model | | | | | Cost/loss fn L(, f) to measure diff btw and f. This is almost always a pointwise definition: l((x), f(x))  Overall cost L(h, f) = average of pointwise cost l(h(**X**), f(**X**)). Training cost: Ltrain = . Test cost: Ltest = Ex{l((**X**), f(**X**)} | | | | | | | | |
| Curse of Dimensionality | | | | | | | Sparsity problem: maintain density of samples need exponential growth of data. Generate more points, s.d. grows slowly | | | | | | |
| 3 Learning problems | | | | | | |  |  |  | | --- | --- | --- | | Classification | Regression | Logistic Regression (prob) | | ... | ... | ... | | | | | | | | |
| Classification | | | | | | h(**x**): (where x0 = 1 by default, i.e. bias). By default, vectors are col vectors. So h(**x**) =  With an entire matrix **X** of instances (each **x** in 1 row, suppose only 2 features): do **X** = [; ] = [; ]  So hypothesis fn (x) = sign() (which creates a hyperplane w threshold) and approve credit if y = 1, deny if y = -1 | | | | | | | |
| Linear Regression | | | | Linear regression output: (**x**) = = **.** For entire matrix, do **X**. -valued cost fn: make use of squared error: ((**x**) - f(**x**))2  Training error/cost fn: Ltrain(h) = , where is the actual value for | | | | | | | | | |
| 1) Normal Equation: No need to choose . No need to iterate. Need to compute , an O(n3) operation  2) Gradient Descent: Works well, even when n is large. Works even if is non-invertible | | | | | | | | | |
| 1) Normal Equation: Ltrain ... differentiate ... Hence **.** Note = **a**, and = 2A**x.** = pseudo inverse of **X**. Matrix inverse = O(n3) operation | | | | | | | | | |
| 2) Gradient Descent (works because Ltrain is a convex fn of , i.e. has a min pt)  Is an iterative general mtd for nonlinear optimization. Start at (0); take a step in the opp dirn of the steepest slope  Goldilocks step size: if n too small, takes too long to converge; if n too large, might not converge; need just the right n  Hence, should use = –Ltrain((t)) instead of = nv = –n, where = fixed learning rate instead of n = fixed learning step | | | | | | | | | |
| Logistic Regression | | | | Logistic/sigmoid function: g(s) = . g(s) maps any x: to y: (0,1). (x) = g(s) is interpreted as a probability, where s =  Labels, y are binary but output is probabilities (we are inferring a probability from event labels). Soft threshold: uncertainty  P(y|**x**) = . Note f: [0,1] is the probability. We are trying to find (**x**) = g() to approximate f(**x**)  Cost fn for logistic regression: Ltrain() = = cross entropy (CE) error/logistic loss | | | | | | | | | |
| If = f, we can write P(y|**x**) = g(y \* ) (since g(-s) = 1 - g(s))  Likelihood of **X** where samples are iid is: P(,...,|,...,) = (since independent) =  max likelihood = min CE | | | | | | | | | |
| Gradient Descent for Logistic Regression | | | | Using gradient descent for finding :  1. Initialize weights at t = 0 to (0)  2. Compute gradient (t) = Ltrain((t)) = | | | | | | 3. Move in dirn v(t) = -(t). i.e. update weights (t+1) = (t) - Ltrain  4. Continue to next iteration, until termination condition reached  5. Return final weights \* | | | |
| Termination Condition: 1. Error change is small and/or 2. error is small; 3. max num of iteration reached | | | | | | | | | |
| Stochastic Gradient Descent | | | | (t) only considers 1 random point (x\*, y\*)  "Average" dirn: [–l((**x**(\*)), y(\*))] = = –Ltrain | | | | | | | Benefits: 1. Cheaper computation (1/m cheaper per step)  2. Stochastic: Helps escape local minima 3. Simple  - Might require more steps to converge but overall faster | | |
|  | | | | Combine both GD and SGD: use mini batch gradient descent, i.e. pick a few (**x**(\*), y(\*)) at a time | | | | | | | | | |
| Support Vector Machine (SVM) | | | | SVM creates better linear separation (by maximizing margin). Also inherently handles noisy data. Has unique h(**x**)  Hyperplane eqn: h(**x**) = + b. dictates orientation of plane; b dictates offset (bias)  Define dist d to optimal plane as "1" (unit distance). + b = 1 (+ve class) and + b = -1 (-ve class)  Note: only a subset of the dataset/support vectors determines our unique h(**x**), which are the most difficult instances to classify  Inductive bias of SVM: dataset is separable (i.e. distinct classes are separated by a wide margin) | | | | | | | | | |
| Non-linear mapping | | | | Linear regression, and linear classification, sign() works because of the linearity of weight. But won't work when y is affected by xi in a non-linear way. is linear in . Any fn : **x** **z** preserves this linearity. Note: **y** & = (, ..., ) are not transformed  So in overview of pipeline, in training examples, use **z**1, ..., **z**m instead of **x**1, ..., **x**m | | | | | | | | | |
| Kernel | | | | In X space, "pre-images" of support vectors. SVM natively handle non-linear transformations through the use of kernels. Transformation still maintains the margin in the Z space (i.e. dist btw line and pt in X space is same in Z space)  Great generalization, since in model, num of params num of support vectors  A kernel is a fn that returns a dist/similarity measure of 2 instances (often an inner product). K(x, x') = zTz' for some z space | | | | | | | | | |
| Soft Margin SVM | | Rationale: want our SVM to have a huge margin even if it misclassify some points (if perfect fit, margin might be small -> overfitting)  y(\*)((\*) + b) ≥ 1 means prediction is correct (since prediction and actual label of the same sign).  If y(\*)((\*) + b) < 0: pt is misclassified. If 0 ≤ y(\*)((\*) + b) < 1: pt is a margin violation (pt btw margin and correct side of hyperplane)  Putting eqn all tgt, y(\*)((\*) + b) ≥ 1 - , where ≥ 0. If = 0: correct prediction, 0 < ≤ 1: margin violation; > 1: pt misclassified  is known as the slack variable: soft error on (x(\*), y(\*)). Total violation = | | | | | | | | | | | |
| Loss fn: Hinge Loss. Ltrain() = = width of margin + penalty for misclassification and margin violation (hinge loss)  In hard margin SVM, there are no misclassification (i.e. only concern with width of margin)  In diagram: If pt at 0 (i.e. on margin line), penalty = C. As pt more misclassified, penalty becomes larger  Soft error C parameter: how badly we want to penalise misclassification and margin violation  Small C = more tolerant = wider margin. Higher C = less tolerance = smaller margin. At very high C hard margin SVM | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Perceptron | | Linear classification. x2 = mx1 + c w2x2 + w1x1 + w0 = 0.  , x0 = 1. | | | | | | | Perceptron classification. xr = inputs, wr = weights, Activation fn g = sgn. | | | |
| Perceptron Learning Algo (PLA) | | 1. Initialize weights **w** (could be all 0, or random small values). | | | | | | | | | | |
| Perceptron Weight Update (i.e **w**). **w** **w** + **x** (new = old + learning rate \* learning error \* **x**), i.e. **w** = **x**  Cosine similarity: s = cos() = = . (If s = 1 -> correct classification) | | | | | | | | | | |
| Activation Fns | | Graphical user interface  Description automatically generated with medium confidenceGraphical user interface  Description automatically generated with medium confidenceDiagram, schematic  Description automatically generatedDiagram, schematic  Description automatically generatedIf g (activation fn) is diffferentiable, we can use gradient descent to find min of error faster | | | | | | | | | | |
| Gradient Descent | | Iteratively find wi with min error (loss function)  So (**w**) = (g(f(**w**))), where = (g), = g = g(f), f = f(**w**)  , Note f = **w**T**x**, and = = **x**, so | | | | | | | | | | Gradient Descent Weight Update  **w** **w** - , where is dirn of fastest error increase  [; ...; ] |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Error |  |  | Activation = g(f) |  |  | Weighted sum f(**w**) |  | | MSE |  |  | Sigmoid |  | (1 – g)g | **w**T**x** | **x** | | Binary Cross Entropy |  |  | tanh | tanh f = | 1 – g2 |  |  | | Error |  |  | ReLU | max(0, f) | [f > 0] |  | xi | | Categorical CE |  |  | Softmax |  |  |  |  |   E.g. using MSE + Sigmoid + wTx: | | | | | | | | | | |
| Neural Networks | NN = Multi-layer perceptron (diagram shows 2-layer MLP). RMB ADD BIAS TERM  General idea: [layer l activations = layer l activation fn((layer l weights)T layer l-1 activations) ]  Universal Approximation Theorem: Each neuron contributes a piecewise fn. Many piecewise fn can approximate a curve | | | | | | | | | | | |
| Multiclass / Multilabel NN | | For single output NN: g =  Note , = prob for class c | | | | | | 1) = Categorical Cross Entropy = (for multiclass classification)  2) = Euclidean dist = (for vector regression) | | | | |
| |  |  |  | | --- | --- | --- | | Sigmoid | Multiple Sigmoids | Softmax | | Is prediction true? | Which predictions are true? | Which class is most probably true? | | For binary classification | For multi-label classification | For multiclass classification | |  |  |  | |  |  |  | | | | | | | | | | | |
| Notation (n = num of features, m = num of instances in dataset) | | | | | | | Vector-by-vector: **y**(**x**) = w**x** = w = (e.g. ). Matrix-by-matrix: 1) Hadamard product (elem wise multiply ): **y**(**X**) = 2) Convolution operator \* (elem wise multiply then sum): **Y**(**X**) = **W\*X** | | | | | |
|  | | GD weight update: **W** **W** - . (perceptron) & . Note | | | | | | | | | | |
| Gradient Descent with backpropagation | | | | | Backpropagation efficiently computes gradient by avoiding duplicate calculations, not computing unnecessary intermediate values,  Gradient of weight input of each layer if calculated from back [l+1] to [l]: , and  Reverse Polish Notation (forward propagation): (**x**[0], **W**[1])f[1]g[1]...f[L]g[L] = (calculate from left to right) | | | | | | | |
| Deep Neural Network (DNN) | | | | | | DNN = many hidden layers (≥ 3). Shallow network (input, 1 hidden, output). More parameters -> need more data. | | | | | | |
| CNN Convolutional Filters | | | | Kernel size: how many pixels to consider at once. Stride: how many pixels to move by. Padding: how many extra pixels to add to input  E.g. kernel size **k** = {32}. Padding **p** = {(2+2)0} (2 top + 2 bottom, 0 left right) Stride **s** = {12} (down, right)  dim **y** = = = {43}. Output = activation map (2D matrix) | | | | | | | | |
| Multi-Channel | | | RGB image: apply filter for each RGB channel -> get 3 activation map. Add the 3 activation map element wise to get 1 final 2D matrix | | | | | | | | | |
| Multiple Convolution Kernels | | | {}. (**W**[l] is a 4D matrix) {}  Each col in **W**[l] represent 1 filter. Each row represent 1 channel | | | | | | | | | |
| Fully Connected (FC) vs Conv. Layers | | FC layers: Each layer has multiple activations  Each activation is a 0D scalar  Layer (of multiple activations) is a 1D vector  All weights connect activations of previous layer (1D) to current layer (1D)  All weights represented as a 2D vector  (FC) vs  (Conv) | | | | | | | | | Conv Layer: Each layer has multiple activation maps  Each activation map is a 2D matrix. Each map is on a different channel (1D)  Layer is a 3D matrix  Kernels convolves on activation map (2D) of all channels (1D) in previous layers, then summed  Each kernel represented as a 3D vector, **W**1 = [**W**11; **W**12; **W**13]  Each kernel stored as separate filters (1D)  All kernels represented as 4D vector | |
| Convolu-tional Layer | | Num of kernels k, Kernel size **k**, Padding **p**, Stride **s**  k[l-1] (num of filters from previous layer) = c[l] (num of channels into current layer) | | | | | | | | Pooling layer: - downsamples feature maps - reduces dimensionality  - help to train later kernels to detect higher-level features  - Aggregation mtds: Max-pool (most common), average-pool, sum-pool | | |
| Key concepts | | |  |  | | --- | --- | | Learn spatial Features | Series of multiple convolution + pooling layers. Progressively learn more diverse and higher-level features | | Flattening | Convert to fixed length 1D vector | | Learn Nonlinear features | With fully connected layers. Learns nonlinear relations w multiple layers (analogy: semantic reasoning) | | Classification | Softmax (multiclass logistic regression). Feature input = image embedding vector | | | | | | | | | | | |
| Recurrent NN (RNN) | | To process sequential data (e.g. speech, music, sentiment classification, DNA sequence analysis, translation)  Whole model: . At each "time": and  i.e. current output depend on historical inputs. LSTM uses input, output, forget gates.GRU (gated recurrent unit) uses update, reset gate | | | | | | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bad data | 1. Erroneous data: noise in instance values or labels. 2. Irrelevant data: Refers to features/attributes. 3. Imbalanced data: Refers to class/labels.  4. Missing data: Refers to instance values. 5. Too many features (high dimensionality): Curse of dimensionality | | | | |
| Issues with data | | | 1. Linear separability 2. Curse of Dimensionality 3. Imbalanced Data | | |
| 1. Linear separability | | - Many models assumes data features are linearly separable | | | |
| How to check for linear separability. a) Can use scatterplot matrix to visualize linear separability  b) Use computational metrics like linear SVM, reduce dimensions (LDA, PCA) or others (linear programming, convex hulls)  bi) Linear Soft-Margin SVM: Margin violation: y(\*)((\*) + b) ≥ 1 - , where ≥ 0. Total violation:  Each is the dist that the misclassified point i is from its correct margin. Higher violation = less linearly separable  Chart, scatter chart  Description automatically generatedbii) PCA vs LDA: visualise separability with new basis vectors (i.e. change axis)  PCA: find axis that maximizes variance of data points. 2nd axis next best maximises variance  LDA: find axis that best distinguishes classes in data. Maximises F =  Similar projection for LDA, but F1 > F2 (F for 1st axis > F for 2nd axis)  PCA & LDA: All axes are orthogonal. PCA: supervised and unsupervised learning. LDA: supervised | | | |
| How to mitigate it  1) Find useful features  - feature extraction (collect new features) - feature selection | | 2) Transformation of features:  - features engineering (x -> x2) - change basis vectors (e.g. PCA, LDA)  - Kernel trick (e.g. for kernel SVM) - Feature learning (e.g. NN) | |
| 2. Curse of Dimension-ality: more features than data instances (n ≥ m | | - Data too sparse to inform about true decision boundary (for classification) -> easy to overfit model on sparse training data | | | |
| How to check? Visualise histogram of distances (check for variance) | | | Or more simply: just aim for n < m/5 |
| How to mitigate it? 1) Feature selection: wrapper methods or filter methods 2) Dimensionality Reduction  1a) Wrapper methods (e.g. Recursive Feature Elimination, RFE) (use elbow method to determine num of features to keep)  Train model M w features -> For each feature, measure decr in M performance -> Eliminate feature w least decr -> Repeat  1b) Filter methods: Mutual information = Information gain OR use correlation  2a) Linear Matrix Factorisation (e.g. PCA, LDA). 2b) Non-linear Manifold Learning (e.g. SOM, MDS, t-SNE, UMAP). 2c) Deep Auto-Encoders | | | |
| Benefits of feature selection: avoid curse of dimensionality, faster model training - Fewer features -> easier to interpret | | | |
| 3. Imbalanced Data | | Lead to misleading evaluation metrics, and model overfitting to majority class | | | |
| How to check for it? Visualise histogram or bar chart of feature values | | | |
| How to mitigate it? Collect more data instances OR resamples instances (undersampling, oversampling, SMOTE)  Undersampling: sub-samples of majority class. Oversampling: duplicates of minority class  Data leakage (snooping): first split dataset to train-test, then resample train and test datasets separately  Synthetic Minority Oversampling Technique (SMOTE)  1. Consider minority and majority instances in vector space 2. For each minority-class instance pair, interpolate their features values  3. Randomly synthesize instances and label w minority class 4. More instances added to minority class | | | |

|  |  |  |
| --- | --- | --- |
| Feature Extraction & Engineering | | Tabular data: - use custom eqns - counting, aggregation, difference, min, max as new features, then select most impt feature |
| Temporal features: - previous value, aggregate, - moving average, moving SD, moving range (min, max), moving trend (slope of linear fit) |
| Image features: scale image to square shape -> decompose into channels -> concatenate and represent as 3D tensor  - size (may not be good due to diff zoom levels), - color dist (histogram across RGB channels); not ideal as some image have same colors  - Edge detection for shape features (convert to grayscale measure color c at each pixel position px compute 1st derivative ; high magnitude indicates edge In 2D, calculate gradient threshold; > cth i.e. above cth confirm is edge)  Prewitt operator/kernel: convolutional kernel to detect vertical edge, (horizontal edge)  Note . Same for . So , (where represent convolution operator)  Should use dimensionality reduction (e.g. PCA) to reduce num of features generated. (Not Tested: Scale-invariant feature transform, SIFT)  - texture: use convolutional filters |
| Text features: - text length, keywords, non-keywords/stop words  - Approach: extract words word variation uninformative words Identify informative words  - Extract words: tokenization: split string of text by delimiters  - Word variation: Stemming: remove word endings 'ed', 'ing', 'ly'... (easier to implement, faster)  Lemmatization: replace word w base form (harder to implement, slower)  - Uninformative words/stop words: NLTK have list of stop words (e.g. and, all, we, those). May remove informative words (e.g. not)  - Informative words: Bag of Words (BOW) Encoding. Basically one hot encoding but for words, then count freq of each word |
| Bias & Variance | Bias: diff btw average prediction and true value (higher bias -> higher diff -> less accurate). Variance: spread of prediction  High bias -> low variance. Low bias -> High variance. Match model complexity to data resources (quality & qty), not target complexity  Simpler model: High bias (high loss), low variance (converge quicker). Complex model: Low bias (low loss), high variance (converge slower) | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Overfitting | | | Overfitting: Ltrain = 0 but Ltest is huge. Fitting the data more than necessary. Overfitting fit the noise as well which is harmful  Approximation-Generalisation Tradeoff: Small Ltest = good approximation of f on unseen test data  More complex = better chance of approximating f. Less complex = better chance of generalising on test data. Ideal = {f} | | | | | | |
|  | | | In KNN: Higher k: lowers variance (but incr bias)  In DT, pruning lowers variance (but incr bias) | | | In Linear model: each additional parameter make model more complex  In SVM: num of support vectors complexity of model | | | |
| Ensembles | | | How to aggregate?  1. Select most trustworthy of them based on their usual performance  2. Let each model have a vote (uniform vote) | | | | | | 3. Weight model non-uniformly (used in random forest)  4. Combine predictions conditionally  Ensembling diverse models/hypotheses generalise better |
| Analysis of Bias & Variance | | | Decompose Ltest into 1. How well can approximate f overall  2. How well we can zoom in on a good  Applies to real-valued targets and uses squared error (for binary targets, use VC analyses; related to SVM??) | | | | | | Ex[z(x)] = expected value of z(), given dist of values of x  hD = hypothesis of learner trained on Dataset D  ED[z()] = expected values of z(), given dist of possible datasets D |
|  | | | Ltest(hD) = Ex[(hD(x) - f(x))2] (loss on test set with model hD). ED[Ltest(hD)] = Ex[ED[(hD(x) - f(x))2]] (expected loss on test set with model hD)  Thus, ED[Ltest(hD)] = Ex[ED[(hD(x) - f(x))2]] = Ex[var(x) + bias(x)2] var + bias. Bias[h(x)] = |(x) - f(x)|. Var[h(x)] = E[(h(x) - (x))2] | | | | | | |
| Noise | | | Stochastic Noise: random data error  Re-measure y: stochastic noise change  Change : stochastic noise still the same | | Deterministic Noise: the part of f we lack the capacity to model (model error)  Re-measure y: Deterministic noise the same  Change : Deterministic noise changes | | | | |
| Noise and Bias-Variance | | If f is a noisy target: y = f(x) + (x). E[(x)] = 0  [(hD(x) - y)2] = [(hD(x) - f(x) - (x))2] = [(hD(x) - (x) + (x) - f(x) - (x))2] = [(hD(x) - (x))2 + ((x) - f(x))2 - ((x))2 + cross terms]  ED[(hD(x) - f(x))2] = ED,x[(hD(x) - (x))2] + Ex[((x) - f(x))2] + ((x))2 var + bias + var + deterministic noise + stochastic noise | | | | | | | |
|  | | | To reduce overfitting: 1) Regularisation (restrain the model). 2) Validation (reality check by peeking at the bottom line) | | | | | | |
| Validation | | | Both regularisation and validation: Ltest(h) = Ltrain(h) + overfit penalty.  Validation tries to do a "mock test". Dataset is now split into train, validation and test set. Lval estimate Ltest | | | | | | |
| Estimated loss | | | On a test point (x,y), the cost I((x), y) is Squared error: ((x) - y)2. Binary error: [[(x) ≠ y]] (Iversons bracket)  E[I((x), y)] = Ltest(). Var[I((x), y)] = | | | | | | |
| On a validation set (**x**(1), y(1)), ..., (**x**(K), y(K)), the cost is Lval(h) = . Lval(h) = Ltest(h) ± O()  E[Lval(h)] = = Ltest(h). Var[Lval(h)] = = (K points are iid, covariance = 0) | | | | | | |
| Validation | | | Retrain model on entire D, to get h. Use h in the end  Rule of thumb: K = n/5 | | | | O(1/). Small K: bad estimate of Ltest.  Large K: h- trained on too few examples, lead to bad h-, bad estimate | | |
| Error estimates: Ltrain, Ltest, Lval. Contamination: deceptive bias in estimating Ltest  Training set: totally contaminated. Validation set: slightly contaminated. Test set: totally clean | | | | | | |
| Cross Validation | | | Ltest(h) ≈ Ltest(h-) (when K is small). Ltest(h-) ≈ Lval(h-) (when K is big) | | | | | | |
| Leave out one CV (LOOCV): Hypothesis learned will be highly correlated, as most points are identical  Final hypothesis learned from Dcv is . lcv = lval() = l( (**x**(cv)), y(cv)). Cross validation cost: Lloocv = | | | | | | |
| K fold CV. K training sessions on n/K points each time. Recommend: 10-fold CV  If CV were averaging independent estimates: LOOCV shld see lower var, since training sets overlap substantially  But since our training sets are highly correlated: LOOCV has high variance, so K-fold CV better | | | | | | |
| Evaluation metrics + Confusion Matrix | | | | Correctness = [ = y], where = M(**x**) is predicted value from model M for data **x**, y is true value, [] is Iverson bracket  Accuracy = "average correctness" = . Which class is positive/negative defined based on application | | | | | |
| Cost-Sensitive metrics  1. Precision & Recall | | | Recall = . Among actual positives, what fraction of instances were recalled? Maximise this if FN is costly (e.g. cancer prediction)  Precision = . Among +ve predictions, how precisely were +ve instances predicted? Maximise this if FP is costly (e.g. email spam)  F1 score = = . F1 score is more robust (less sensitive to extreme values) | | | | | | |
| Cost-Sensitive metrics  2. Vary Prediction Threshold | | | TPR = . FPR = . To summarise varying threshold results, use ROC (Receiver Operating Characteristic) & AUC (AU ROC Curve) | | | | | | |
| ROC draws TPR vs FPR. Diagonal line indicates 50% chance of correctness. If ROC curve above random line, model is more accurate than chance. Perfect curve has TPR = 1 and FPR = 0 always | | | | | | |
| AUC is a concise metric instead of a full figure. AUC > 0.5 means model is better than chance. AUC ≈ 1 means model is very accurate | | | | | | |
| Precision-Recall (PR) Curve AUC for imbalanced classification. ROC would not show that well for imbalanced data | | | | | | |
| Multiclass Classification | | | Basically treat 1 class as positive, other classes as negative. For 3 classes, get 3 diff confusion matrix | | | | | | |
| Micro-Average (average then metric): = average of TP across the 3 confusion matrix. Same for , and  Then for new confusion matrix (on RHS), calculate micro-average accuracy, precision, recall, F1, AUC | | | | | | |
| Macro-Average (metric then average): Calculate A, P, R, F1, AUC for each confusion matrix. (For 3 class, get 3 diff A, P, ...). Then average | | | | | | |
| Micro weighs each instance equally. Accounts for imbalanced data. Macro weighs each class equally (use only if test data is balanced) | | | | | | |
| Summary | | | |  |  | | --- | --- | | Imbalanced actual classes: 2,3,6,7 | 1. Accuracy. 2. Precision. 3. Recall. 4. F1 Score  5. ROC AUC. 6. PRC AUC. 7. Micro-Average. 8. Macro-Average | | Multiclass classification: 7,8 | | Cost-dependent classes: 2,3,5 | | | | | | | |
| K-Means Clustering | | | Want to minimise Within-Cluster Sum-of-Squares (WCSS) (i.e. variance) and between-cluster distances  To minimize WCSS: L = . Where S = {S1,..., Sk} is the set of clusters (k = total nm of clusters),  = centroid point of , = squared Euclidean dist from **x** to | | | | | | |
| K-means algo | | | Initialize cluster centroids -> while not converged {assign each pt to nearest cluster centroid, update cluster centroid} | | | | | | |
| To choose k | | | Use domain knowledge | | | | | Note k w diminishing return ("Elbow" method for WCSS) | |
| k-Means vs kNN | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Purpose | k is | Distance metric | Measures dist btw | Training | | k-means | Group neighbors | clusters | Only squared Euclidean (to match var) | Training **x** & centroids | Yes | | kNN | Label based on neighbors | neighbors | Any distance metric | Test **x** & training neighbors | No | | | | | | | | | |