|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ML Pipeline | | | | 1. Data Collection. 2. Data extraction (Feature Engineering). 3. Data understanding (visualisation). 4. Data preprocessing. 5. Model choice/design. 6. Model training. 7. Model validation. 8. Model deployment | | | | |
| AI vs ML | | | | AI -> Intelligent Agents (Addresses all kinds of problems related to human actions/thinking)  Learning Agent Intelligent Agent (i.e. ML AI)  Includes a learning component. ML is used to build learning agents, which are a subset of general AI agents | | | | |
| Supervised Learning | | | | A learning problem is defined by its data and objectives  Data D (matrix). Each of the rows describes one instance (feature vector). Each of the cols = 1 variable = x (indep variable). Last col = label for each instance = y (dependent variable)  If y is cts -> regression learning problem. If y is discrete -> classification learning problem  Model (hypothesis, e.g. function h: **x** -> y)0  Given input data, D, assume each row is drawn from a pop P (often assume D is iid)  Assume there is an unknown target fn, f: **x** -> y (i.e. ground truth). Use a supervised learning algo, A, to determine h (s.t. h ≈ f)  Objective is to find h that is accurate (i.e. accurately predict a label for any row in D) | | | | |
| Learning algo | | 2 components of the learning algo: 1) Hypothesis set/Representation for h, H = {h1, ..., h|H|} (can incur representation bias)  Diagram  Description automatically generated 2) Learning algo A to select h H s.t. h ≈ f w high accuracy (can incur procedural bias)  Both make up the learning model. We choose H, and learning algo A then select h  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 - is the only way to make learning feasible  Each learning algo A has a diff take on its inductive bias, which forms of h it can represent, H and how it selects h (i.e. inductive bias = representation + procedural) | | | | | | |
| kNN | 1. Compute distances. 2. Pick k nearest instances. 3. Aggregate labels from knn and classify new instance  Need to choose: 1) distance metric, 2) k (how many neighbours), 3) Aggregation mtd (majority vote)  These are algo design choices that will change what knn constructs as h to make predictions, aka hyperparameters  Note knn is a lazy learner (i.e. does not require training). D is aka training set. Some part of D may be reserved for validation set | | | | | | | |
| Chart, box and whisker chart  Description automatically generatedVoronoi diagram (representation of h under kNN)  Edges in voronoi diag correspond to perpendicular bisectors (in 1-NN) | | | | | | | |
| Distance Metrics: Euclidean distance/L2(**X**i, **X**j) =  Any distance metric (non-negative, symmetric, obeys triangle inequality) would work (e.g. Manhattan distance/L1)  Possible issues: Discrete data: Non-ordinal -> one-hot encoding (e.g. x {R,G,B}, xi = R, then convert x.Ri = 1, x.Gi = 0, x.Bi = 0)  Discrete data: Ordinal -> assume standard interval ranges or use one-hot encoding  Note that when using 1 hot encoding, w now have multiple variables representing 1 original variable, they must be weighted accordingly  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 (e.g. use weight w = 1/d2, if d(**X**i, **X**j) = 0, use label of reference instance **X**j, since same as **X**i) | | | | | | | With weighted vote, all-NN is not just a mode-classifier (i.e. classifier that always pick the mode label) |
| Decision Tree | | Multiple if else blocks. Most node test exactly one component of **x** (a feature, e.g. x0)  A branch in the tree corresponds to a test result. Each leaf node assigns a class/real value (depending on classification/regression)  Each node filters data further. Ideally at leaf node, we want the filtered instance to have 1 label or a high frequency in 1 label | | | | | | |
| DT Representations (axis prependicular to decision boundaries)    Chart  Description automatically generated | | | | function DTL(examples, attributes, default) returns a decision tree  if examples is empty, return default  else if all examples have same classification, return classification  else if attributes is empty, return MODE(examples)  else  best <- ChooseAttribute(attributes, examples)  tree <- a new decision tree root test best  for each value vi of best do  examplesi <- {elements of examples w best = vi}  subtree <- DTL(examplesi, attributes - best, MODE(examples))  add a brance to tree w label vi, and subtree subtree  return tree | | |
| Need to choose: 1) how to pick feature to split on. 2) How to discretize cts feature. 3) Where to prune trees | | | | | | |
| 1) Ideally, a good feature splits the training set into subsets that are all positive or all negative (as we want a higher purity)  To implement ChooseAttribute, we use Entropy to measure purity, H(X) = , where K = num of labels in X, Pi = prob of class i  Diagram  Description automatically generatedCan imagine entropy as unpredictability, when prob = 0.5, unpredictability is max at 1 (for 2-class entropy)  A chosen feature Vi divides the dataset, S into subsets S1, ..., SC according to the Ci distinct values for Vi  The entropy then reduces to the entropy of the subsets S1, ..., SC: remainder(S, Vi) =  Information Gain = reduction in entropy from knowing value of Vi, IG(S, Vi) = H(S) - remainder(S, Vi)  Choose Vi that gives the largest IG. (entropy and IG have unit bits) | | | | | | |
| 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 leaves get too small (require leaves to have at least i training data)  b) Don't let tree grow too deep | | | | | c) Use part of the training data to check performance (aka validation) -> remove node and replace by MODE of labels y if pruned tree performs better than original | |
| Interpretable: Intuitive for data exploration  Efficient: both in training (greedy search) and testing  Discards irrelevant feature through use of IG | | | Instability: susceptible to small fluctuations (high variance)  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))  (e.g. squared error ((**X**) - f(**X**))2, or binary error [(**X**) ≠ f(**X**)], where [] is Iverson bracket, [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**)] (square bracket here refers to expectation) | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Curse of Dimensionality | | Sparsity problem: maintaining density of samples depends on exponential growth of the data (i.e. need more data as num of dimensions increase for performance of model to not degrade)  - This is seen in as we generate more points, standard deviation grows slowly  In high dimensional space, most points are nearly the same distance away -> learners that depend on distance break down in high dim (since cannot accurately distinguish points)  Might need to choose another appropriate distance metric AND use feature engineering | | | | |
| 3 Learning problems | (linear signal) | | | | | |
| 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 =    . Set = **0**. Then . Hence **.** Note = **a**, and = 2A**x**  = pseudo inverse of **X**. But is expensive to compute for large data. Matrix inverse = O(n3) operation  Summary: construct matrix **X** and vector **y** from dataset. Compute pseudo inverse. Then | | | | | |
| 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  For a fixed step size n: (1) = (0) + nv, where v is the dirn vector we choose (i.e. one that minimize Ltrain)  Ltrain = Ltrain((t + 1)) - Ltrain((t)) = Ltrain((t) + nv) - Ltrain((t)) = Ltrain((t)) + Ltrain((t)) \* nv + O(n2) - Ltrain((t)) = n \* Ltrain((t)) \* v + O(n2) ≥ -n. We want v to be a unit vector, so take v =  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). Soft threshold: uncertainty  (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)  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() = | | | | | Cost fn for logistic uses iterative soln  Cost fn for linear uses closed-form soln, but iteration also possible |
| 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) =  Maximizing likelihood (higher prob = more accurate) Minimizing cross entropy:    = cross entropy error/logistic loss | | | | | |
| 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 | Variation of GD that considers only error on 1 data point  Pick one (**x**(\*), y(\*)) at a time (change pt every time)  Apply GD to l((**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 | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Linear regression not suitable for classification as it minimizes squared error (i.e. best fit line is sensitive to outliers) | |
| Noisy target = 2 instance have same features but diff outcomes | Noisy feature = data reported is feature col is wrong |
| Support Vector Machine (SVM) | | SVM creates better linear separation (by maximizing margin). Also inherently handles noisy data  Hyperplane eqn: h(**x**) = + b  +ve pts must line above hyperplane; -ve pts below  dictates orientation of plane; b dictates offset (bias)  Define dist d to optimal plane as "1" (unit distance)  This sets up constrained quadratic optimization problem that identifies the unique h(**x**)  Note: only a subset of the dataset determines our unique h(**x**)  The subset are called support vectors (i.e. points in green circle), 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  So need to transform data. 1. Original Data **x**(j) X -- -> 2. Transform data **z**(j) = (**x**(j)) Z -->  3. Separate data in Z space, (**z**) = sign() -- -> 4. Classify in X space,  Note **x** = (x1, ..., xn) -- -> **z** = (z0, ..., zn). So, **x**(1), ..., **x**(m) -- -> **z**(1), ..., **z**(m). Note: y labels and = (, ..., ) 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, we say that we have "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, by definition, there are no misclassification (i.e. only concern with width of margin)  In diagram, if pt is > 1: no penalty since correctly classified. If pt at 0 (i.e. on margin line), penalty = C. As pt becomes more misclassified (gg towards left on graph), 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 -> equivalent to hard margin SVM | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Perceptron | Chart, scatter chart  Description automatically generatedLinear 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). 2. For each instance i w features **x**(i): Classify  3. Select one misclassified instance: Update weights: **w** **w** + **w**  4. Iterate steps 2 to 3 until: convergence (classification error < threshold) or max num of iterations reached | | | |
| Perceptron Weight Update (i.e **w**). **w** **w** + **x** (new = old + learning rate \* learning error \* **x**)  Cosine similarity: s = cos() = = . (If s = 1 -> correct classification) | | | |
| A picture containing text, clock  Description automatically generatedFor misclassification: y = +1, = sgn(**w**T**x**) = -1  = -1 **w**T**x** ≤ 0 ≥ 90  But we want: = +1 **w**T**x** > 0 < 90  i.e. **w** need to point in a more similar dir as **x**  i.e. **w**' = **w** + **x** | A picture containing text, clock  Description automatically generatedFor misclassification: y = -1, = sgn(**w**T**x**) = +1  = +1 **w**T**x** > 0 < 90  But we want: = -1 **w**T**x** ≤ 0 > 90  i.e. **w** need to point in a less similar dir as **x**  i.e. **w**' = **w** - **x** | | |
| Combining both: **w** **w** + **x**, i.e. **w** = **x** | | | |
| Activation Fns | Diagram, schematic  Description automatically generatedGraphical user interface  Description automatically generated with medium confidenceDifferentiable Activation fn  .      If g (activation fn) is diffferentiable, we can use gradient descent to find min of error faster | | | |
| Gradient Descent | Iteratively find wi with min error  Gradient descent search for best param/weight values in weight space  Learning rate too small: slow; too large: overshoot  Note h(x) = h(g(f(x))), then h'(x) =  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 Cross Entropy |  |  | Softmax |  |  |  |  |   E.g. using MSE + Sigmoid + wTx: | | | |
| Neural Networks | Diagram  Description automatically generatedNN = Multi-layer perceptron (diagram shows 2-layer MLP)  E.g. Model weights for = |x-1| (non-linear fn). = ReLU  , ,  ,    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 | Diagram  Description automatically generatedFor single output NN: g =  Note  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 | Vector: **x** = . Matrix: **X** = , n = num of features, m = num of instances in dataset  Scalar-by-scaler: y(x) = wx. Scaler-by-vector: y(x) = **w**T**x** = w1x1 + w2x2. Vector-by-vector: **y**(**x**) = w**x** = w = (e.g. )  Matrix-by-matrix: 1) Hadamard product (elem wise): **y**(**X**) =  2) Convolution operator \* (elem wise multiply then sum): **Y**(**X**) = **W\*X** = . (1D = vector, 2D = matrix, ≥ 2D Tensors) | | | | |
|  | (Assumes Cartesian coordinates, linear & orthogonal).  Gradient descent weight update: **W** **W** - . Where (perceptron), and  Note | | | | |
| Gradient Descent with backpropagation | | | Backpropagation efficiently computes gradient by avoiding duplicate calculations, not computing unnecessary intermediate values, computes gradient of each layer  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)  (eventually get recursive gradients of layer weights, derivation in lect 6a) | | |
| Deep Neural Network (DNN) | DNN = many hidden layers (≥ 3). Shallow network (input, 1 hidden, output)  Many layers needed to model target functions of real world tasks which are more complex  More parameters -> need more data  More data -> better performance (avoid curse of dimensionality) | | | | Chart, line chart  Description automatically generated |
| Convolutional NN (CNN) | | Image classification: classify images. Object localization: locate obj in image and draw bounding box around it. Object Detection: object localization + img classification. Image segmentation: object detection + mask image (get the shape of object instead of just a box)  With images, can represent it in 2D matrix, flatten matrix, then feed to NN (but this does not consider relation btw nearby pixels)  CNN does automatic feature learning, and exploit spatial relations with convolutions by using filters | | | |
| Convolutional Filters | | First layer detect simple shapes, like edges -> next layer build up to more complex shape -> top layer would identify object  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  Diagram  Description automatically generated with low confidenceE.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 Convolutions | | Basically for RGB image, separate into RGB channel, apply filter for each channel -> get 3 activation map. Add the 3 activation map element wise to get 1 final 2D matrix | | | |
| Multiple Convolution Kernels | | Diagram  Description automatically generatedWe have 3 channel, but 4 filters  All 4 filters will be applied to the same image.  Each filter has 3 "sub-filters" for the 3 channels  These are then sum to give us an activation map for each filter.  {}  (**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 =  Each kernel stored as separate filters (1D)  All kernels represented as 4D vector | |
| Convolu-tional Layer | Table  Description automatically generatedHyperparameters: 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  - help to train later kernels to detect higher-level features  - reduces dimensionality  - Aggregation methods:  -- Max-pool (most common), average-pool, sum-pool | | | | |
|  | Diagram  Description automatically generated | | | | |
| 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 and reset gate | | | | |
| Generative Adversarial Networks (GANs) | Diagram  Description automatically generated | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bad data | 1. Erroneous data: noise in instance values or labels. Could be due to problem in data collection (human/instrumental error). Some algos are robust against noisy data, others more susceptible  2. Irrelevant data: Refers to features/attributes. Too many irrelevant features impact model performance (apply feature selection)  3. Imbalanced data: Refers to class/labels. Difficult for learning algos to learn effectively. Apply undersampling  4. Missing data: Refers to instance values. Data could be incomplete/wasn't recorded. Need to input missing data  5. Too many features (high dimensionality): Data is too sparse in feature space to adequately learn an accurate model. Apply feature selection | | | | |
| Issues with data | | | 1. Linear separability 2. Curse of Dimensionality 3. Imbalanced Data | | |
| 1. Linear separability | | - Many models assumes data features are linearly separable  - Irrelevant features will be uninformative to train model to discriminate btw prediction labels  - If features are not linearly separable, cannot learn a good linear mode. Need to use more complex models  - Usually occur for "fresh" unprocessed data or unstructured (non-tabular) data, e.g. images, time, text | | | |
| 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 fails  Quantify this: y(\*)((\*) + b) ≥ 1 - , where ≥ 0. Total violation:  Chart, scatter chart  Description automatically generatedEach is the dist that the misclassified point i is from its correct margin. Higher violation = less linearly separable  1bii) 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. 1) Identify basis vectors. 2) Rank basis vectors by importance. 3) Truncate selection of basis vectors  PCA good for supervised and unsupervised learning. LDA better for supervised classification | | | |
| How to mitigate it  1) Find useful features  - feature extraction (collect new features of data)  - feature selection (keep fewer, more useful features) | | 2) Transformation of features::  - features engineering (e.g. 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 | | Sparsity problem: maintaining density of samples depends on exponential growth of data  In high dimensional space, most points are nearly the same distance away  - Data too sparse to inform about true decision boundary (for classification) -> easy to overfit model on sparse training data  - Occurs when we extract more feautres than data instances (n ≥ m). Or in unstructured 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)  Train model M w features -> For each feature, measure decr in M performance -> Eliminate feature w least decr -> Repeat  For RFE in scikit learn library, use elbow method to determine num of features to keep  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 (optimising fewer parameters on fewer features)  - Fewer features -> easier to interpret | | | |
| 3. Imbalanced Data | | Imbalanced data refers to uneven distribution of class/labels. Features can be skewed.  Lead to misleading evaluation metrics, and model overfitting to majority class  Occurs when events rarely occur (cancer) or when data collection is uneven (only positive survey respondants) | | | |
| 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 | Process of transforming raw data to improve accuracy of models. Enables you to:  - capture domain knowledge, express non-linear r/s using linear models, encode non-numeric features to be used as inputs | |
| Tabular data: - use custom equations based on domain knowledge using composite of old features to derive new feature  - counting, aggregation, difference, min, max as new features, then select most important feature | |
| Temporal features: - previous value, aggregate (average, standard deviation, min, max, , trend/linear regression)  - better to use 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, shape, texture  - color distribution (histogram across RGB channels); may not be good 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   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Stemming | truncates words w/o contextual knowledge | remove word endings 'ed', 'ing', 'ly', 'ment'... | easier to implement, runs faster | walking -> walk  having -> have  better -> better | | Lemmatization | groups inflected forms of word | looks up dict to replace word w base form | harder to implement, runs slower | walking -> walk  having -> have  better -> good |   - Extract words: tokenization: split string of text by delimiters  - Word variation:  - 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  Not tested: TF-IDF (if word is unique, more informative to label), N-grams (consider adjacent words), Parts of Speech (POS) & Grammer | |
| Bias & Variance | | Bias: difference btw average prediction and true value (higher bias -> higher diff -> less accurate)  Variance: variability of model prediction (spread of prediction)  High bias -> low variance. Low bias -> High variance. Match model complexity to data resources (quality & qty), not target complexity  Chart, line chart  Description automatically generatedChart, line chart  Description automatically generatedLeft: simpler model. High bias, low variance  Right: complex model. Low bias, high variance |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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: model free = data dictates the model  Higher k lowers dependence of model on a particular data pt, makes model more robust and 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 complexcity 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)] = ED[Ex[(hD(x) - f(x))2]] = 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] | Define (x) = ED[hD(x)]. i.e. (x) ≈  So ED[(hD(x) - f(x))2] = ED[(hD(x) - (x) + (x) - f(x))2]]  = ED[(hD(x) - (x))2 + ((x) - f(x))2 + 2(hD(x) - (x))((x) - f(x))]  = ED[(hD(x) - (x))2] + ((x) - f(x))2 + 2 \* 0 \* ..  = ED[(hD(x) - (x))2] + ((x) - f(x))2  = variance + bias2 | |
| Noise | | Stochastic Noise: fluctuations that we cannot model (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) =  E[Lval(h)] = = Ltest(h). Var[Lval(h)] = = (K points are iid, covariance = 0)  Lval(h) = Ltest(h) ± O() | | |
| Validation | | Given a training dataset D, w n points  Split D into D­train (w n-K pts) and Dval (w K pts). Train on Dtrain to get h-.  Test h- on Dval to get Lval(h-). Use cost Lval(h-) to estimate Ltest(h-)  Retrain model on entire D, to get h. Use h in the end | | O(). Small K: bad estimate of Ltest.  Large K: h- trained on too few examples, lead to bad h-, bad estimate  Rule of thumb: K = n/5 |
| 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): n-1 pts for training, 1 pt for validation  Hypothesis learned will be highly correlated, as most points are identical: 1st hypothesis use pts 2,...,n, 2nd use 1,3,...n  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 | |  |  |  |  | | --- | --- | --- | --- | |  | | Actual Label | | | **P**ositive | **N**egative | | Predicted Label | **P**ositive | TP | FP | | **N**egative | FN | TN |   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 |
| Confusion Matrix |
| Cost-Sensitive Evaluation 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 = = , where R = Recall, P. = Precision  F1 score is more robust (less sensitive to extreme values) |
| Cost-Sensitive Evaluation metrics  2. Vary Prediction Threshold | Usually: . In this case, threshold = 0.5. TPR = . FPR =  By varying prediction threshold, predictions changes and confusion matrix changes  To summarise varying threshold results, use ROC (Receiver Operating Characteristic) & AUC (Area Under 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. Concise metrics enable clearer comparisons  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 | |  |  |  |  | | --- | --- | --- | --- | | Class | 1 | 2 | 3 | | 1 | TP | FP | FP | | 2 | FN | TN | TN | | 3 | FN | TN | TN |  |  |  |  |  | | --- | --- | --- | --- | | Class | 1 | 2 | 3 | | 1 | TN | FN | TN | | 2 | FP | TP | FP | | 3 | TN | FN | TN |  |  |  |  |  | | --- | --- | --- | --- | | Class | 1 | 2 | 3 | | 1 | TN | TN | FN | | 2 | TN | TN | FN | | 3 | FP | FP | TP |     Basically treat 1 class as positive, other classes as negative. For 3 classes, get 3 diff confusion matrix  To combine the 3 into 1: either use micro-average or macro-average |
| |  |  |  | | --- | --- | --- | |  | +ve | -ve | | +ve |  |  | | -ve |  |  |   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 the metric obtained from each confusion matrix. |
| Micro-Average weighs each instance equally. Accounts for imbalanced data.  Macro-average weighs each class equally (use only if test data is balanced) |
| Summary | |  |  | | --- | --- | | Problem | Evaluation Metric | | 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 | for c = 1 to k:  Random() #initialize cluster centroids  while not Converged():  for j = 1 to m:  y(j)  c = #find nearest cluster centroid  x #assign **x**(j) to nearest cluster centroid  for c = 1 to k:  #update cluster centroids  t += 1  return **y** #cluster labels of all datapoints | Converged() =  arg min: = Big Num  for c = 1 to k:  d =  if > d: { = d; y(j) c}  : ∑ = 0  for jc = 1 to mc: ∑ +=  ∑/mc | |
| To choose k | Use domain knowledge | Note k w diminishing return ("Elbow" method for WCSS) | |
| k-Means vs kNN | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | Paradigm | Purpose | k is | Distance metric | Measures dist btw | Model training | | k-means | Supervised | Group neighbors | num of clusters | Only squared Euclidean (to match var) | Training set **x** & cluster centroids | Yes | | kNN | Unsupervised | Label based on neighbors | num of neighbors | Any distance metric | Test set **x** & training set neighbors | No | | | |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |