Intro

14 January 2019 16:23

Machine Learning:

- **Supervised**
  - Learn a mapping from paired input and expected output
  - $f(X_i) = Y_i$
  - Training set $D = \{(X_i, Y_i)\}_i^N$
    - X - **feature space**
    - Y - **label space**
      - Categorical - classification
      - Real-valued scaler - regression
- **Unsupervised**
  - Discover hidden patterns in the data
    - Dimensionality reduction
    - Clustering - categorical
- **Reinforcement**
  - Correct solution not given, just a **reward signal**
  - Feedback
    - Delayed
    - Next data supplied affected by action taken by agent

Main ML problems:

- **Classification**
  - Predict correct label
  - supervised
- **Regression**
  - Approx. an unknown function
- **Clustering**
  - Group data - unsupervised
- **Dimensionality Reduction**
  - unsupervised
- **Density Estimation**
  - Estimate probability of the observed data
- **Policy Search**
  - Reinforcement
  - Which action an agent should take
    - Depending on current state
    - To maximize reward

Instance Based Learning

14 January 2019 16:02

k-Nearest Neighbors:

- Find k-Nearest points *based on distance* from features
  - Manhattan (L1-norm)
    - $d(x_i, x_q) = \sum_g |a_g(x_i) - a_g(x_q)|$
  - Euclidian distance (L2-norm)
    - $d(x_i, x_q) = \sqrt{\sum_g (a_g(x_i) - a_g(x_q))\text{^}2}$
  - Chebyshev (L-infinity norm)
    - $d(x_i, x_q) = \max_g |a_g(x_i) - a_g(x_q)|$
    - Max independent feature distance
- Classify based on these nearest points' classification

- **Choosing k:**
  - **Small k**
    - Good borderline resolution
    - Prone to noise
  - **Large k**
    - Bad borderline resolution
    - Less susceptible to noise
  - Choose with a validation data set
- k-NN is slow if large data set
  - Calculating distances

Distance Weighted k-NN:

- Weighted distance
  - Closer the other point, the higher the weighting
  - Can use
    - *Inverse of distance*
    - *Gaussian distribution*

- Advantage - **more robust to noise**
- Disadvantage - some irrelevant features affect distance - affect weight
  - Remedy - **weigh each feature differently**

- Can use this for regression
  - Weighted average

- **k<n - local method**
- **k=n - global method**
  - Considering all data

As dimensions increases
Avg distances increases
- **Points aren't that close in high-dimensional feature space**

Lazy Learner:

- Stores data - feature + label
- Generalizing beyond data **done after explicit request** made
- +ve
  - Suitable for complex and incomplete problems
  - Large datasets with few attributes
- -ve
  - Large memory requirements
  - Long query time
- e.g. k-NN

Eager Learner:

- Creates an **explicit target function**
- +ve
  - Memory **efficiency**
  - Lower query times
  - Dealing with noise
- -ve
  - **Bad local approximations**
- e.g. ANN / Decision Trees
- Slide 57 - note

Decision Trees

14 January 2019 16:08

DT: approximate discrete classification functions

Algorithm:

1. Search for a **split point (or attribute)** using a **statistical test of each attribute** to determine how well it classifies the training examples when considered alone
2. **Split** your dataset according to your split point (or attribute)
3. *Repeat 1. and 2.* on **each of the created subsets**

Choose feature that reduces total entropy: (max gain, min rmder)

- $G(q) = H(\boldsymbol{dataset}) - \left(\frac{|\boldsymbol{subsetA}|}{|\boldsymbol{dataset}|} H(subsetA) + \cdots\right)$

- **Subset A :** Feature 1 - split into subsets A,B,…
- Probabilities - labels

Ordered Values: (real values)

- for each feature, ***first sort***
- split between two examples in sorted order
  - *that have different classifications*

Symbolic Values:

- search for **most informative feature**
  - reduces total entropy the most
- create as *many branches as there are different values* for this feature

Statistical tests:

- **Information gain**
  - reduction of information entropy
  - $\boldsymbol{H(V)} = -\sum_k \boldsymbol{P(v_k)} \log_2(P(v_k))$
  - $H(V) = -\int_x f(x) \log_2(f(x))$
- Variance reduction
  - regression trees
    - leaf is a linear function
  - target is continuous

Overfitting:

- split into *training* and ***validation*** datasets
- **Pruning:**
  - For all nodes **only connected to *leaves***
  - Check if **accuracy on validation** dataset would **not decrease**
    - if this node became one of the possible leaves
  - Do this **recursively**
    - might create new node just connected to leaves

Evaluation

14 January 2019 16:08

Test Dataset:

- Shuffle
- Then split
- Never use this to tune parameters

Parameter Tuning

- 3 sets
  - Training
    - Diff models
  - Validation
    - Pick best model
  - Test
    - **Final evaluation**
- 60/20/20
  - 50/25/25 if lots of examples

Hold Out:

1. **Train models first**
2. **pick best with validation set**
3. **Re-train**
   - Same parameters
   - training+validation sets
4. **Production**
   - Re-train with ***entire*** dataset

Cross Validation

- Better for smaller datasets
- Divide by k (usually 10)
- Perform **k** folds
  - test set never overlaps between folds

1. **Estimate test set performance**
   - Use k-2 for training
   - 1 validation - optimise parameters
   - 1 test set - estimate performance
     - take avg
   - diff set of optimal parameters in each fold

1. **Only Tuning**
   - know avg accuracy
   - k-1 folds for training
   - 1 validation - optimise parameters

- **Production**
  - Using optimal parameters from 2
  - train on entire dataset

Performance Metrics

28 January 2019 16:36

Classification Rate: (accuracy)

- $\frac{TP+TN}{TP+TN+FP+FN} = \frac{Trace}{Sum}$
- Classification Error = 1 - rate

Recall:

- $\frac{TP}{TP+FN} = \frac{C1\ Predicted\ Correctly}{C1\ Actual}$
- $\Pr(correctly\ classified \mid class\ 1)$

Precision:

- $\frac{TP}{TP+FP} = \frac{C1\ Predicted\ Correctly}{Total\ C1\ Predicitons}$
- $\Pr(positive \mid classified\ as\ positive)$

High Recall, Low Precision

- low FN, high FP
- Class 1 correctly recognised, but many FP

Low Recall, High Precision

- high FN, low FP
- Low Class 1 recognition, but confident when recognised as class 1

Confusion Matrix:

- Treat Class 1 as Positive
-

.	C1 Predicted	C2 Predicted
C1 Actual	TP	FN
C2 Actual	FP	TN

Unweighted Average Recall:

- compute recall for each class
- $UAR = mean(R1, R2 \dots)$

F-measure:

- $F_\alpha = (1+\alpha^2)\frac{Precision * Recall}{\alpha^2 * Precision + Recall}$

measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision

Multiple Class

.	C1P	C2P	C3P	C4P
C1A	TP	FN	FN	FN
C2A	FP	TN	?	?
C3A	FP	?	TN	?
C4A	FP	?	?	TN

- CR and UAR same
- Recall, Precision, F-measure for each class

Imbalanced Test set:

- CR goes down - affected by majority class
- **Precision of other classes goes down**
- UAR can help detect if a class is completely misclassified
- Solutions
  - normalise rows (sum to 1.0)
  - Unsample / Downsample
    - repeat and retrain with diff set
    - mean and s.d. of metrics

Confidence Interval:

- $n \geq 30$
- $error_S(h) \pm Z_N \sqrt{\frac{error_S(h) * (1 - error_S(h))}{n}}$

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Comparing Two Algorithms:

- **Two-sample T-test - diff datasets**
- Paired T-test - diff algo, same dataset
- Get a **"p-value"**
  - $\Pr(null\ hypothesis\ rejected)$
  - null - no performance diff
  - 0 - there is a diff, 1 - no diff
  - **Statistically significant if p<0.05**

ANN Design

14 January 2019 16:08

Neuron:

- $x \in \mathcal{R}^{nx1}$
- $w \in \mathcal{R}^{nx1}$
- $f(z) - acivation\ function$
- $y = f(w^T x)$

Layer:

- Has M neurons
- $x \in \mathcal{R}^{Nx1}$
- $W \in \mathcal{R}^{NxM}$
- $b \in \mathcal{R}^{Mx1}$
- $y = f(W^T x + b)$

Perceptron:

- Binary classification
  - Any linearly separable function
- $h_w(x) = 1\ if\ w.x \geq 0,\ else\ 0$
- $w_i \leftarrow w_i + \alpha(y - h_w(x))x_i$

Feed-Forward Networks:

- Depth = hidden + output layers
- Width = neurons in a layer
- Initialise weights
  - Zeros (useful for bias)
  - Normal
  - **Xavier Glorot**
    - $W \sim U\left[\frac{-\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$
    - $n_j = layer\ input, n_{j+1} = outputs$

Activation Function:

- Sigmoid = $\frac{1}{1+e^{-x}}$
  - Compress between 0 and 1
  - $(0,0.5)$
- Tanh = $\frac{2}{1+e^{-2x}} - 1$
  - Compress between -1 and 1
  - $(0,0)$
- ReLU
  - $f(x) = x\ if\ x > 0, \quad else\ 0$
  - Preserves linear properties while not being linear
- softmax
  - n-dimensional version of sigmoid
  - Compress sum of output vector to 1

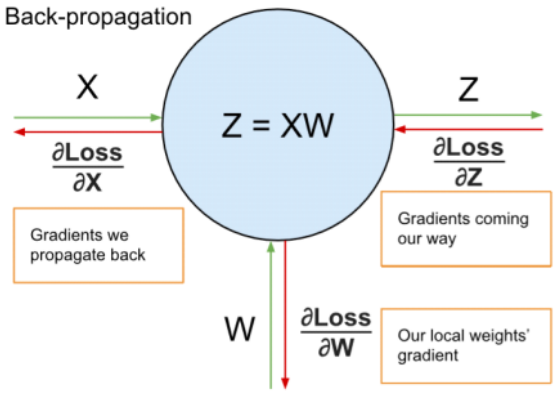
Loss Functions:

Type	Output Layer	Desired Output	Loss
Regression	Linear	value	(mean)-square-error
Binary	Sigmoid	0 or 1	Binary cross entropy
Multi-class	Softmax	One-hot	Categorical x-entropy
Multi-label	Sigmoid	0s and 1s	Binary x-entropy

ANN Training

14 January 2019 16:08

- $Z = XW + b$



$$\frac{\partial Loss}{\partial X} = \frac{\partial Loss}{\partial Z} W^T = \frac{\partial Loss}{\partial Z} \circ f'(X) \;\; (Activation\ Fn)$$

$$\frac{\partial Loss}{\partial W} = X^T \frac{\partial Loss}{\partial Z}$$

$$\frac{\partial Loss}{\partial b} = \mathbf{1}^T \frac{\partial Loss}{\partial Z}$$

Gradient Descent:

- $W \leftarrow W - \alpha \frac{\partial L}{\partial W}$

1. forward pass
2. compute derivate of loss w.r.t network outputs
3. back-propagate (all layers)
4. **THEN** update weights

^DERIVATIONS^

09 March 2019 20:53

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial A}\frac{\partial A}{\partial z}\frac{\partial z}{\partial W}$$

- Work out each term by differentiating

$\frac{\delta L}{\delta A}$:

- Sub A into Loss function the differentiate

$\frac{\delta A}{\delta z}$:

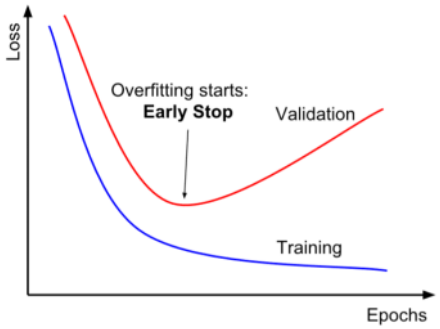
- Sigmoid: $= \sigma(z) = \sigma(z)\big(1 - \sigma(z)\big)$
- Tanh: =

$$\frac{\delta z}{\delta W} = X$$

ANN Evaluating

19 February 2019 13:31

Early Stopping:



Regularisation:

- limit weight max value
- L2 regularisation
- $$J(\theta) = Loss(Y, A) + \lambda \sum_{w} w^2$$
- $$w \leftarrow w - \alpha(\frac{\partial Loss}{\partial w} + 2\lambda w)$$
  - larger weight - larger loss
  - large weights shrink faster
  - encourage combination of inputs
- L1 regularisation
  - also **decays** weight to 0
  - more sparse (more 0s)
  - *feature selection*

$$J(\theta) = Loss(Y, A) + \lambda \sum_{w} |w|$$

$$w \leftarrow w - \alpha(\frac{\partial Loss}{\partial w} + \lambda \text{sign}(w))$$

- dropout - randomly set a neurons output to 0
  - applied at training time
  - reduces inter-dependency between neurons across layers

Data pre-processing:

- data augmentation
  - add noise
  - flip, rotate etc
- data normalisation
  - feature scaling
    - fixed scaling
  - z-normalisation
    - Normal distributed data
  - updating weights which uses this gradient relies on the magnitude of the input

Unsupervised Learning & Density Estimation

14 January 2019 16:08

Density Estimation - anomaly / novelty detection

k-means:

1. **Initialisation**
   a. select the of cluster **k**
   b. randomly place the **k** centroids
2. **Assignment**
   a. each data point is assigned to the closest centroid
3. **Update**
   a. calculate **mean** of all associated points for each centroid
   b. set the centroids to this mean
4. **Repeat**
   a. Repeat 2-3 until convergence

Selection of k:

- **elbow method**
  - $score = \frac{1}{K}\sum_{k=1}^{K} meanDist_k$
  - select 'elbow' corner of K-score plot
- cross-validation
  - run k-means on N-1 folds
  - compute score on remaining fold

Pros - simple, efficient $O(tkn) \sim O(n)$, popular

Cons:

- mean must be defined (k-modes exists)
- reliant on random init - finds local min
- sensitive to outliers
- not suitable for discovering non hyper-ellipsoid clusters

$likelihood = p(\mathcal{X}|\theta)$

$= \prod_{n=1}^{N} p(\boldsymbol{x_n}|\boldsymbol{\theta})$ $(i.i.d\ data)$

We use the negative log-likelihood - (minimise)

$$\mathcal{L} = -\log p(\mathcal{X}|\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(\boldsymbol{x_n}|\boldsymbol{\theta})$$

Mixture Models:

- weigh different models to form a likelihood
- e.g. **Gaussian Mixture Model**
  - $GMM: p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\mu_{\boldsymbol{k}}, \Sigma_{\boldsymbol{k}})$
  - $\pi_k: 0..1, \quad all\ sum\ to\ 1$
  - responsibilities
  - for **kth** model on **nth** data-point
  - $r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$

Expectation Maximisation

- E-step: compute the responsibilities
- M-step: re-estimate the parameters $\boldsymbol{\theta}$

GMM-EM:

1. Initialisation
2. **E-step:**
   a. compute **responsibilities**
      i. for all data points
      ii. and each mixture component
3. M-Step:
   a. Update the **mean** of **each** mixture component
      $$\mu_{\boldsymbol{k}} = \frac{1}{N_k}\sum_{n=1}^{N} \boldsymbol{r_{nk}}\boldsymbol{x_n} \quad , \qquad N_k = \sum_{1}^{N} \boldsymbol{r_{nk}}$$
   b. Update the **covariance** of **each** mixture component
      after the update to the mean
   c. Update the **weight** of **each** mixture
      $$\pi_k = \frac{N_k}{N}$$
4. Repeat E and M steps until convergence

Selecting number of components - Occam's razor

- selecting simplest of all models that fits
- Minimise **Bayesian Information Criterion**
  - $BIC(K) = \mathcal{L}(K) + \frac{P(K)}{2}\log(N)$

k-means vs GMM-EM

09 March 2019 22:26

Similarities:

- select **k**
- converge when changes are small
- Sensitive to initialisation
    - Often, GMM-EM means are initialised from k-means

Differences:

- k-mean
    - **hard clustering**
        - every point belongs to exactly one cluster
    - **minimise** sum of squared **distance**
    - Assumes **spherical clusters** with equal probability of a cluster
- GMM
    - **soft clustering**
        - every point belongs to several clusters
        - responsibilities determine the probability of each data point belonging to a cluster (mixture component)
    - maximise (min) (neg) **log-likelihood**
    - Can be used for **non-spherical** clusters

Genetic Algorithms

14 January 2019 16:08

Main Concept:

1. Population
2. Evaluate
   a. using a **fitness function**
3. Ranking
4. Seeding
   a. Best ones
      i. keep
      ii. breed
      iii. mutate
   b. Discard worst
   c. Back to 1.

Genetic Operators:

- **Selection**
  - after evaluating with fitness function
  - **biased roulette wheel** (CDF)
  - **Tournament** - pit random individuals against each other
  - **Elitism** - pick top (10%) to always stay
- **Cross-over**
  - akin to *breeding*
  - for binary string - basic split points for genotypes
- **Mutation**
  - **explore nearby solutions**
  - flips bits - with probability m = 1/sizeof(genotype)

Used to have:

- Genetic algorithm
  - string of bits
- Genetic programs
  - program represented as a tree
- Evolutionary strategies
  - floats
  - usually don't cross-over

Unification:

- **Evolutionary Algorithms**

$(\mu + \lambda) - Evolutionary\ Srategy$:

1. Randomly gen population of size $(\mu + \lambda)$
2. Evaluate
3. Select the μ best individuals to keep as **parents**
4. Generate λ amount of offspring from the parents
   a. each λ_i can mutate (add some $\mathcal{N}(\mathcal{O}, \sigma)$)
5. Have new population
6. Return to 2.

Finding Sigma:

- Large - quick, bad refine ; Small - slow, local opt, but refined
- **Concept:** Add sigma to the genotype