Information Theory

Getting information based on entropy(unorderliness in a set, higher entropy means more disordered). $I(P(v_1)...P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$ For a training set with p positive examples and n negative examples : $I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$. $I(\frac{1}{2}, \frac{1}{2}) = 1$, $I(1, 0) = 0$. Entropy measures the number of bits to represent data. Higher entropy means that more bits can be used to represent data.

Info Gain

Entropy of children given by : $remainder(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$

Information gain : $IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A)$ Then choose the attribute A with the largest information gain. Information gain is how much entropy we removed. **What if information gain is the same for 2 attributes? How to choose and does order affect?** - Order doesn't affect, choose either.

Performance Measurement

Decision tree is the hypothesis approximation. On training set, deeper tree means more accurate. On test set, deeper tree decreases accuracy. Leads to **overfitting**. Decision tree don't account for temporal factor.

Occam Razor

Prefers short hypothesis. Short hypothesis that fits data is unlikely to be coincidence, but long hypothesis that fits may be(Lucky that long hypothesis fits both test and training).

Pruning

Min-sample, once a node exceeds the min-sample value, fold and take the dominant one. Max-depth: when depth exceeds, fold nodes that exceeds the depth.

Continuous-Valued Attributes

For values that have ranges, we can partition them into smaller sets with fixed ranges instead. Gives them some form of ordering and its branching factor dependent.

Attributes with Many Values

Problem - gain will select attribute with many values like dates. **Solution** - use gain ratio instead, bias against attributes with alot of values. $GainRatio(C, A) = \frac{Gain(C,A)}{SplitInfo(C,A)}$ Attribute A divides training set E into subsets $E1...Ed$ corresponding to d distinct values of A. SplitInfo can be calculated as the negative sum of ratios multiplied by the log of the ratio.

Attributes w differing costs

Problem - How to learn a consistent DT with low expected cost? **Solution** - Replace Gain by : $\frac{Gain^2(C,A)}{Cost(A)}, \frac{2^{Gain(C,A)}-1}{(Cost(A)+1)^w}$ Solution will be biased against high cost. w is the weight of the cost.

Missing Attribute Values

Problem - what if some examples are missing values of attribute? **Solution** - Randomly assign based on current attribute values. Could use most common value.

Logistic Regression

Function : $\frac{1}{1+e^{-\theta t x}}$ Cost function : $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i log h_\theta(x^i) + (1-y^i) log(1-h_\theta(x^i))$. $h_\theta(x)$ is the estimated probability that $y = 1$ on input x. Gradient descent same as linear regression.

Multi-class Classification

Perform binary classification with each class and take every other class as the other set. Then pick the class that maximises the heuristic function.

Overfitting

Resolved by :

- Reduce number of features
- Regularise - keep all features but reduces magnitude of parameters θ_j

LR with Regularisation

Hypothesis: $h_\theta(x): \theta^T x$

Cost Function: $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}\theta_i^2\right]$

$\theta_n := \theta_n - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}).x_n^{(i)} - \frac{\lambda}{m}\theta_n$

$:= \left(1 - \frac{\alpha\lambda}{m}\right)\theta_n - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}).x_n^{(i)}$

slightly less than 1 — "regular" gradient descent

Second term known as **regularisation paramter** - aims to avoid over-fitting by scaling the values of θ using λ.

Regularised Normal Equation

Mapping from linear regression, create identity matrix but the 0-th row and 0-th column value is 0. Because this is θ_0 which is the bias.

$$\theta = (X^T X)^{-1} X^T Y$$

$X^T X$ needs to be invertible

$$\theta = (X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix})^{-1} X^T Y$$

This works even if $X^T X$ is non-invertible if λ>0!

Regularised Logistic Regression

Hypothesis: $h_\theta(x): \frac{1}{1+e^{-\theta^T x}}$

Cost Function: $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) + \frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2$

Gradient Descent: $\theta_n := \theta_n - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}).x_n^{(i)} - \frac{\lambda}{m}\theta_n$

SVM

SVM splits 2 sets of data into 2 groups, finding the best way to split them in the process with the biggest margin. $cost_1$ refers to when the cost is 1. Difference between logistic regression cost function is the difference in $cost_i$. Solved using libraries. Can be seen as $CA + B$, where A is the summation of the costs, and B is the regularisation parameter. $C \approx \frac{1}{\lambda}$. If λ is large, then SVM ignores outliers. Points that affect solution is along margins.

Kernel Tricks

Non-linearly separable - solution using polynomial regression. Kernel is basically increasing the number of dimensions to find a way to separate the points. Important to perform feature scaling before using kernels.

Gaussian Kernel

$e^{-\frac{|x-li|^2}{2\sigma^2}}$ Based on the distance of the point to a landmark, decide whether to include it in the boundary. Maximum value is 1.

SVM with Kernels

From training data, choose landmark for each data point. Generate hypothesis.

- C : if large, more variance, less bias.
- σ^2 - if large, features vary more smoothly (more spread out), higher bias, less variance.

Training Logistic Regression

- minimise the cost function
- compute test set error
- misclassification error - some form of bias towards test set.

How to choose a model? - train every model, and pick the model with lowest test set error.

Measuring Goodness

Split test set into cross-validation and test. 3 ways to measure goodness :

1. Training Error
$$J_{train}(\theta) = \frac{1}{2m_{train}}\sum_{i=1}^{m_{train}}(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)})^2$$
2. Cross Validation Error
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$
3. Test Error
$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

Compute $J_{cv}(\theta)$, and pick the lowest. Then use test error to estimate performance on unseen samples. Validation set makes resulting hypothesis bias towards it.

Bias and Variance

Underfitting - high bias. Overfitting - high variance. As the degree of polynomial increase, error of training set decreases because we can better fit the data. With low degree, fit is bad, more error. With high degree, fit is good. But for a sample that is not seen before, the error will be high. On underfit - J_{train} high and $J_{train} \approx J_{cv}(\theta)$. On overfit J_{train} low and $J_{train} \ll J_{cv}(\theta)$. Adding more data points won't help high bias but will help high variance.

Neural Networks

Aim to linearly separate 2 groups of points. Independent on number of features, can be used on data with lots of features.

Perceptron

Inputs with their respective weights, sum them up, apply a function to get an output. g refers to the non-linear activation function. If g is linear, become linear regression. Possible to create new perception from weights of old perceptrons if labels were swapped. A xor B = (A or B) and (A and B)'

Perceptron Update Algo

Select initial weights, apply onto features, classify, then update. Repeat till converge.$w = w + \eta(y - \hat{y})x$. Since perceptron aims to find best fit line, aims to reduce the angle size between wrongly classified points and their correct classfications. y is the correct classifier of the point and $\hat{y}$ is the current classifier of the point.

Perceptron	Linear SVM
• not robust: Can select any model to linear, not deterministic • Cannot converge on non-linearly separable data	• Perceptron of "optima stability" • Maximizes margin • Soft margin: can learn from non-linearity separable data

Gradient Descent

To find the appropriate weights, perform gradient descent, ie find the lowest MSE for $y - \hat{y}$. As a result of some differentiation using sigmoids and chain rule, we get $w = w + \eta(\hat{y} - y)\hat{y}(1 - \hat{y})x$, where x is the feature value. $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Multilayer Perceptron

Layer count starts from left most. Hidden layer count does not include input and output layer.

Back Propagation

Used to determine the weights of the hidden layers. Have to go forward first, using the weights of the previous layer, then use back propogation to get the accurate weights of the entire network. Hadamat product - element wise product.

$$\frac{\partial \varepsilon}{\partial w_2} = \frac{\partial \varepsilon}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_2} \text{ (Chain Rule)}$$

Similarly, $$\frac{\partial \varepsilon}{\partial w_1} = \frac{\partial \varepsilon}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_1} \text{ (Exactly like } w_2\text{)}$$

$$= \frac{\partial \varepsilon}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a_1}\frac{\partial a_1}{\partial w_1} \text{ (Chain Rule)}$$

Matrix Form

$\hat{y}'(W^{[l]}) = a^{[l-1]}\ \delta^{[l]}$

$\delta^{[l]} = g'^{[l]}(f^{[l]})\ W^{[l+1]}\ \delta^{[l+1]}$

g is the activation function used. f is the product of weights and the inputs. a denotes activations.

Properties

Efficiently computes gradient by :

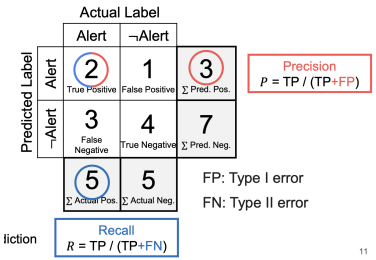
- avoiding duplicate calculations
- not computing unnecessary intermediate values
- compute gradient of each layer

Basically uses something like dynamic programming. Base case : start at last layer, compute the loss in weight. Subsequent layers uses calculated value of the next layer multiplied by the loss in weight of the current layer. Problems :

- Activation must be differentiable, otherwise weights will be inaccurate
- Issue of vanishing/exploding gradient.

Confusion Matrix

- Precision : correctly convict a guilty person (minimise if false positive is costly)
- Recall : percentage of correct convictions (minimise if false negative is costly)
- Accuracy : (TP + TN)/everything
- Correctness : when classfication is same as actual.



11

F1 score

$\frac{2*P*R}{P+R}$ More robust, less sensitive to extreme values. Consider that numerators of P and R are the same, compares denominators only.

Receiver Operator Characteristic Curve

Values below threshold lines are supposed negative. Area under graph give a measure of accuracy, closer to 1 is better. Can be derived by varying threshold and calculating TPR and FPR. Plot TPR again FPR. $FPR = \frac{FP}{FP+TN}$, $TPR = \frac{TP}{TP+FN}$

Spatial Exploitation

Convolution sends groups of pixels, reducing number of inputs. With kernel, element-wise multiplication then sum everything up. Different types of kernels has different effects on the image.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Padding & Stride

Pixels are lost along edges, use padding. Instead of going sequentially, use something like python 'step' works for both horizontal and vertical, this is stride.

Pooling Layer

Feature maps can get very big, downsample feature maps. Helps to train later kernels to detect higher-level features. Reduces dimensionality. Groups pixels in groups, essentially reducing the size of the image by size of group. This means more blurry and pixelated.

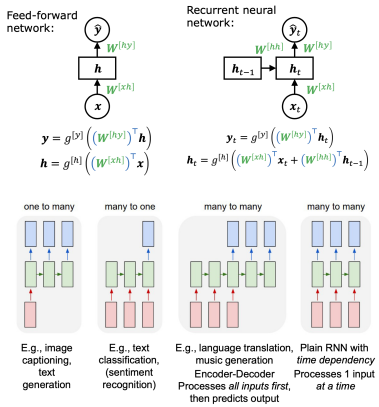
Softmax

Detects maximum of a set of values, not as an absolute value. Bigger values are closer to 1.

$$P(y = j|\theta^{(i)}) = \frac{e^{\theta^{(i)}}}{\sum_{j=0}^{k} e^{\theta^{(j)}}}$$

Recurrent Neural Network

Exploit some form of temporal locality. Predice output based on input and the previous function. This is known as **Exponentially-Weighted Moving Average (EWMA)**. λ between 0 and 1.



Overfitting

Use regularisation to resolve.

- Drop-out : some activations become 0 randomly. More training will actually improve and not plateau
- Early stopping : Since loss from test will increase after certain point, we can stop when we reach minimum loss on **test set**.

Vanishing/Exploding Gradients

Sigmoid functions are less than 1. As more values are multiplied together, approach 0 and 'vanishes'. If gradients keep getting larger, will diverge and 'explode'. **Solutions:**

- Proper weights initialisation - random weight initialisation
- Don't use functions that have limits - choose something like Relu
- Batch normalisation - something like feature scaling (normalise inputs of layers)
- Gradient Clipping - cap the loss so it doesn't get too big

Unsupervised Learning

Little to no idea what results should look like. No feedback on predicted results

K-Means

- Pick k random points for K clusters (centroids)
- Group points closest to centroids as the same cluster
- Find the mean squared distance between the point and its own centroid - loss
- Repeat till converge

If we somehow picked a centroid that lead to lowest mean squared distance, may get stuck (local optimum). Solution - Try again, then pick the lowest. **K-medoids** - pick points that are furthest from all other points. Running K-means with probability of randomly walking.

Choosing Number of Clusters

More clusters, loss decreases. Mean squared distance between point and centroid decreases as number of cluster increases.

- Elbow method - find last point of inflexion on loss-k graph. Use that value as k
- Business need - depending on context, the value of k is fixed.

Hierachical Clustering

Instead of picking k, let some algo pick it for us instead. Idea :

- Every data point is a cluster
- Find clusters that are closest to each other and group them into one new cluster
- To find closest - use feature scaling or mean normalisation
- Eventually end up with 1 big cluster

Everytime we reduce the number of clusters by 1, eventually converge to 1. Then to choose, look at dendrogram. Draw a threshold horizontal line. Count the number of big clusters below the line. Result may still show overlap because points are clustered in higher dimensions. Problems :

- computational complexity
- long training times
- higher risk of inaccuracy
- human intervention on output
- lack of transparency on how data were clustered

Singular Value Decomposition

$X = U \sum V^T$. $\sum$ is a diagonal matrix containing σ, which increases down the matrix. Remaining matrices are singular vectors. Unitary means that it matrix multiplication of itself with its transpose gives the identity matrix. Singular values in this case refers to the weights.

Principal Component Analysis

Compute the covariance matrix and pass it throuugh SVD. Select first k vectors(columns). Exploit the fact that the result of selecting the vectors produces a matrix of shape n x k. Transposing it and multiplying it with a data point (n x 1) will result in k x 1, which is in a lower dimension than the data point input.

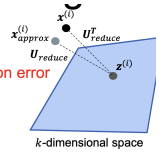
Intuition

$$\min_U \frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} - x_{approx}^{(i)} \right\|^2$$

average squared projection error

$$\frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} \right\|^2$$

total variation in data



Choose **minimum** k such that average squared error divided by total variation is smaller than 1%

Applying

Apply to training set. Once we get the reduced matrix, apply to test or cross validation sets.**PCA is not meant to resolve overfitting!**

Reconstruction

To get the approximate original back, multiply the reduced matrix with the lower dimension matrix. Result is approximately the same since we did not preserve all the vectors in the lower dimension. It is possible to identify useless features based on the size of σ. The bigger the values of σ chosen, the easier it will be to recover the **original** matrix.