Contents

1 Notation

Superscript (i) indicates relation to training example i.
Superscript $[i]$ indicates relation to neural network layer i.
Matrices are indexed by M_{rc}, r = row, c = col.

2 Regression

Superscript (i) indicates relation to training example i.

2.1 Linear regression

2.1.1 Squared error cost function

Measures how well line fits training data
m = num of training examples
$x^{(i)}$ is training example x value i
$y^{(i)}$ is training example y value i
$\hat{y}^{(i)} = wx^{(i)} + b$

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$\frac{1}{m}$ finds average error for larger data sets, $\frac{1}{2m}$ makes later calculations neater

2.1.2 Gradient descent

Find w, b for minimum of cost function $J(w,b)$

1. Start with some w, b (commonly $0, 0$)
2. Look around starting point and find direction that will move the point furthest downwards for a small step size

α = learning rate
Must simultaneously update w and b

$$w := w - \alpha\frac{\partial}{\partial w}J(w,b)$$
$$b := b - \alpha\frac{\partial}{\partial b}J(w,b)$$
$$\frac{\partial}{\partial w}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x^{(i)}$$
$$\frac{\partial}{\partial b}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})$$

2.2 Multiple linear regression

n_f = number of features
m = number of data points
$\vec{w}$ = vector of weights (length n_f)
$\vec{x}^{(i)}$ = vector of x values for training example i, length n_f
Sum of predictions of all features is the prediction of multiple linear reg

$$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$$

Gradient descent

$$\vec{w}_j := \vec{w}_j - \alpha\frac{\partial}{\partial \vec{w}_j}J(\vec{w},b)$$
$$b := b - \alpha\frac{\partial}{\partial b}J(\vec{w},b)$$

Cost function and its partial derivatives

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$
$$\frac{\partial}{\partial \vec{w}_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\vec{x}_j^{(i)}$$
$$\frac{\partial}{\partial b}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

2.3 Logistic regression

Sigmoid function

$$g(z) = \frac{1}{1+e^{-z}}$$
$$z = f_{\vec{w},b}(\vec{x})$$
$$\hat{y}^{(i)} = g(f_{\vec{w},b}(\vec{x}^{(i)}))$$

$\hat{y}^{(i)}$ can be interpreted as the "probability" that class is 1, $0 \leq \hat{y}^{(i)} \leq 1$
ex. $\hat{y}^{(i)} = 0.7$ means there is a 70% chance y is 1
Logistic regression requires a new cost function because $f_{\vec{w},b}(\vec{x})$ for logistic regression is non-convex, trapping gradient descend in local minima.
Cost function

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$
$$L(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1-\hat{y}^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$

Simplified form
$$L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)}\log(\hat{y}^{(i)}) - (1-y^{(i)})\log(1-\hat{y}^{(i)})$$

The loss function will decrease as $\hat{y}_i$ approaches $\vec{y}_i$ on a graph of L vs f.
$\frac{\partial J(\vec{w},b)}{\partial \vec{w}_j}$ and $\frac{\partial J(\vec{w},b)}{\partial b}$ are the same as in linear regression, just the definition of f has changed.

2.4 Softmax regression

Generalization of logistic regression, y can have more than two possible values.
The most probable value of y is the value that when given to L yields the largest loss.
Calculate $\vec{z}_i$ with $\vec{x}$ only consisting of data points that have label i. In implementation, set all y values of data points with label equal to i to 1, and 0 for everything else.
n_f = num features
n_y = number of possible y outputs
W is a matrix of dimensions $n_y \times n_f$.
$\vec{b}$, $\vec{z}$, $\vec{a}$ are vectors of length n_y.

$$1 \leq i \leq n_y$$
$$\vec{z}_i = W_i \cdot \vec{x} + \vec{b}_i$$
$$\vec{a}_i = \frac{e^{\vec{z}_i}}{\sum_{k=1}^{n_y} e^{\vec{z}_k}}$$

$$L(\vec{a},y) = \begin{cases} -\log \vec{a}_1 \text{ if } y = 1 \\ -\log \vec{a}_2 \text{ if } y = 2 \\ \vdots \\ -\log \vec{a}_n \text{ if } y = n \end{cases} \tag{1}$$

2.5 Feature scaling: z-score normalization

After z-score normalization, all features will have a mean of 0 and a standard deviation of 1
n_f = num features
$\vec{\mu}_j$ = mean of all values for feature j (length n_f)
$\vec{\sigma}_j$ = standard deviation of feature j (length n_f)

$$\vec{x}_j^{(i)} = \frac{\vec{x}_j^{(i)} - \vec{\mu}_j}{\vec{\sigma}_j}$$
$$\vec{\mu}_j = \frac{1}{m}\sum_{i=1}^{m} \vec{x}_j^{(i)}$$
$$\vec{\sigma}_j^2 = \frac{1}{m}\sum_{i=1}^{m} (\vec{x}_j^{(i)} - \vec{\mu}_j)^2$$

2.6 Over / underfitting

Underfit / high bias: does not fit training set well ($wx + b$ fit onto data points with $x + x^2$ shape)
Overfit / high variance: fits training set extremely well but does not generalize well ($w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$ fit onto training set of shape $x + x^2$ can have zero cost but predicts values outside the training set inaccurately)

Addressing overfitting
- Collect more data
- Select features ("Feature selection")
- Reduce size of parameters ("Regularization")

2.6.1 Regularization

Small values of $w_1, w_2, \cdots, w_n, b$ for simpler model, less likely to overfit
Given n_f features, there is no way to tell which features are important and which features should be penalized, so all features are penalized.

$$J_r(\vec{w}, b) = J(\vec{w}, b) + \frac{\lambda}{2m}\sum_{j=1}^{n_f} \vec{w}_j^2$$

Can include b by adding $\frac{\lambda}{2m}b^2$ to J_r but typically doesn't make a large difference.
The extra term in J_r is called the regularization term.
Effectively, $\lambda \propto \frac{1}{w}$. When trying to minimize cost, either the error term or the regularization term must decrease. The larger the lambda, the more the regularization term should decrease to minimize cost, decreasing w parameters.
Regularized linear regression

$$J_r(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2\right] + \frac{\lambda}{2m}\sum_{j=1}^{n_f} \vec{w}_j^2$$

For gradient descent, only $\frac{\partial J_r}{\partial \vec{w}_j}$ changes (b is not regularized):

$$\frac{\partial J_r}{\partial \vec{w}_j} = \frac{1}{m}\sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\vec{x}_j^{(i)}\right] + \frac{\lambda}{m}\vec{w}_j$$

Regularized logistic regression

$$J_r(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n_f} \vec{w}_j^2$$

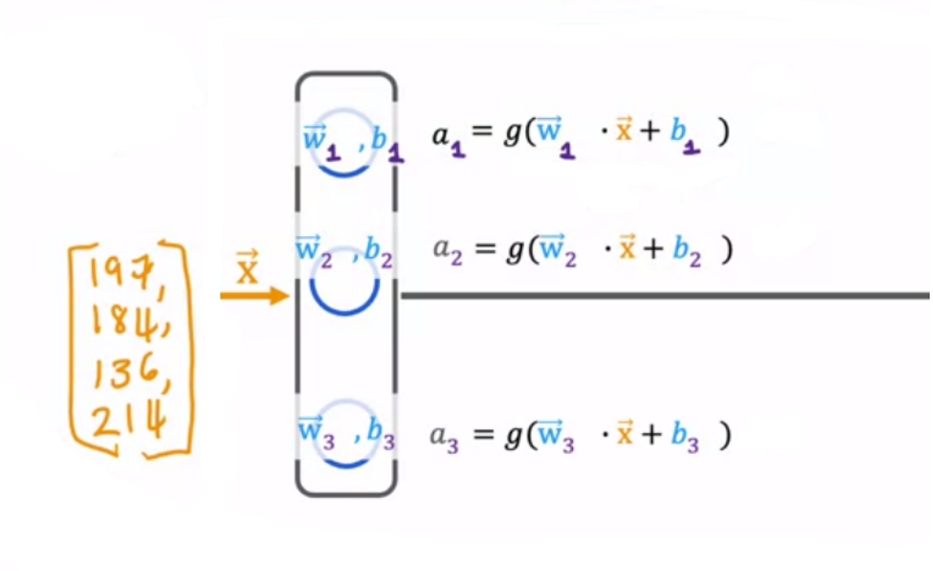
For gradient descent, only $\frac{\partial J_r}{\partial w_j}$ changes (b is not regularized):

$$\frac{\partial J_r}{\partial \vec{w}_j} = \frac{1}{m}\sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\vec{x}_j^{(i)}\right] + \frac{\lambda}{m}\vec{w}_j$$

3 Neural networks

3.1 Intuition

The input layer, or the data, is layer 0. Hidden layers start from layer 1, and the output layer is the last layer. Superscript square brackets are commonly used to refer to some layer; for example, $\vec{a}^{[i]}$ refers to the vectors of activations in layer i.



where g is some activation function (ex. sigmoid). $\vec{a}$ will be the output of this layer, which is a vector of the activations $[a_1, a_2, a_3]$ from the previous layer.

A more compact way of expressing the activations in a layer is $a_j^{[\ell]} = g(\vec{w}_j^{[\ell]} \cdot \vec{a}^{[\ell-1]} + b_j^{[\ell]})$.

3.2 Activation functions

Linear: $g(z) = z$, used for ex. change in stock prices tomorrow
Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$, used for binary classification
ReLU: $g(z) = \max(0, z)$, used for non-negative predictions ex. housing prices
When choosing $g(z)$ for the output layer:

- Sigmoid for binary classification
- Linear for linear regression with both positive / negative results
- ReLU is similar to linear but if output values can only be non-negative

Choosing $g(z)$ for hidden layers:

- ReLU is the most common (faster, doesn't flatten out like sigmoid)
- Linear is typically not used in hidden layers, because they remove complexity from the model's predictions. A neural network with many layers that all use the linear activation function is not any better than a single regression using the activation function of the output layer.

3.3 Training the model

Derivatives can be computed by changing a function's argument by some small ϵ and seeing how the function changes as a reuslt of its argument change. The derivative is then change in function / change in argument.

For example, given cost function $J(w) = w^2$, $\frac{\partial J}{\partial w}\Big|_{w=3}$ can be determined with $\frac{J(w+\epsilon)-J(w)}{\epsilon}$. Values for ϵ can be as small as 0.0001, because ϵ should simulate an infinitely small number.

3.3.1 Forward propagation

$n^{[\ell]}$ = number of neurons in layer ℓ
$n_f = n^{[0]}$ = num features
m = number of examples

$Z^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times m$ for $\ell > 0$ which holds the values to pass to the activation function of layer ℓ. The $n^{[\ell]}$ rows hold one feature of the m data points in each neuron in layer ℓ. Not applicable for input layer.

$A^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times m$ for $\ell > 0$ which holds activation values that come from $g^{[\ell]}(Z^{[\ell]})$.

$g^{[\ell]}$ is the activation function for layer ℓ.

$W^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times n^{[\ell-1]}$ for $\ell > 0$, which holds $\boldsymbol{w}$ values for each neuron in layer ℓ. Not applicable for input layer.

$\boldsymbol{b}^{[\ell]}$ is a list of b values for each neuron in layer ℓ, length $n^{[\ell]}$.

$A^{[0]} = X$ is a matrix of dimensions $n_f \times m$ which holds all input data.

$$Z^{[\ell]} = W^{[\ell]} \cdot A^{[\ell-1]} + \boldsymbol{b}^{[\ell]}$$
$$A^{[\ell]} = g^{[\ell]}(Z^{[\ell]})$$

3.3.2 Back propagation

Including variables from the forward propagation section:
L = last layer
Y is a matrix of dimensions $n^{[L]} \times m$ which holds training data labels.
$\mathrm{d}Z_i^{[\ell]}$ is the ith column of $\mathrm{d}Z$, giving a vector of length $n^{[\ell]}$.
For output layer:

$$\mathrm{d}Z^{[L]} = A^{[L]} - Y$$
$$\mathrm{d}W^{[L]} = \frac{1}{m}\,\mathrm{d}Z^{[L]} A^{[L-1]T}$$
$$\mathrm{d}\boldsymbol{b}^{[L]} = \frac{1}{m}\sum_i \mathrm{d}Z_i^{[L]}$$

For hidden layers:

$$\mathrm{d}Z^{[\ell]} = W^{[\ell+1]T}\,\mathrm{d}Z^{[\ell+1]} * g^{[\ell]\prime}(Z^{[\ell]})$$
$$\mathrm{d}W^{[\ell]} = \frac{1}{m}\,\mathrm{d}Z^{[\ell]} A^{[\ell-1]T}$$
$$\mathrm{d}\boldsymbol{b}^{[\ell]} = \frac{1}{m}\sum_i \mathrm{d}Z_i^{[\ell]}$$

3.4 Improving model

Cross validation: split data into training and test, use test data to determine how well the model generalizes

3.4.1 Fixing high bias/variance

High bias (underfit): J_{train} high, $J_{train} \approx J_{cv}$
High variance (overfit): J_{train} may be low, $J_{cv} \gg J_{train}$
High bias and high variance: J_{train} high, $J_{cv} \gg J_{train}$
How to fix:

1. Get more training examples (fix high variance)
2. Try smaller sets of features (fix high variance)
3. Add more features (fix high bias)
4. Add polynomial features (fix high bias)
5. Decrease λ (fix high bias)
6. Increase λ (fix high variance)

Neural networks and bias/variance
If J_{train} is high, make the network larger
If J_{cv} is high, get more data

3.4.2 Adding data

Data augmentation: add data with distortions (ex. distorted letters in a letter recognition program)

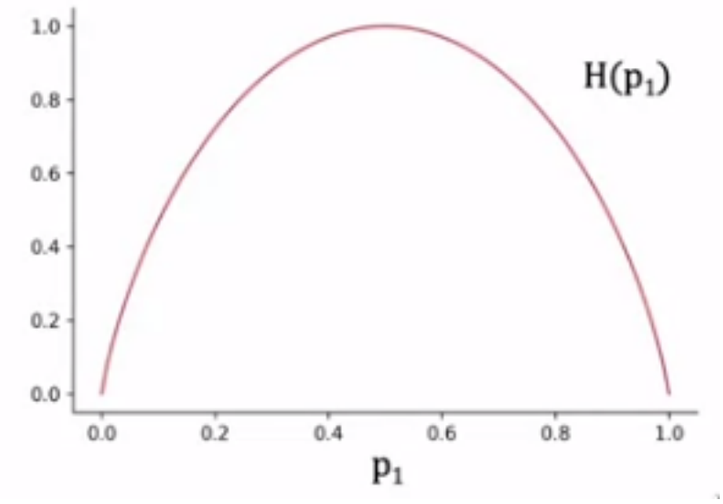
4 Decision trees

4.1 Measuring purity

4.1.1 Entropy as a measure of impurity

p = fraction of examples that are cats

$p = \frac{1}{2}$
Impurity can be measured with the entropy function $H(p)$

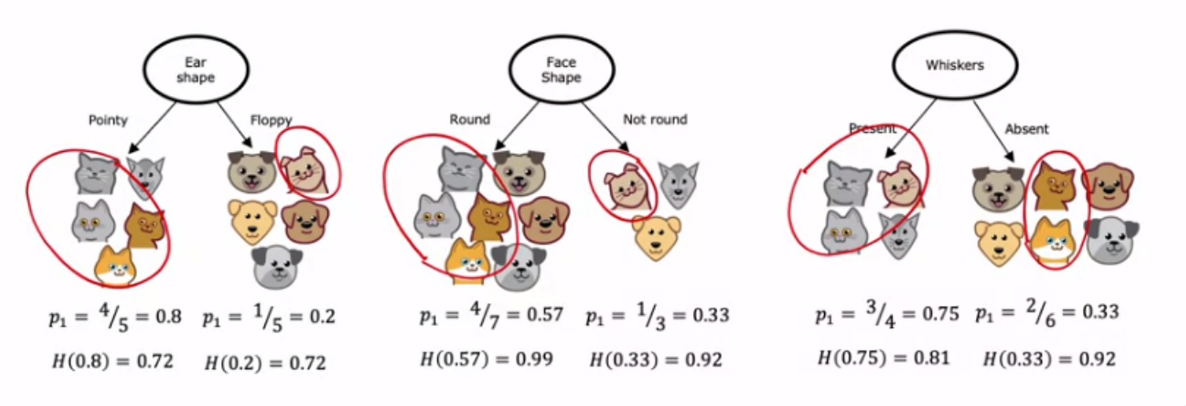


Higher H = less pure, more information gain
Mathematically, H is defined as:

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$$

$0\log(0)$ is defined as 0 for the function H

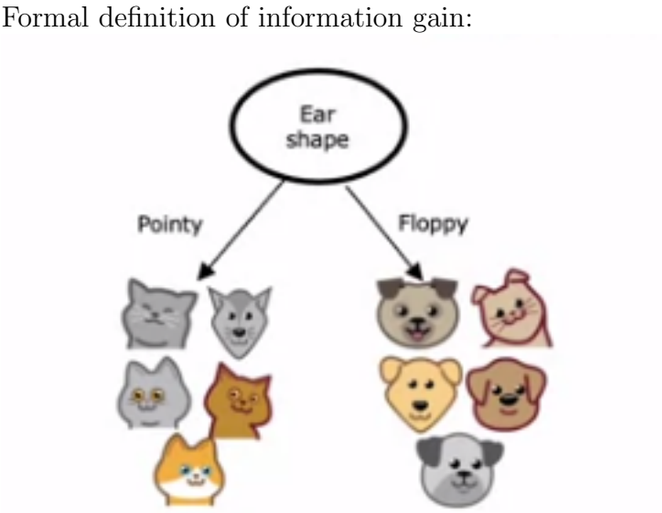
4.2 Choosing a split



To choose which feature to split data on is the best, calculate the weighted average of the entropy on the left and right branches, then choose which split has the highest entropy (least pure, which will give a good split).

Average of ear shape split entropy: $0.5H(0.8) + 0.5H(0.2) = 0.28$
Average of face shape split entropy: $0.7H(0.57) + 0.3H(0.33) = 0.03$
Average of whiskers split entropy: $0.4H(0.75) + 0.6H(0.33) = 0.12$
Ear shape has the largest entropy, so the best choice is to split based on ear shape.

Formal definition of information gain:



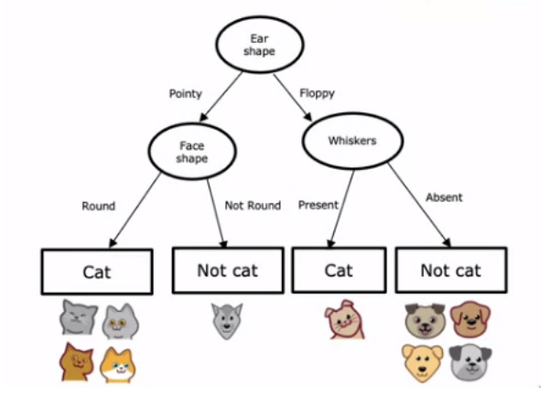
p_{root} = percentage of positive examples (0.5 in this case)
$p_{left} = 4/5$

$p_{right} = 1/5$
$w_{left} = 5/10$
$w_{right} = 5/10$
$H(p_{root}) - (w_{left}H(p_{left}) + w_{right}H(p_{right}))$

4.3 Constructing a decision tree

1. Start with all examples at root node
2. Calculate information gain for all possible features, pick one with highest information gain
3. Split dataset according to selected feature, creating a left and right branch
4. Stop when stopping criteria is met (node is 100% one class, information gain from more splits is less than a threshold, num examples is below a threshold)

Final decision tree



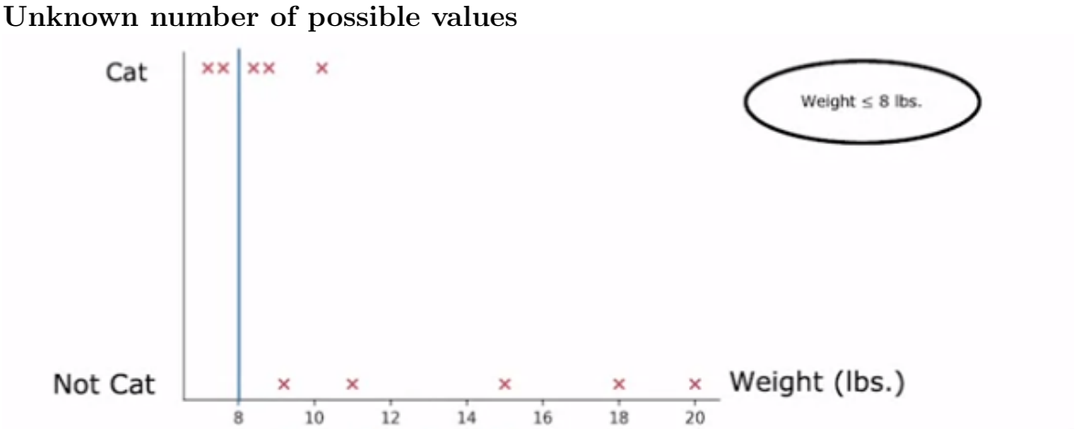
4.4 Features with multiple possible values

Known number of possible values

If a categorical feature can take on k values, create k binary features

ex. Create a true/false feature for pointy ears, floppy ears, and oval ears instead of a single feature "ear shape" which takes on three possible values.

Unknown number of possible values



Split on weight ≤ 8 lbs

$p_{root} = 0.5$
$p_{left} = 1$
$p_{right} = 3/8$
$w_{left} = 2/10$
$w_{right} = 8/10$
$H(0.5) - (\frac{2}{10}H(1) + \frac{8}{10}H(\frac{3}{8}))$

To find most optimal information gain (maximize H), make splits between every pair of adjacent data points and choose the one with the highest information gain.

4.5 Tree ensembles

Training multiple decision trees will lead to more accurate predictions since a single decision tree is sensitive to small changes in data.

4.5.1 Sampling with replacement

Take original training set of size m and randomly select from the original training set to create a new training set of size m. Repeated data is expected.

This will create new datasets that are similar to the original dataset, but are slightly different which will create unique decision trees.

4.5.2 Random forest algorithm

From $b = 1$ to B: sampling with replacement to create new dataset, train decision tree on new dataset.

B is commonly around 100. Setting B too large doesn't hurt performance but gives diminishing returns as it increases.

Randomizing feature choice is another way to create more unique decision trees: Given n features, give each decision tree a subset of all features of size k.

A good value for k is $k = \sqrt{n}$.

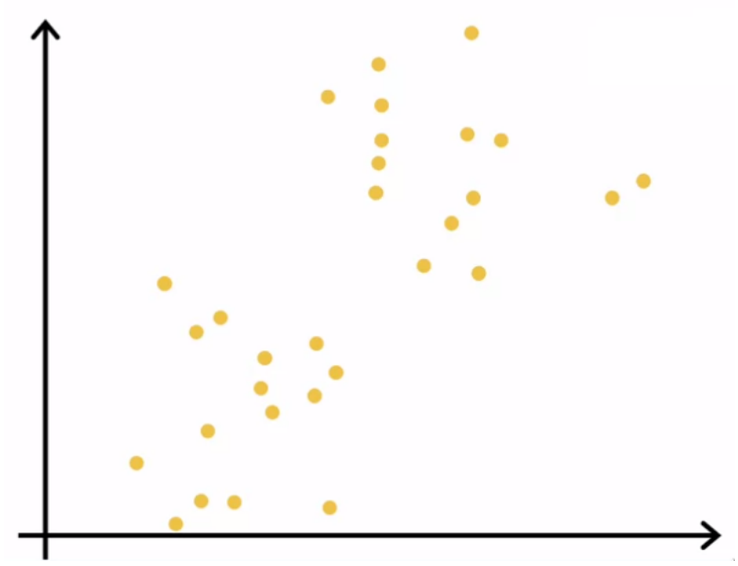
4.5.3 XGBoost

Instead of picking from all training data with equal probability in the random forest algorithm, make it more likely to pick misclassified examples from previous decision trees

5 Unsupervised learning

5.1 Clustering: K-means

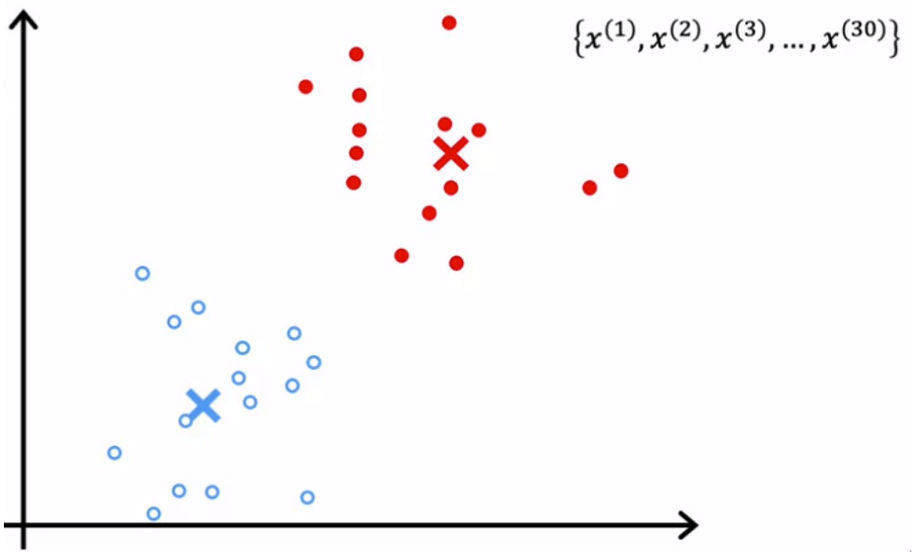
5.1.1 Algorithm



Given a dataset like this, the algorithm will guess the centers of two different clusters (Determining number of clusters will be covered later).

Once two cluster centers (or centroids) are guessed, each data point on the graph will be associated with the centroid it's closest to. The centroid will then move to the average position of all its data points.

Eventually, the centroids will move to the center of the two clusters:



5.1.2 Cost function

k = num clusters

$\vec{c}_i$ = index of cluster (1,2,...,k) to which example $\vec{x}_i$ is currently assigned

μ_i = cluster centroid i

$\mu_{\vec{c}_i}$ = cluster centroid of cluster to which example $\vec{x}_i$ is currently assigned to

$$J(\vec{c}, \vec{\mu}) = \frac{1}{m} \sum_{i=1}^{m} \|\vec{x}_i - \vec{\mu}_{\vec{c}_i}\|^2$$

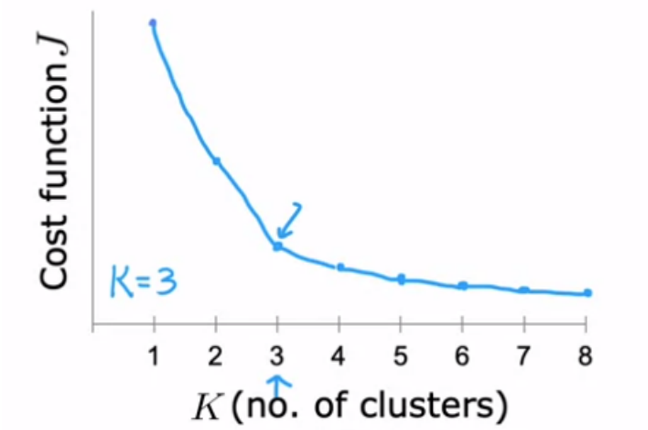
The cost function can be used to determine how well the centroids predicted the clusters, and it can also determine when k-means is converging.

The most optimal centroids can be determined by running k-means multiple times with random initial centroid positions every time, then choosing the result with the lowest cost.

5.1.3 Choosing k

Elbow method

Plot cost as a function of k, choose k where cost begins to decrease at a slow rate.



In this graph, $k = 3$ might be a good number of clusters. Although cost does continue to decrease as k increases beyond 3, the number of clusters is too large and makes for less meaningful clusters.

The "right" value of k is often ambiguous however, which is an issue with the elbow method.

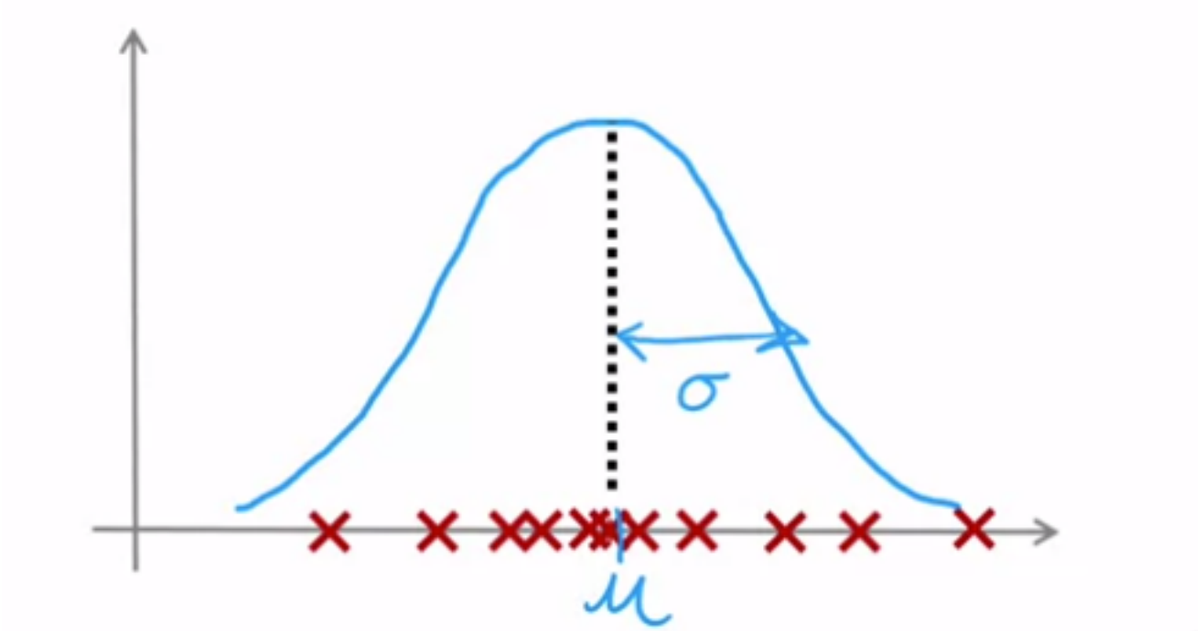
5.2 Anomaly detection

5.2.1 Normal distribution

The equation for the normal distribution is given by

$$p(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Change in μ will shift the curve on the x axis, and change in σ will make the curve thinner or wider. Smaller σ makes curve narrow, larger σ makes curve wide.



Values of μ and σ that produce a normal distribution which will fit the data well can be determined like this:

$$\mu = \frac{1}{m}\sum_{i=1}^{m}\vec{x}_i$$
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(\vec{x}_i - \mu)^2$$

5.2.2 Density estimation

Training set: matrix X of size $m \times n$ (m examples, n features)

$$p(\vec{x}) = \prod_{i=1}^{n} p(\vec{x}_i, \mu_i, \sigma_i^2)$$

5.3 Recommender systems

n = num features
n_i = num items
$r(i,j) = 1$ if user j has rated item i (0 if otherwise)
$y_{i,j}$ = rating given by user j on item i (if defined)
$W_j, \vec{b}_j$ = parameters for user j
X_i = feature vector for item i
$\vec{m}_j$ = number of items user j has rated
For user j, predict rating of item i with $W_j \cdot X_i + \vec{b}_j$
Feature example:

Movie	X_{i1} (Romance)	X_{i2} (Action)	X_{i3} (Horror)
Romance movie ($i = 1$)	1.0	0.1	0.0
Action movie ($i = 2$)	0.0	1.0	0.0
Comedy movie ($i = 3$)	0.5	0.0	0.0
Horror movie ($i = 4$)	0.0	1.0	1.0

Cost function to learn parameters for user j:

$$J(W_j, \vec{b}_j) = \frac{1}{2\vec{m}_j}\sum_{i:r(i,j)=1}(\vec{w}_j \cdot X_i + \vec{b}_j - y_{i,j})^2$$

Learn parameters for all users n_u:
W is a matrix of size $n_u \times n$
$\vec{b}$ is a vector of length n_u

$$J(W, \vec{b}) = \frac{1}{2}\sum_{j=1}^{n_u}\left[\sum_{i:r(i,j)=1}(\vec{w}_j \cdot X_i + \vec{b}_j - y_{i,j})^2\right]$$

5.3.1 Collaborative filtering

Given user parameters $\vec{w}$ and b, predict features.
To learn X_i:

$$J(X_i) = \frac{1}{2}\sum_{j:r(i,j)=1}(\vec{w}_j \cdot X_i + \vec{b}_j - y_{i,j})^2$$

To learn $X_{1\ldots n_m}$:

$$J(X) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}(\vec{w}_j \cdot X_i + \vec{b}_j - y_{i,j})^2$$

5.3.2 Cost function

$$J(\vec{w}, \vec{b}, X) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}(\vec{w}_j \cdot X_i + \vec{b}_j - y_{i,j})^2$$

5.3.3 Binary labels

Predict probability of $y_{i,j} = 1$ using $a_{i,j} = g(\vec{w}_j \cdot X_i + \vec{b}_j)$ where $g(z) = \frac{1}{1+e^{-z}}$
Loss for binary labels

$$L(a_{i,j}, y_{i,j}) = -y_{i,j} \log(a_{i,j}) - (1 - y_{i,j}) \log(1 - a_{i,j})$$

Cost function

$$J(\vec{w}, \vec{b}, X) = \sum_{(i,j):r(i,j)=1} L(g(\vec{w}_j \cdot X_i + \vec{b}_j), y_{i,j})$$

5.3.4 Mean normalization

Allows the algorithm to give better predictions for users who have rated very few movies.
Given a chart like this,

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

create a matrix

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & ? & ? \end{bmatrix}$$

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

After subtracting every value in each row by its corresponding μ:

$$\begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

5.3.5 Finding related items

Find item k with X_k similar to X_i

$$\sum_{l=1}^{n} (X_{k,l} - X_{i,l})^2$$

5.3.6 Content based filtering

Recommends items user might like, in contrast to collaborative filtering which decides if a user would like an item

5.4 Reinforcement learning

5.4.1 State action value function

Denoted by $Q(s, a)$ where s is the current state and a is an action that might be taken in that state.