Machine Learning

Contents

1 Regression

1.1 Linear regression

1.1.1 Squared error cost function

Measures how well line fits training data
m = num of training examples
$\vec{x}$ holds training example x values (length m)
$\vec{y}$ holds training example y values (length m)
$\hat{y}^{(i)} = w\vec{x}^{(i)} + b$

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

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

1.1.2 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

$$\begin{aligned}
w_1 &= w_0 - \alpha\frac{\partial}{\partial w}J(w_0, b_0) \\
b_1 &= b_0 - \alpha\frac{\partial}{\partial b}J(w_0, b_0) \\
\frac{\partial}{\partial w}J(w,b) &= \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - \vec{y}_i)\vec{x}^{(i)} \\
\frac{\partial}{\partial b}J(w,b) &= \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - \vec{y}_i)
\end{aligned}$$

1.2 Multiple linear regression

n_f = number of features
m = number of data points
$\vec{w}$ = vector of weights (length n_f)
X is a list of x vectors which hold n_f features (size $m \times n_f$)
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

$$\begin{aligned}
\vec{w}_j &= \vec{w}_j - \alpha\frac{\partial}{\partial \vec{w}_j}J(\vec{w}, b) \\
b &= b - \alpha\frac{\partial}{\partial b}J(\vec{w}, b)
\end{aligned}$$

Cost function and its partial derivatives

$$\begin{aligned}
J(\vec{w},b) &= \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i)^2 \\
\frac{\partial}{\partial \vec{w}_j}J(\vec{w},b) &= \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i)X_j^{(i)} \\
\frac{\partial}{\partial b}J(\vec{w},b) &= \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(X^{(i)}) - \vec{y}_i)
\end{aligned}$$

1.3 Logistic regression

Sigmoid function

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

$\hat{y}^{(i)}$ 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

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, \vec{y}_i)$$

$$L(\hat{y}^{(i)}, \vec{y}_i) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } \vec{y}_i = 1 \\ -\log(1-\hat{y}^{(i)}) & \text{if } \vec{y}_i = 0 \end{cases}$$

Simplified form

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

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.

1.4 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 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.

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

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

1.5 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)

$$X_j^{(i)} = \frac{X_j^{(i)} - \vec{\mu}_j}{\vec{\sigma}_j}$$
$$\vec{\mu}_j = \frac{1}{m}\sum_{i=0}^{m-1} X_j^{(i)}$$
$$\vec{\sigma}_j^2 = \frac{1}{m}\sum_{i=0}^{m-1}(X_j^{(i)} - \vec{\mu}_j)^2$$

1.6 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")

1.6.1 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

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

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

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

Regularized logistic regression

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

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

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

2 Neural networks

n_ℓ = num layers excluding input
$n_n^{[\ell]}$ = n neurons in layer ℓ
n_f = num features
$\vec{W}$ is a vector (length n_ℓ) of matrices of size $n_n^{[\ell]} \times n_n^{[\ell-1]}$
$\vec{x}$ is a vector of outputs from each neuron in previous layer
$\vec{b}$ is a vector (length n_ℓ), holds a bias value for each layer
Z and A: vector (length n_ℓ) of vectors (length $n_n^{[\ell]}$)
g: activation function
$1 \leq i \leq n_\ell$
a (activation) = scalar output of a single neuron
Superscript $[i]$ is used to notate information relating to the ith layer in a neural network.

2.1 Choosing an activation function

sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
tanh: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
linear: $g(z) = z$
ReLU: $g(z) = \max(0, z)$
Leaky ReLU: $g(z) = \max(\epsilon z, z)$ where ϵ is a small nonzero positive value < 1

For output layer

Binary classification, y = 0 or 1: use sigmoid
Regression, $-\infty \leq y \leq \infty$: use linear activation function
Regression, y $\geq$ 0: use ReLU

For hidden layer

ReLU is most common

2.2 Training a model

2.2.1 Forward propagation

Input $A^{[\ell-1]}$, output $A^{[\ell]}$, cache $Z^{[\ell]}$, $W^{[\ell]}$, $\vec{b}^{[\ell]}$

$$\begin{aligned} Z^{[\ell]} &= W^{[\ell]} A^{[\ell-1]} + \vec{b}^{[\ell]} \\ A^{[\ell]} &= g^{[\ell]}(Z^{[\ell]}) \end{aligned}$$

Up to $A^{[n_\ell]}$, in which case $\hat{y} = A_0^{[n_\ell]}$ assuming output layer has one unit

2.2.2 Back propagation

Input $\mathrm{d}a^{[\ell]}$, output $\mathrm{d}a^{[\ell-1]}$, $\mathrm{d}W^{[\ell-1]}$, $\mathrm{d}\vec{b}^{[\ell-1]}$

$$\begin{aligned} \mathrm{d}Z^{[\ell]} &= \mathrm{d}A^{[\ell]} \cdot g'^{[\ell]}(Z^{[\ell]}) \\ \mathrm{d}W^{[\ell]} &= \frac{1}{m} \mathrm{d}Z^{[\ell]} \cdot A^{[\ell-1]T} \\ \mathrm{d}\vec{b}^{[\ell]} &= \frac{1}{m} \sum_i \mathrm{d}Z_i^{[\ell]} \\ \mathrm{d}A^{[\ell-1]} &= W^{[\ell]T} \cdot \mathrm{d}Z^{[\ell]} \end{aligned}$$

2.3 Improving model

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

2.3.1 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

2.3.2 Adding data

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

3 Decision trees

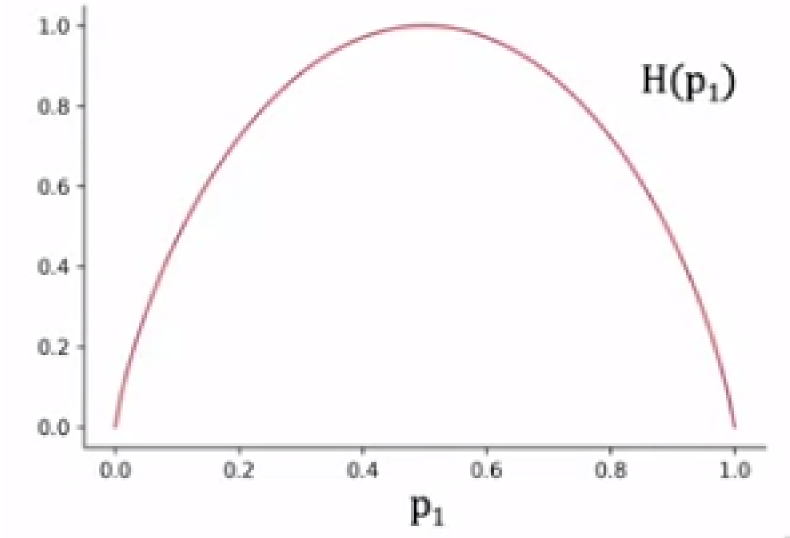
3.1 Measuring purity

3.1.1 Entropy as a measure of impurity

p = fraction of examples that are cats

Cat
Cat
Cat
Dog
Dog
Dog

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

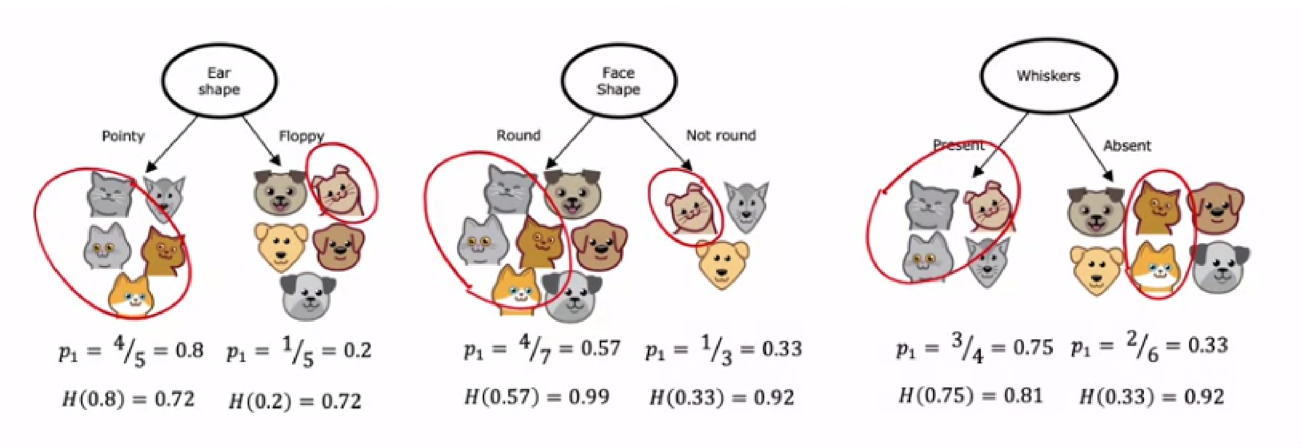


Higher H = less pure, more information gain
Mathematically, H is defined as:

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

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

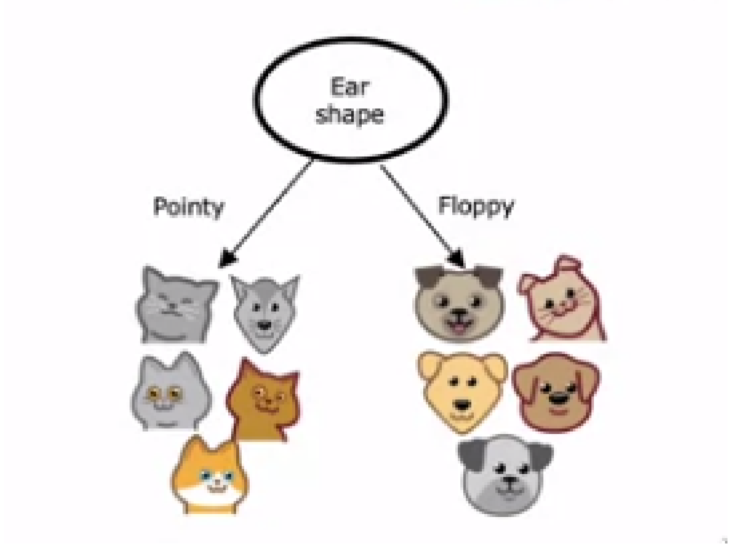
3.2 Choosing a split



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:

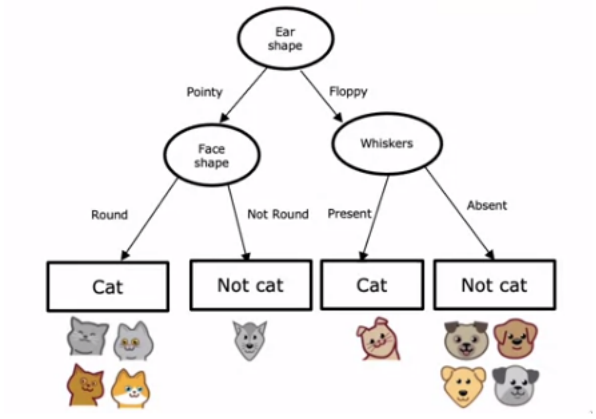


p_{root} = percentage of positive examples (0.5 in this case)
$p_{left} = 4/5$
$p_{right} = 1/5$
$w_{left} = 5/10$
$w_{right} = 5/10$
$H(p_{root}) - (w_{left}H(p_{left}) + w_{right}H(p_{right}))$

3.3 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



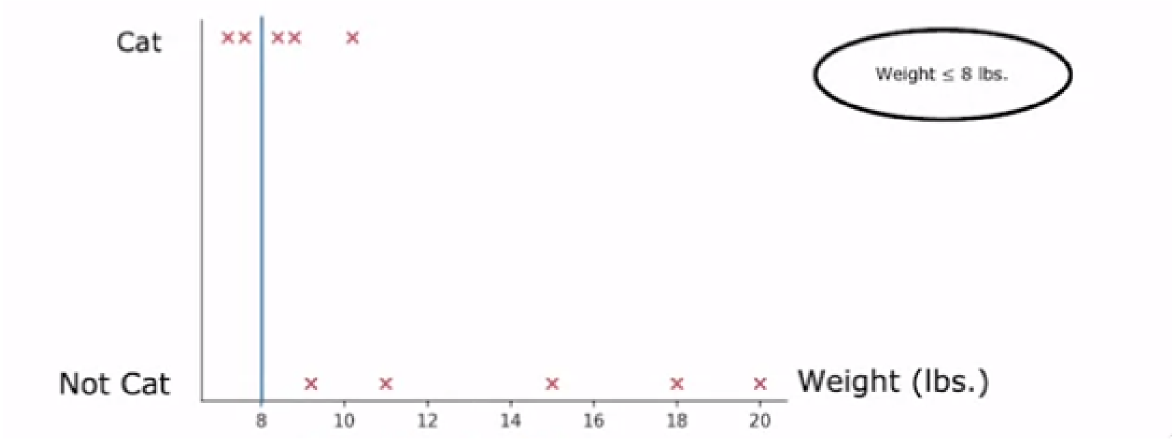
3.4 Features with multiple possible values

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 $\leq$ 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.

3.5 Tree ensembles

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

3.5.1 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.

3.5.2 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 $k = \sqrt{n}$.

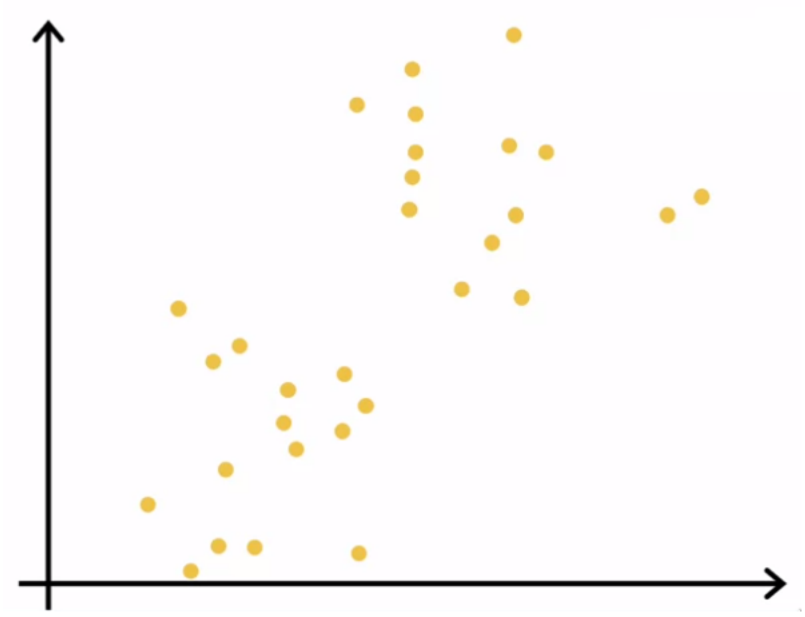
3.5.3 XGBoost

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

4 Unsupervised learning

4.1 Clustering: K-means

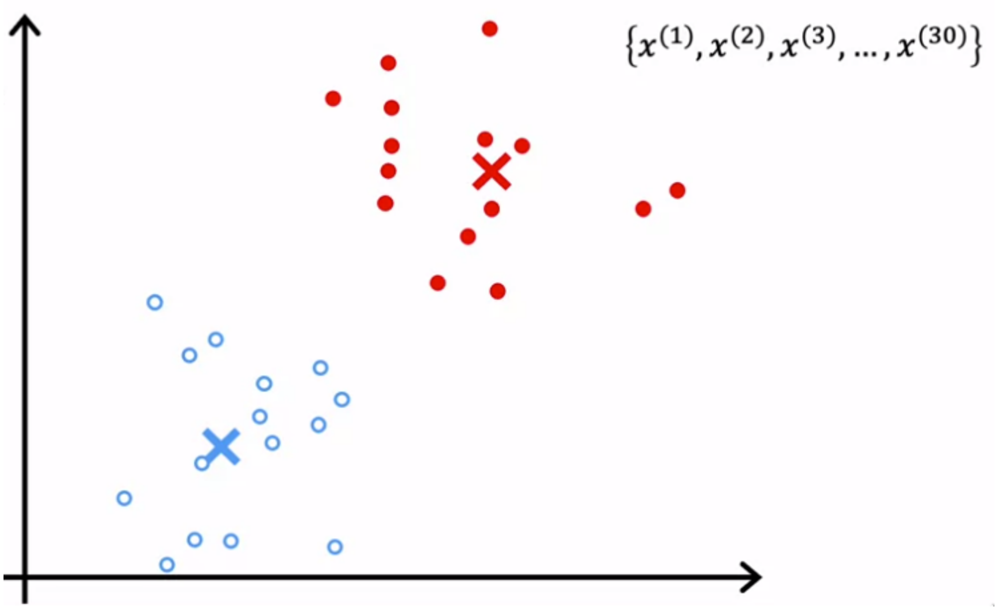
4.1.1 Algorithm



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:



4.1.2 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

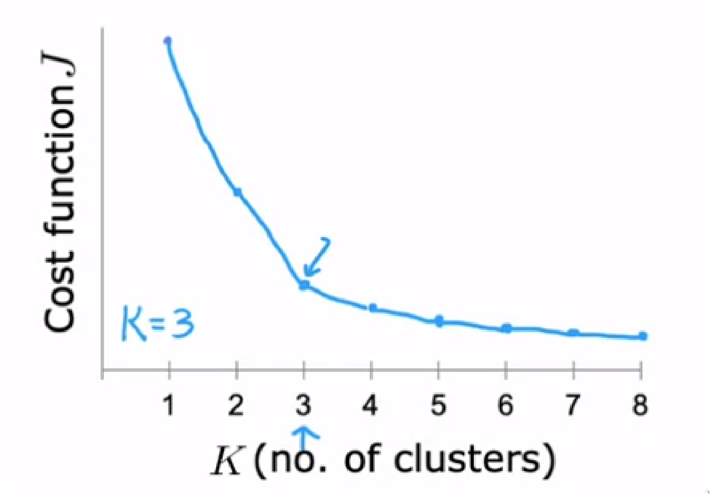
$$J(\vec{c}, \vec{\mu}) = \frac{1}{m} \sum_{i=1}^{m} \|\vec{x}_i - \vec{\mu}_{\vec{c}_i}\|^2$$

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.

4.1.3 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.