CS-541: Artificial Intelligence
Lecture 3

Abdul Rafae Khan
Department of Computer Science
Stevens Institute of Technology
akhan4@stevens.edu

February 7, 2022

Machine Learning Models

Supervised Learning: Train model on observations and class labels
Unsupervised Learning: Train model on observations only

Machine Learning Models

Linear Regression:

$$y = \boldsymbol{w}\boldsymbol{x}$$

$$L(\boldsymbol{w}) = \frac{1}{|X|} \sum_{x \in X} (y - \boldsymbol{w}\boldsymbol{x})^2$$

Machine Learning Models

Logistic Regression:

$$
\begin{aligned}
y &= f(\boldsymbol{w}\boldsymbol{x}) \\
f_{\boldsymbol{w}}(\boldsymbol{x}) &= \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}} \\
L(\boldsymbol{w}) &= \frac{1}{|X|} \sum_{x \in X} (y - f(\boldsymbol{w}\boldsymbol{x}))^2
\end{aligned}
$$

Machine Learning Models

Gradient Descent:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta \nabla L(\boldsymbol{w})$$

$$\nabla L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_1} \\ \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_2} \\ \cdots \\ \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_n} \end{bmatrix}$$

Erratum: Stochastic Gradient Descent vs Mini-batch Gradient Descent

Overview

1. Evaluation
2. Model Features
3. Overfitting
4. Regularization
5. Cross-Validation
6. Decision Tree
7. Unsupervised Learning
   7.1 K-Means
8. Neural Network
   8.1 Activation Functions
   8.2 Forward pass
   8.3 Backpropagation

Evaluation

Regression: We can use the the loss functions as our error function, e.g. MSE, RMSE etc.

Classification: Two types of errors
Type I error: False positive
Type II error: False negative

Evaluation

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

$$Error\ Rate = 1 - Accuracy$$

Machine Learning Models

Linear regression: $y = \boldsymbol{w}\boldsymbol{x}$
Polynomial regression: $y = \boldsymbol{w}\phi(\boldsymbol{x})$

Polynomial Models

Polynomial regression: $y = \boldsymbol{w}\phi(\boldsymbol{x})$
Features: $\phi(x) = \{1, x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1^2x_2, x_1x_2^2, \cdots\}$
Feature Map: ϕ
A function which maps from data to feature space

Feature Extraction

How to convert data into a list of features?

Problem 1: Spam Filtering

Given an email, output whether the email is spam or not spam

What can be possible features?

Feature Extraction

How to convert data into a list of features?

Problem 2: Image Classification

Given an image as input, output if it is a cat or a dog

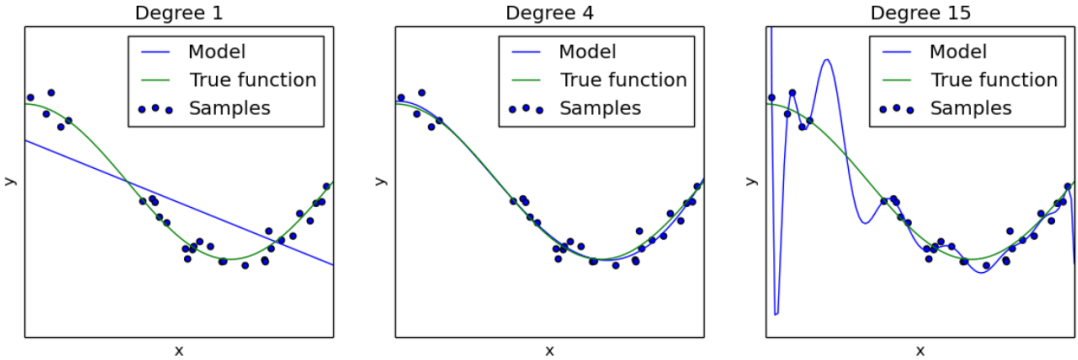
What can be possible features?

Feature Extraction

How to convert data into a list of features?
Based on the problem we are solving and the domain knowledge

Machine Learning Models

How complex the model should be?



Machine Learning Models

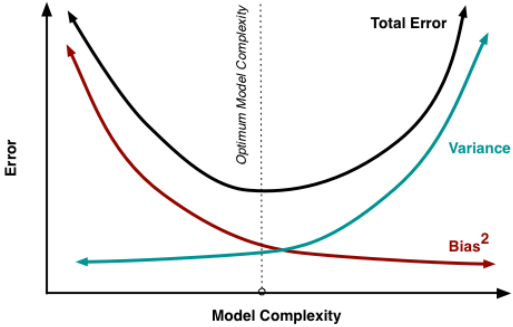
Follow the principles of Occam's razor

Occam's razor: Plurality should not be posited without necessity

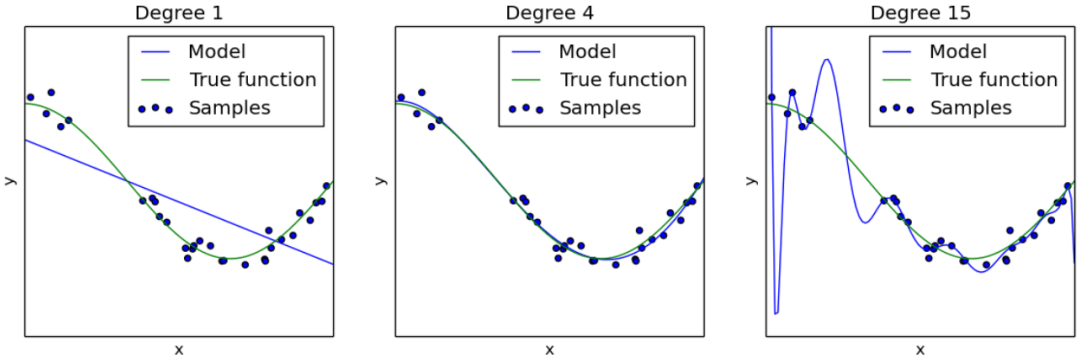
Machine Learning: Prefer the simplest model that fits the data

Overfitting

Simpler models have high **bias**
Complex models have high **variance**
Ideally, we want a model with low bias and low variance



Overfitting



Degree 1 model is underfitting the training data
Degree 15 model overfitting the training data

Overfitting

If we want a model to perform extremely well on the training data only
We can use a lookup table
Our goal is to perform well on any new data
Therefore, we will consider the test data as our unseen dataset

Regularization

A way to restrict model complexity is regularization
Add an additional penalty to the loss function
This penalty should control the model parameters from taking extreme values
Regularization terms:

$$\text{L1} = \lambda \frac{1}{N} \sum_{j=1}^{d} |w_d|$$

$$\text{L2} = \lambda \frac{1}{N} \sum_{j=1}^{d} w_d^2$$

Regularization

λ controls how much penalty should be added to the loss function
$\lambda = 0$ means no penalty is added
λ too large means model may underfit

$$\begin{aligned} \text{L1} &= \lambda \frac{1}{N} \sum_{j=1}^{d} |w_d| \\ \text{L2} &= \lambda \frac{1}{N} \sum_{j=1}^{d} w_d^2 \end{aligned}$$

Regularization

Loss function without Regularization:

$$L(w, y) = (y - wx)^2$$
$$\nabla L(w, y) = -2x(y - wx)$$

Regularization

Loss function with L1 Regularization:

$$L(w, y) = (y - wx)^2 + \lambda |w|$$

$$\nabla L(w, y) = \begin{cases} -2x(y - wx) + \lambda \text{ if } w > 0 \\ -2x(y - wx) - \lambda \text{ if } w < 0 \end{cases}$$

Because $|w|$ for $w = 0$ is not differentiable

Regularization

Loss function with L2 Regularization:

$$L(w, y) = (y - wx)^2 + \lambda w^2$$
$$\nabla L(w, y) = -2x(y - wx) + 2\lambda w$$

Cross-Validation

We now have another parameter to optimize (λ)
Use can use cross-validation

Cross-Validation

- Leave-one-out: 1 test & N-1 training samples
- Leave-k-out: k test & $N - k$ training samples
- k-fold cross validation: N/k test & $N - (N/k)$ training samples:
- Holdout cross validation: $k\%$ test and $(100 - k)\%$ training samples
- Random subsampling: Same as k-fold cross validation but samples selected randomly without replacement (Shuffle and then do k-fold cross validation)

There can be reasons to select one over the other
e.g. training time or data imbalance etc.

Cross-Validation

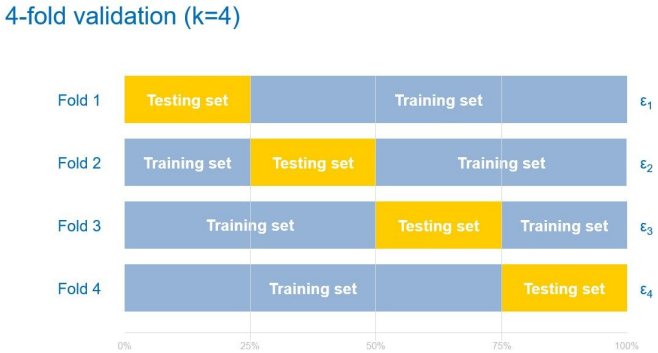
Lets say we do a 4-fold cross validation

We get an performance metric (e.g. error rate) for each fold

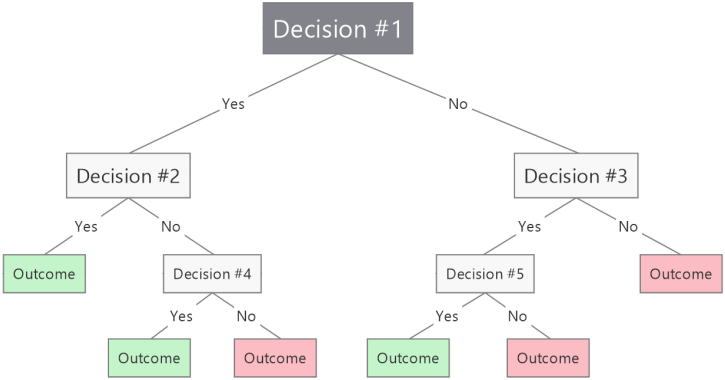
Then we can either

1) Pick the model parameters which give the lowest error rate, or

2) Report the average error rate for all the models



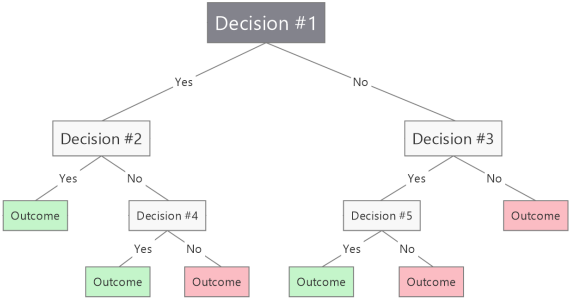
Decision Tree



Decision Tree

Decision trees do not have to be binary
Easy to interpret

Decision #1, Decision #2 etc are **decision nodes**
They define the **decision rules** on how to split the data
Outcome are the **leaves**/**terminal nodes**



Decision Tree

```
if decision1 and decision2 and decision4:
    return outcome1
else if decision1 and decision2 and !decision4:
    return outcome2
else if !decision1 and decision3 and decision5:
    return outcome1
else if !decision1 and decision3 and !decision5:
    return outcome2
else if !decision1 and !decision3 and !decision5:
    return outcome2
```

Decision Tree

Information Gain = Entropy before the split - Total Entropy after the split
Entropy measures the uncertainty or randomness within a set

$$Entropy = -1 * \sum_{i=1}^{N} p_i \log_2 p_i$$

Decision Tree

Lets say $X = [0, 1, 0, 0, 1]$

$$
\begin{aligned}
Entropy(X) &= -P(X=0)\log_2(P(X=0)) - P(X=1)\log_2(P(X=1)) \\
&= -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \\
&= 0.442 + 0.529 = 0.971
\end{aligned}
$$

Decision Tree

One algorithm to create a decision tree is **ID3**

Root node is the complete dataset
For every node, pick the feature for the split which maximizes the information gain

Decision Tree

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Decision Tree

We compute the information gain for all the features

$$IG(Play, Feature) = E(Play) - E(Play, Feature)$$

Then we pick the feature with the highest information gain for the split

Decision Tree

Calculating the entropy

Play	
Yes	No
9	5

$$
\begin{aligned}
Entropy(Play) &= E(9,5) \\
&= -1 * \{\frac{9}{14}\log_2(\frac{9}{14}) + \frac{5}{14}\log_2(\frac{5}{14})\} \\
&= -1 * (-0.412 - 0.531) \\
&= 0.940
\end{aligned}
$$

Decision Tree

		Play		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rain	2	3	5
				14

$$
\begin{aligned}
Entropy(Play, Outlook) &= P(Sunny) * E(3,2) + P(Overcast) * E(4,0) + P(Rain) * E(2,3) \\
&= \frac{5}{14} * 0.971 + \frac{4}{14} * 0.0 + \frac{5}{14} * 0.971 \\
&= 0.693
\end{aligned}
$$

Decision Tree

Calculating the informartion gain

$$
\begin{aligned}
IG(Play, Outlook) &= Entropy(Play) - Entropy(Play, Outlook) \\
&= 0.940 - 0.693 \\
&= 0.247
\end{aligned}
$$

$$
\begin{aligned}
IG(Play, Temperature) &= Entropy(Play) - Entropy(Play, Temperature) \\
&= 0.029
\end{aligned}
$$

$$
\begin{aligned}
IG(Play, Humidity) &= Entropy(Play) - Entropy(Play, Humidity) \\
&= 0.152
\end{aligned}
$$

$$
\begin{aligned}
IG(Play, Wind) &= Entropy(Play) - Entropy(Play, Wind) \\
&= 0.048
\end{aligned}
$$

Decision Tree

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

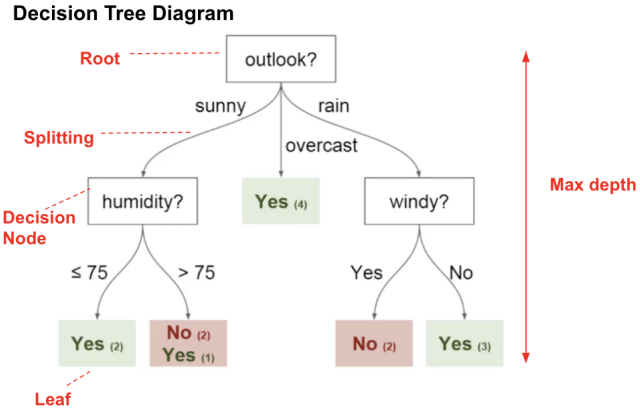
Decision Tree

Overcast already has only one set of labels
Therefore no need to split it further
For other two
We repeat the same process again
For example, for Sunny

$$IG(Play, Temperature) = 0.571$$
$$IG(Play, Humidity) = 0.971$$
$$IG(Play, Wind) = 0.020$$

Decision Tree

Final decision Tree

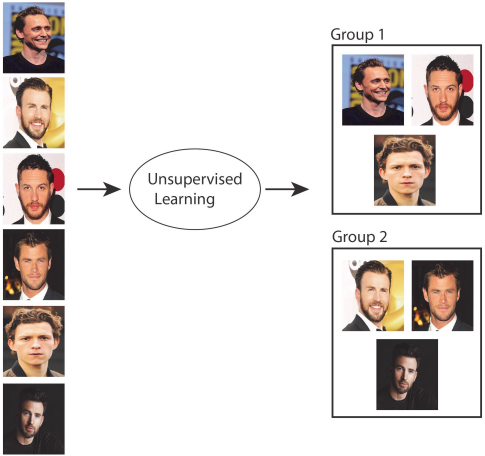


Unsupervised Learning

Two main types of learning approaches

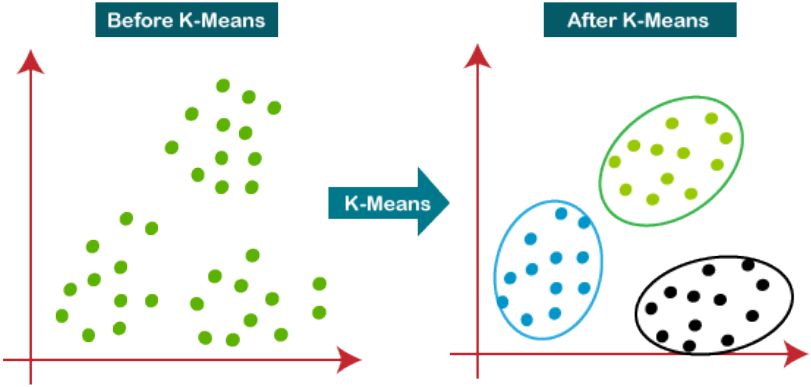
- Supervised learning
  - The data is labelled/annotated
- Unsupervised learning
  - The data is unlabelled

Unsupervised Learning



K-Means

Group data points based on some similarity/distance metric
Commonly used distance measures are euclidean and manhattan distances



K-Means

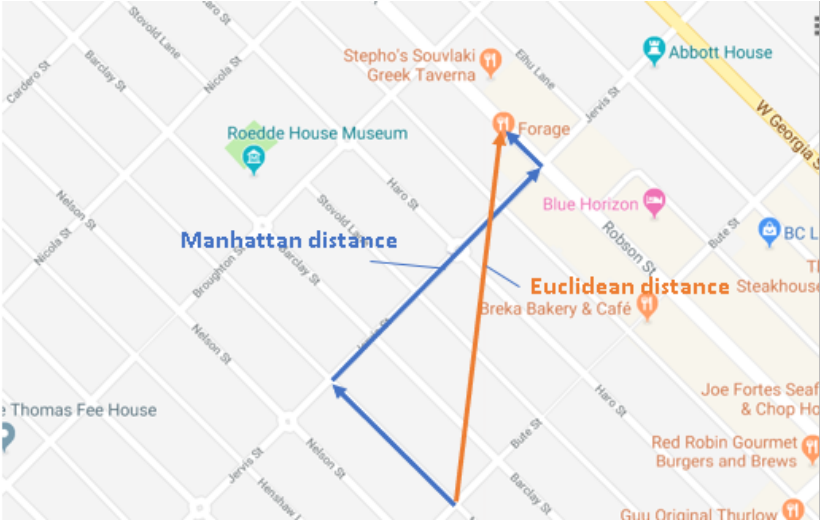
Lets say we have 2 points $a = (x_1, y_1)$ & $b = (x_2, y_2)$
Two Commonly used distance measures are:

$$\text{Manhattan Distance } (a, b) = |x_2 - x_1| + |y_2 - y_1|$$

$$\text{Euclidean Distance } (a, b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

K-Means

Group data points based on some similarity/distance metric



K-Means

```
1. cluster centers = select K points randomly
2. while (!stopping criteria):
2.      for each point p in data
3.          c_i = closest cluster center from p
4.          assign p to cluster c_i
5.      for each cluster
6.          re-compute cluster center
```

K-Means

The algorithm needs a K as input

Stopping criteria can be the cluster centers are not changing by some threshold value
$\text{distance}(center_{old}, center_{new}) \leq threshold$

Given a new data point,
assign it to the cluster with the closest cluster center

K-Means

Running Time:
The running time is $O(NKT)$,
where N is the number of data points
K is the number of clusters
And T is the number of iterations

K-Means

How do you find the optimal K?
Use metrics to compute how good the clustering is

K-Means

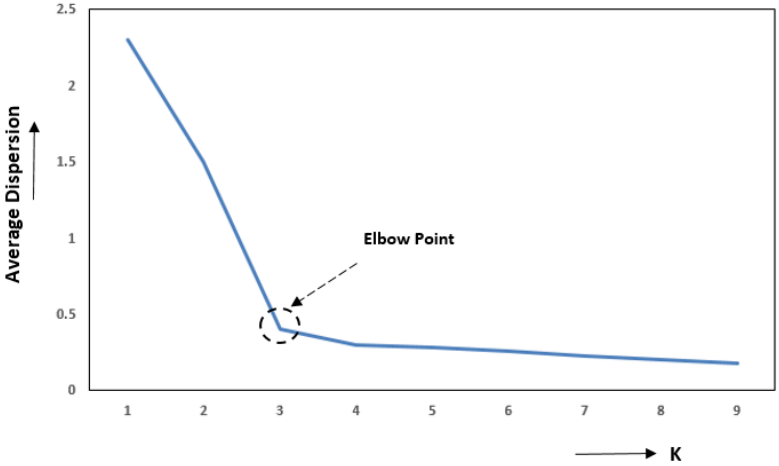
Elbow Method

Distances within a cluster should be minimum

You compute the sum of distance within cluster distances for different K

And pick the one where the change stops

K-Means



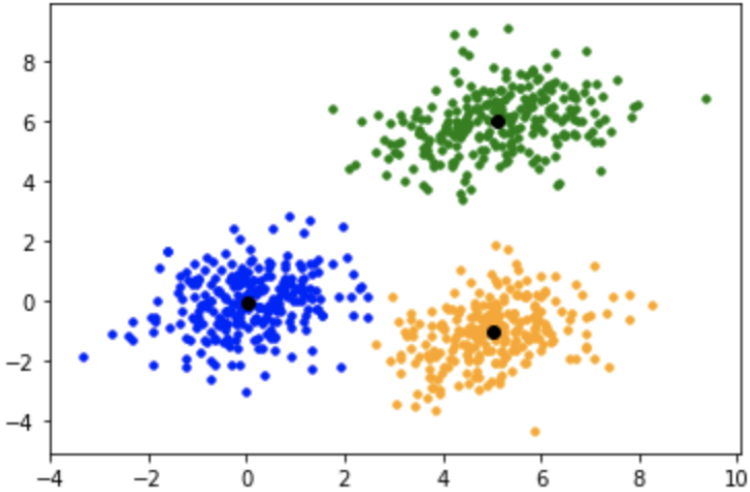
K-Means

Shape of cluster:

Shape of the clusters are dependent upon the distance measure

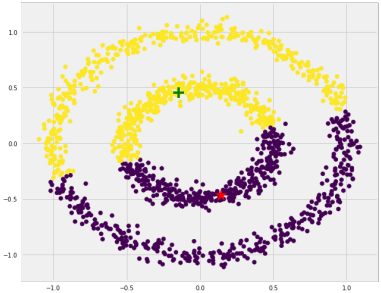
K-Means

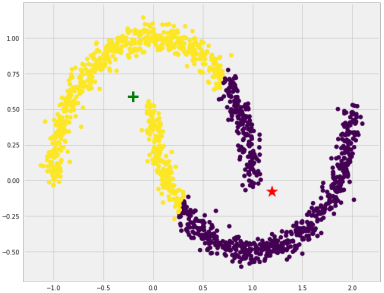
Suitable for K-Means



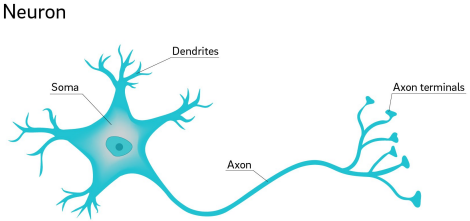
K-Means

Incorrect results using K-Means





Neural Network



Initial idea came from how neurons fire in human body

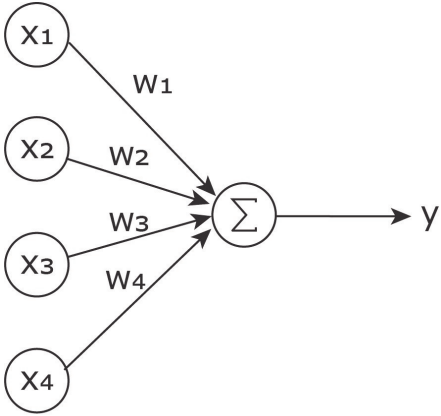
1. Dendritic network takes input from other neurons
2. Signals received are aggregated in the nucleus
3. If it exceeds a threshold, the axon passes a signal down to the other neurons

Neural Network

Linear regression:

$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n$

$y = \sum_{i=1}^{n} w_ix_i$

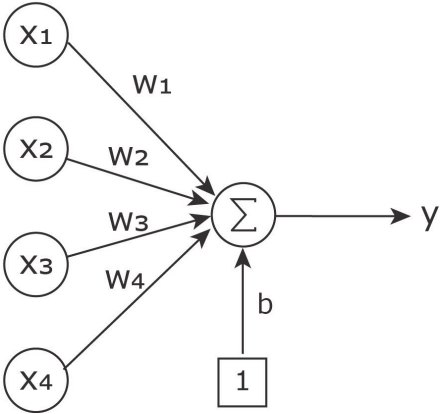


Neural Network

Linear regression (with bias term):

$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

$y = \sum_{i=1}^{n} w_ix_i + b$

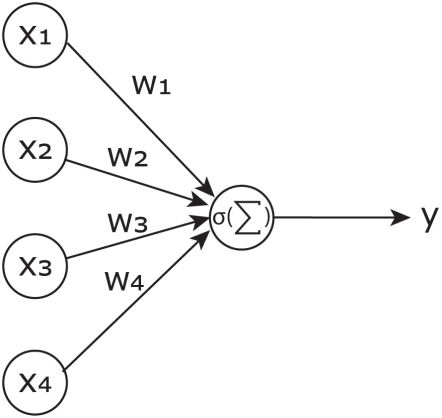


Neural Network

Non-linear Neural Network:

$y = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n)$

$y = \sigma(\sum_{i=1}^{n} w_i x_i)$



Neural Network

The network can be defined as:

$$
\begin{aligned}
f &: \mathbb{R}^4 \mapsto \mathbb{R} \\
z^{(1)} &= w_1^{(0)} x_1^{(0)} + w_2^{(0)} x_2^{(0)} + w_3^{(0)} x_3^{(0)} + w_4^{(0)} x_4^{(0)} \\
x^{(1)} &= \sigma\big(z^{(1)}\big) \\
f\big(x^{(0)}\big) &= x^{(1)}
\end{aligned}
$$

Neural Network

The network can be defined as:

$$
\begin{aligned}
f &: \mathbb{R}^4 \mapsto \mathbb{R} \\
z^{(1)} &= w_1^{(0)} x_1^{(0)} + w_2^{(0)} x_2^{(0)} + w_3^{(0)} x_3^{(0)} + w_4^{(0)} x_4^{(0)} \\
x^{(1)} &= \sigma(z^{(1)}) \\
f(x^{(0)}) &= x^{(1)}
\end{aligned}
$$

Here

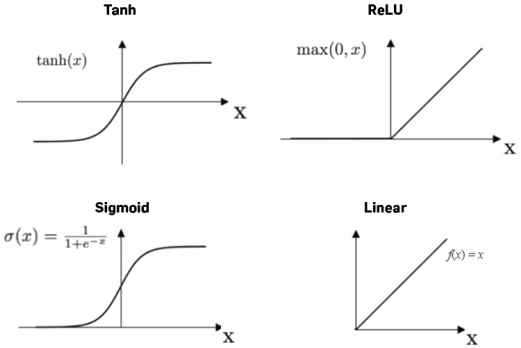
$\left[w_1^{(0)}, w_2^{(0)}, w_3^{(0)}, w_4^{(0)}\right]$ are the network weights

σ is the activation function

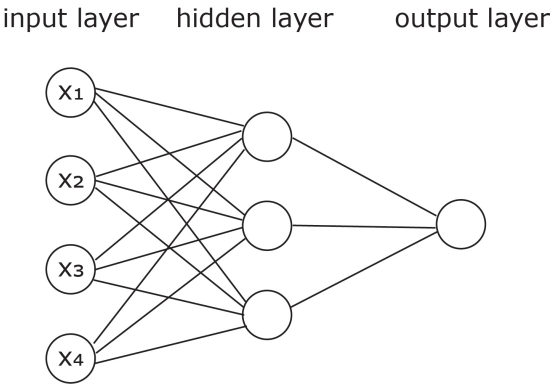
$x^{(1)}$ is the hidden layer output

Neural Network

Some commonly used activation functions



Neural Network



Neural Network

Typically softmax activation is used on output layer
It converts output scores to probabilities
Think of it as an extension of the sigmoid for multiclass classificaiton

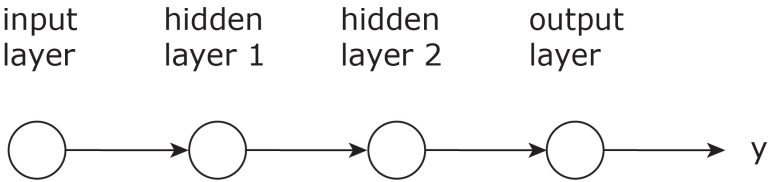
$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{i=1}^{d} e^{z_i}}$$

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Neural Network

How are neural networks trained?

Neural Network



Neural Network

$$f : \mathbb{R} \mapsto \mathbb{R}$$

input: $x^{(0)} \in \mathbb{R}$

$$z^{(1)} = w^{(0)} x^{(0)}$$

$$x^{(1)} = \max\{0, z^{(1)}\}$$

$$z^{(2)} = w^{(1)} x^{(1)}$$

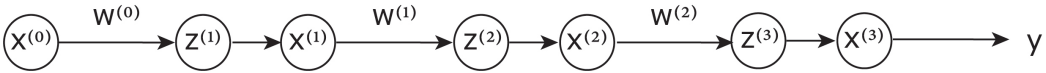
$$x^{(2)} = \max\{0, z^{(2)}\}$$

$$z^{(3)} = w^{(2)} x^{(2)}$$

$$x^{(3)} = softmax(z^{(3)})$$

output: $f(x^{(0)}) = x^{(3)} \in \mathbb{R}$

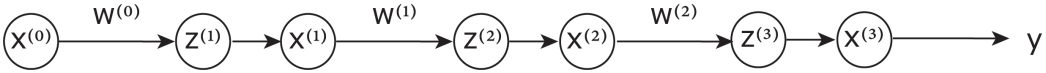
Neural Network



Neural Network

Step 1: Forward pass

Compute each equation until you get the final output



Neural Network

Step 1: Forward pass
Lets say $x^{(0)} = 1, w^{(0)} = 0.5, w^{(1)} = 0.1, w^{(1)} = 0.3$
What is $f(x^{(0)})$?

Neural Network

Step 1: Forward pass
Lets say $x^{(0)} = 1, w^{(0)} = 0.5, w^{(1)} = 0.1, w^{(1)} = 0.3$

$$z^{(1)} = w^{(0)}x^{(0)} = 1 * 0.5 = 0.5$$
$$x^{(1)} = \max\{0, z^{(1)}\} = 0.5$$
$$z^{(2)} = w^{(1)}x^{(1)} = 0.1 * 0.5 = 0.05$$
$$x^{(2)} = \max\{0, z^{(2)}\} = 0.05$$
$$z^{(3)} = w^{(2)}x^{(2)} = 0.3 * 0.05 = 0.015$$
$$x^{(3)} = softmax(z^{(3)}) = \frac{e^{0.015}}{e^{0.015}} = 1$$

Neural Network

Step 2: Backward pass

Backpropagation:

Compute how much change we should make to the weights

Chain Rule:

$$h(x) = g(f(x))$$

$$h`(x) = g`(f(x))f`(x)$$

E.g.

$$h(x) = \sqrt{x + 2x^2}$$

$$h`(x) = \frac{1}{2}(x + 2x^2) + (1 + 4x)$$

Neural Network

We define loss function as

$$L(f(x), y) = \frac{1}{2}(f(x) - y)^2$$

Therefore we need to optimize the following

$$\underset{\{w^{(0)}, w^{(1)}, w^{(2)}\}}{\arg\min} \frac{1}{N} \sum_{j=1}^{N} L(f(x_j, y_j))$$

Neural Network

Recap Stochastic Gradient Descent (SGD)

Randomly select a sample x_j and true label y_j
for $j \in \{1, 2, \cdots, N\}$

$$w_{new} \leftarrow w_{old} - \eta \frac{\partial L(f(x_j) - y_j)}{\partial w_{old}}$$

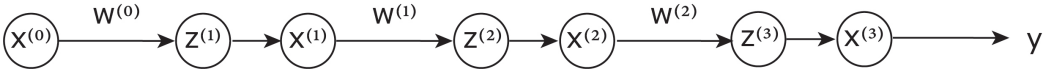
Neural Network

We have to update loss function w.r.t each of the weights

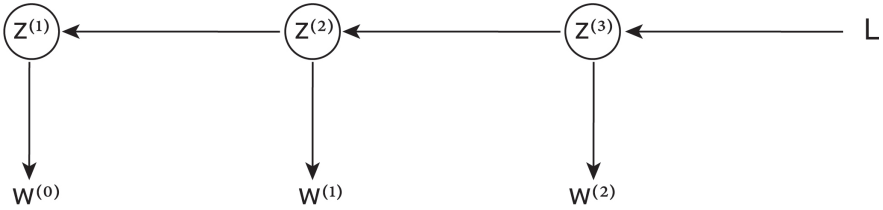
$$w^{(0)} \leftarrow w^{(0)} - \eta \frac{\partial L}{\partial w^{(0)}}$$
$$w^{(1)} \leftarrow w^{(1)} - \eta \frac{\partial L}{\partial w^{(1)}}$$
$$w^{(2)} \leftarrow w^{(2)} - \eta \frac{\partial L}{\partial w^{(2)}}$$

Neural Network

Forward pass



Backward pass



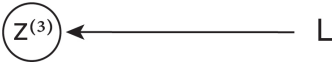
Neural Network

How to compute

$$\frac{\partial L(f(x) - y)}{\partial w}$$

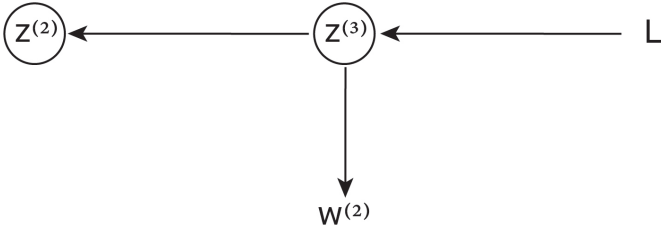
Neural Network

Propagate gradient back from L to $z^{(3)}$



$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial x^{(3)}} \cdot \frac{\partial x^{(3)}}{\partial z^{(3)}} \tag{1}$$

Neural Network



$z^{(3)}$ is a function of $z^{(2)}$ and $w^{(2)}$

$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot \frac{\partial L}{\partial z^{(3)}} \qquad \frac{\partial L}{\partial w^{(2)}} = \frac{\partial z^{(3)}}{\partial w^{(2)}} \cdot \frac{\partial L}{\partial z^{(3)}}$$

Neural Network

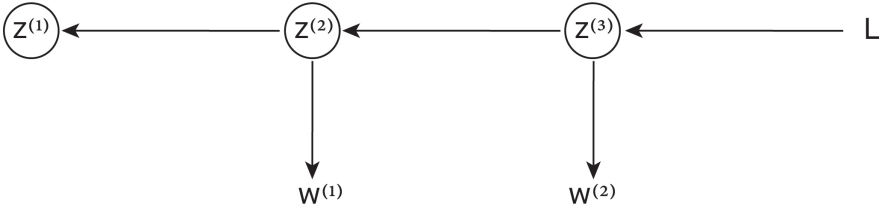
$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot \frac{\partial L}{\partial z^{(3)}} \tag{2}$$

We can now update $w^{(2)}$ using the following equation

$$\frac{\partial L}{\partial w^{(2)}} = \frac{\partial z^{(3)}}{\partial w^{(2)}} \cdot \boxed{\frac{\partial L}{\partial z^{(3)}}}$$

We already computed $\frac{\partial L}{\partial z^{(3)}}$ in the Equation (1)

Neural Network



$z^{(2)}$ is a function of $z^{(1)}$ and $w^{(1)}$

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial z^{(2)}}{\partial z^{(1)}} \cdot \frac{\partial L}{\partial z^{(2)}} \qquad \frac{\partial L}{\partial w^{(1)}} = \frac{\partial z^{(2)}}{\partial w^{(1)}} \cdot \frac{\partial L}{\partial z^{(2)}}$$

Neural Network

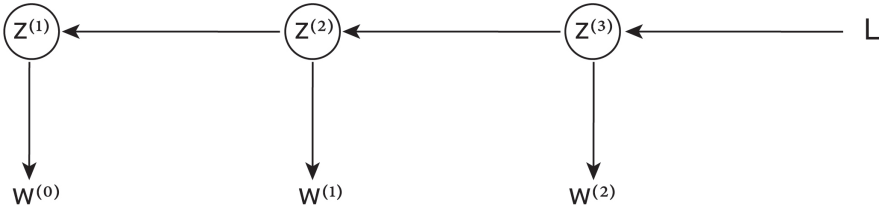
$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial z^{(2)}}{\partial z^{(1)}} \cdot \frac{\partial L}{\partial z^{(2)}} \tag{3}$$

We can now update $w^{(1)}$ using the following equation

$$\frac{\partial L}{\partial w^{(1)}} = \frac{\partial z^{(2)}}{\partial w^{(1)}} \cdot \boxed{\frac{\partial L}{\partial z^{(2)}}}$$

We already computed $\frac{\partial L}{\partial z^{(2)}}$ in the Equation (2)

Neural Network



$z^{(1)}$ is a function of $w^{(0)}$

$$\frac{\partial L}{\partial w^{(0)}} = \frac{\partial z^{(1)}}{\partial w^{(0)}} \cdot \frac{\partial L}{\partial z^{(1)}}$$

Neural Network

We can now update $w^{(0)}$ using the following equation

$$\frac{\partial L}{\partial w^{(0)}} = \frac{\partial z^{(1)}}{\partial w^{(0)}} \cdot \boxed{\frac{\partial L}{\partial z^{(1)}}}$$

We already computed $\frac{\partial L}{\partial z^{(1)}}$ in the Equation (3)

Neural Network

Using SGD we can now update all the weights

$$w^{(0)} \leftarrow w^{(0)} - \eta \frac{\partial L}{\partial w^{(0)}}$$
$$w^{(1)} \leftarrow w^{(1)} - \eta \frac{\partial L}{\partial w^{(1)}}$$
$$w^{(2)} \leftarrow w^{(2)} - \eta \frac{\partial L}{\partial w^{(2)}}$$

Recap

1. **Feature Extraction**
2. **Overfitting** vs **Underfitting**
3. **Regularization:** L1 & L2
4. **Cross-Validation:** Tune λ
5. **Decision Tree:** ID3 Algorithm
6. **Decision Tree:** ID3 Algorithm
7. **Unsupervised Learning:** K-Means
8. **Neural Network:** Activations, Forward pass & Backpropagation

The End