COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Machine Learning

Dr. Martin “Doc” Carlisle

Machine Learning

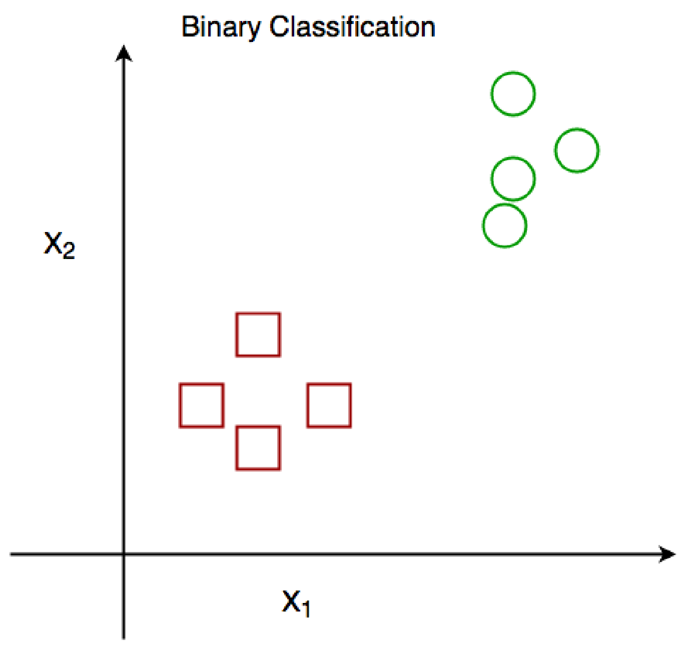
Computer Algorithms that improve with “experience”

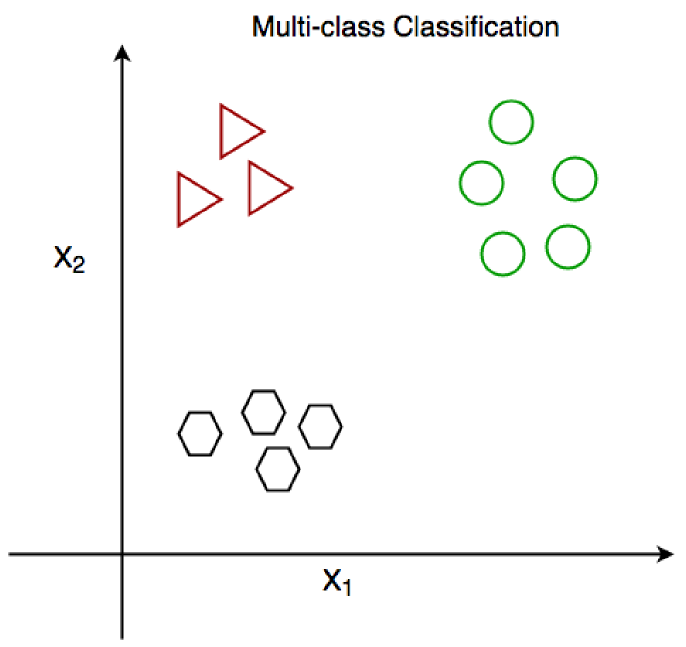
Do we have labeled data? (review)

- Supervised
  - Can train on data with labeled instances of normal vs anomaly classes
  - Not very common
- Semisupervised
  - Labeled instances for only normal data
- Unsupervised
  - No labeled data

Two key problems

- Classification

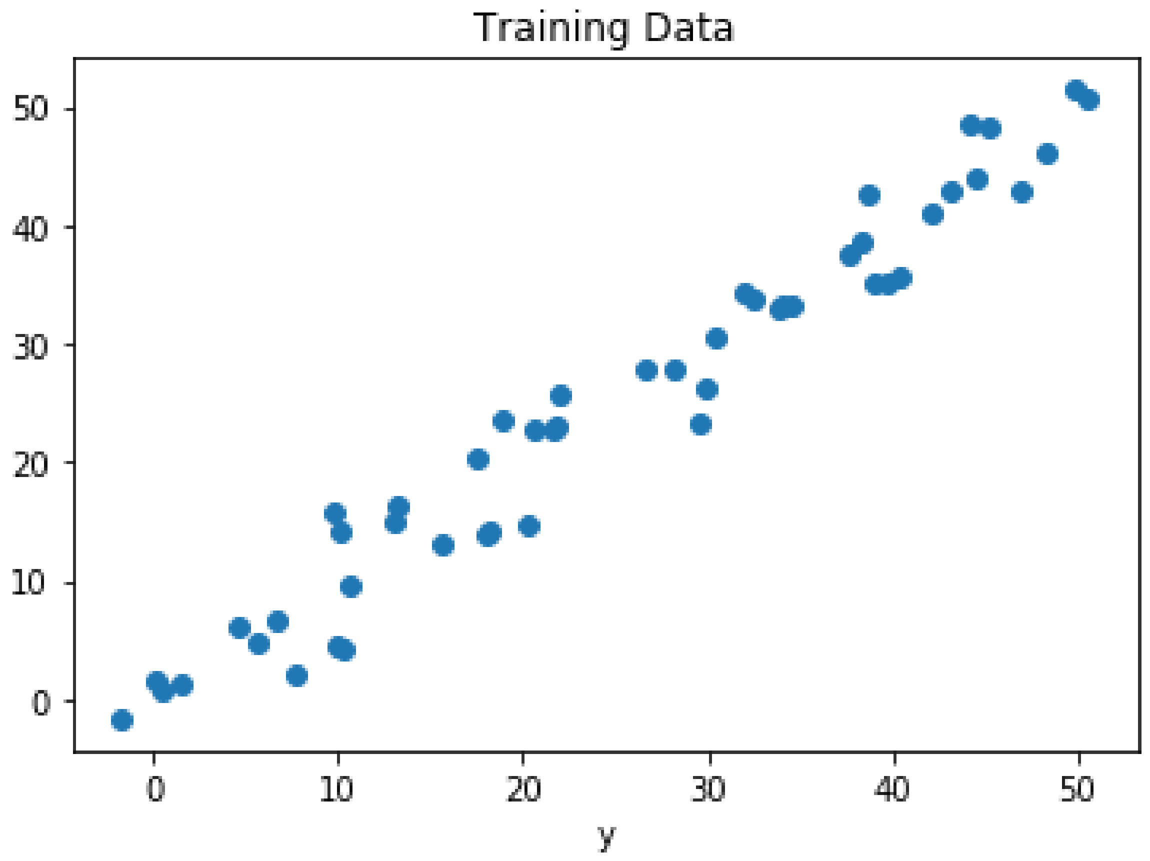




COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Two key problems

- Regression



Classification vs Regression

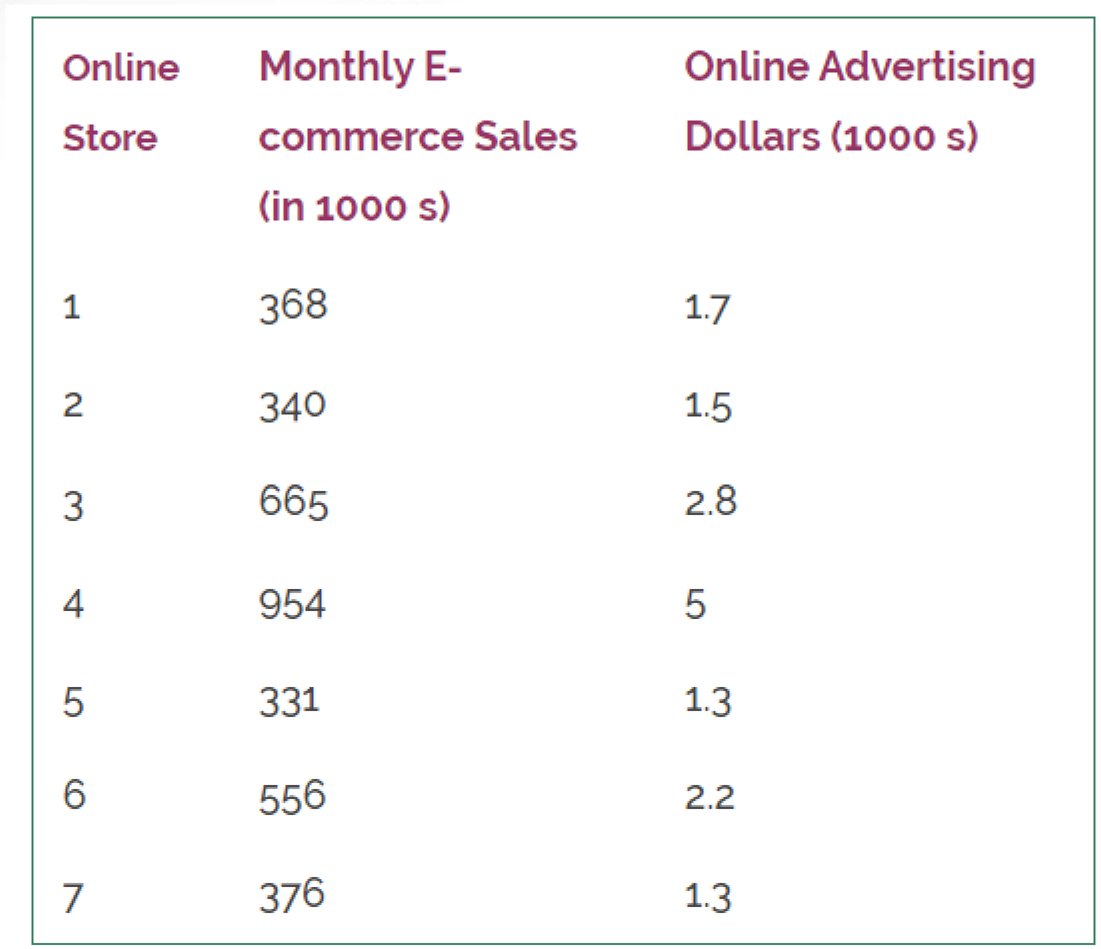
- Classification
  - Map input to discrete value
  - Data is unordered
  - Evaluate by # of correct classifications

$$\sum_{i=1}^{n} I(f(X_i) \neq Y_i)/n$$

- Regression
  - Map input to continuous value
  - Data is ordered
  - Evaluate by root mean squared error

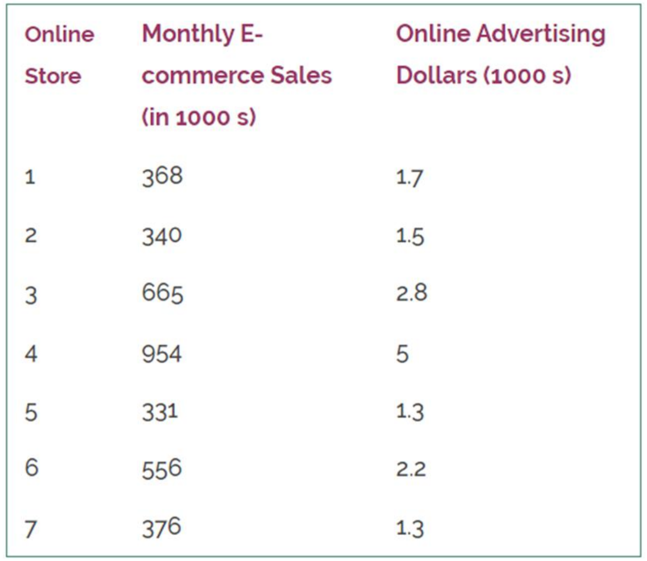
$$\sum_{i=1}^{n} (f(X_i) - Y_i)^2/n$$

Regression Example

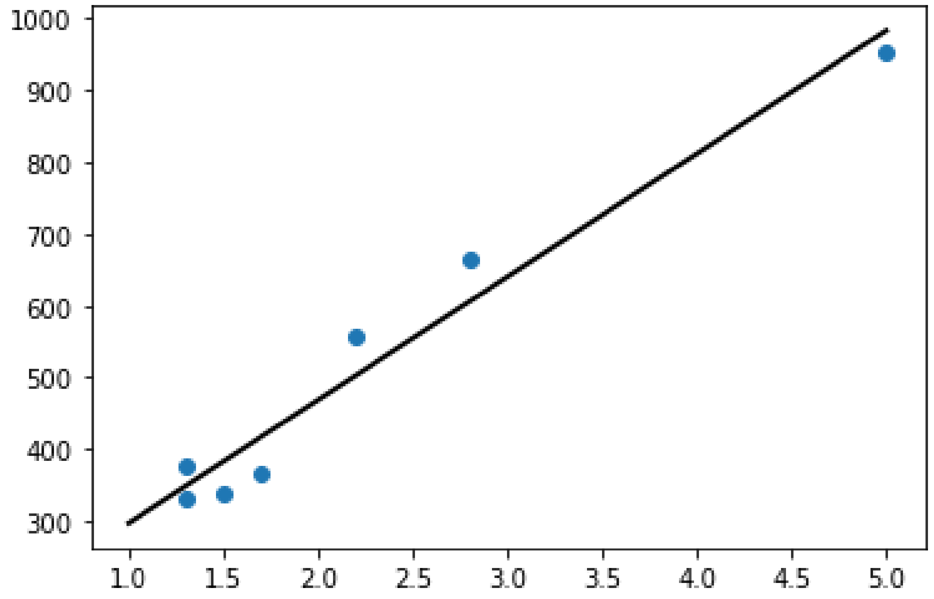


Online Store	Monthly E-commerce Sales (in 1000 s)	Online Advertising Dollars (1000 s)
1	368	1.7
2	340	1.5
3	665	2.8
4	954	5
5	331	1.3
6	556	2.2
7	376	1.3

Regression Example

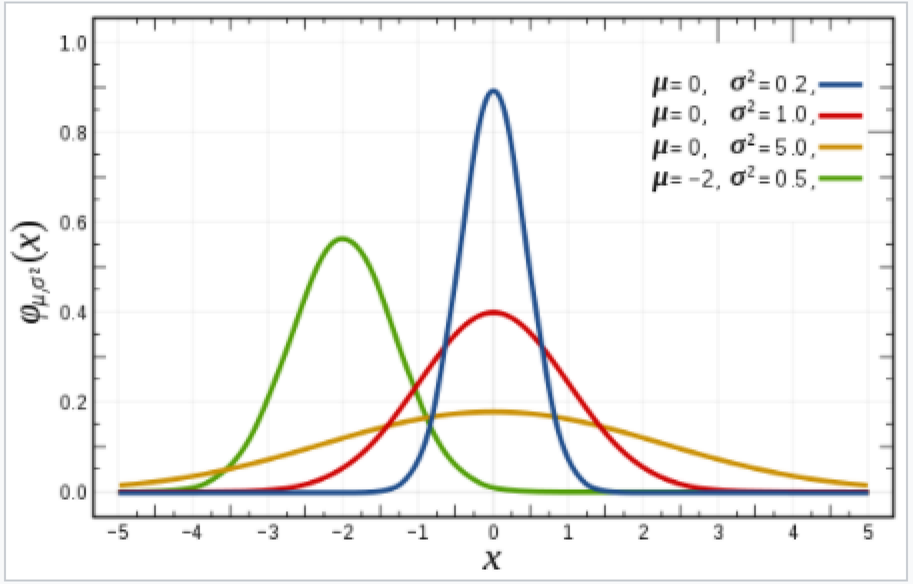


Online Store	Monthly E-commerce Sales (in 1000 s)	Online Advertising Dollars (1000 s)
1	368	1.7
2	340	1.5
3	665	2.8
4	954	5
5	331	1.3
6	556	2.2
7	376	1.3



Jupyter Notebook

- See RegressionExample.ipynb for more examples, including with Gaussian functions

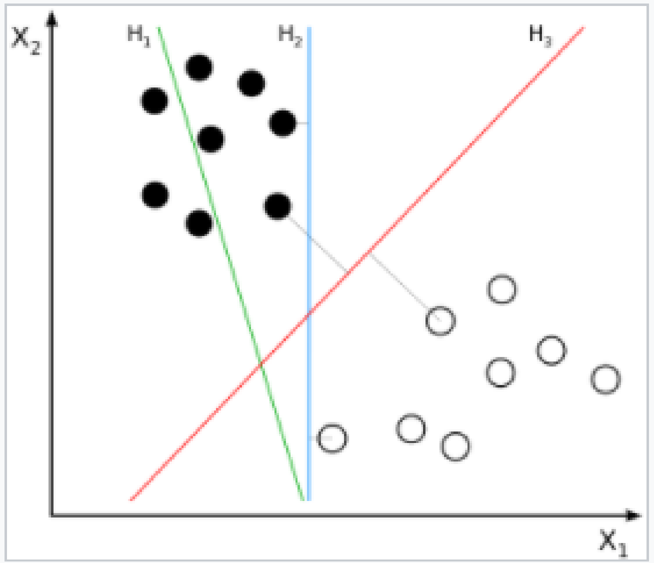


Normalized Gaussian curves with expected value μ and variance σ^2. The corresponding parameters are $a = \frac{1}{\sigma\sqrt{2\pi}}$, $b = \mu$ and $c = \sigma$.

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Classification

- Support Vector Classifier
  - Finds hyperplane(s) to split data



H_1 does not separate the classes.
H_2 does, but only with a small margin.
H_3 separates them with the maximal margin.

SVC margin

- Find hyperplane with largest margin
  - Sometimes you allow some samples to be miscategorized

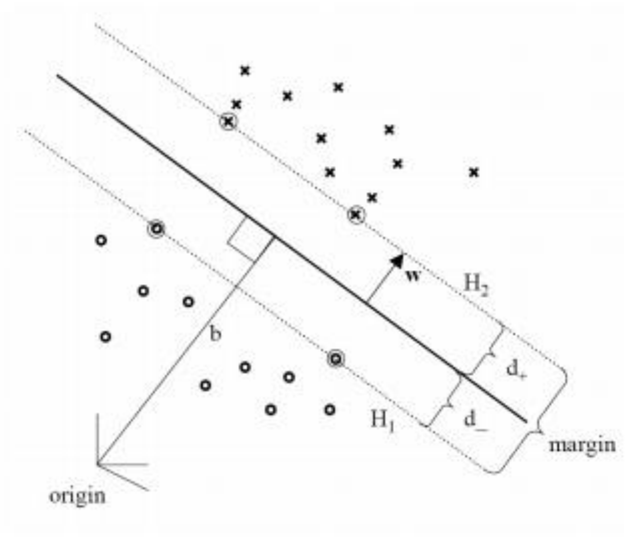


Figure 2: Optimal separating hyperplane with maximum margin

SVC Kernels

- Sometimes data isn't linearly separable
  - Use kernel function to map to linearly separable feature space

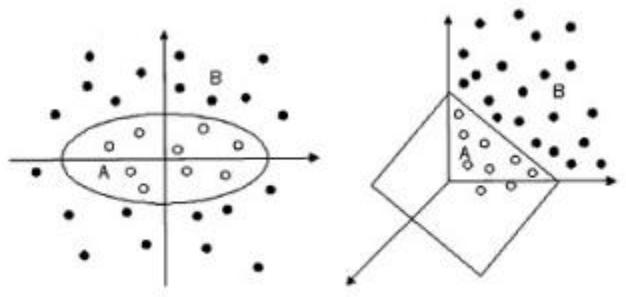


Figure 3: Mapping of non-linear separable training data from $\mathbb{R}^2$ into $\mathbb{R}^3$

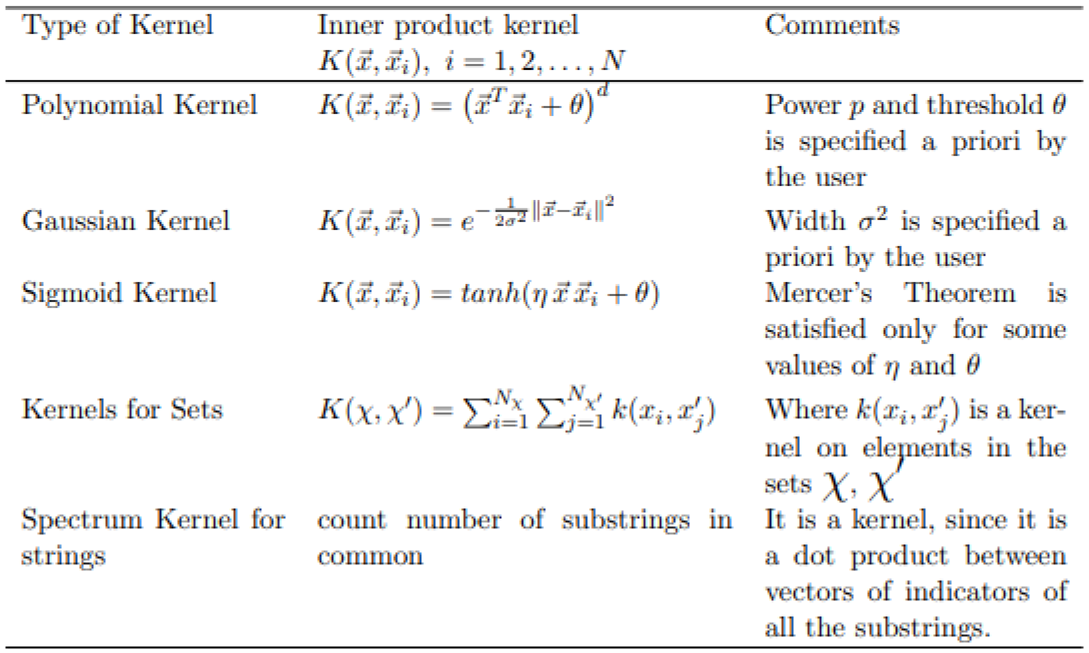
$$\Phi(\vec{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T. \qquad (19)$$

Taking the equation for a separating hyperplane Eq.(1) into account we get a linear function in $\mathbb{R}^3$:

$$\vec{w}^T\Phi(\vec{x}) = w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 = 0. \qquad (20)$$

SVC Kernels

- Sometimes data isn't linearly separable
  - Use kernel function to map to linearly separable feature space



Type of Kernel	Inner product kernel $K(\vec{x}, \vec{x}_i),\ i = 1, 2, \ldots, N$	Comments
Polynomial Kernel	$K(\vec{x}, \vec{x}_i) = \left(\vec{x}^T \vec{x}_i + \theta\right)^d$	Power p and threshold θ is specified a priori by the user
Gaussian Kernel	$K(\vec{x}, \vec{x}_i) = e^{-\frac{1}{2\sigma^2}\lVert\vec{x}-\vec{x}_i\rVert^2}$	Width σ^2 is specified a priori by the user
Sigmoid Kernel	$K(\vec{x}, \vec{x}_i) = tanh(\eta\, \vec{x}\, \vec{x}_i + \theta)$	Mercer's Theorem is satisfied only for some values of η and θ
Kernels for Sets	$K(\chi, \chi') = \sum_{i=1}^{N_\chi} \sum_{j=1}^{N_{\chi'}} k(x_i, x'_j)$	Where $k(x_i, x'_j)$ is a kernel on elements in the sets χ, χ'
Spectrum Kernel for strings	count number of substrings in common	It is a kernel, since it is a dot product between vectors of indicators of all the substrings.

Table 1: Summary of Inner-Product Kernels [Hay98]

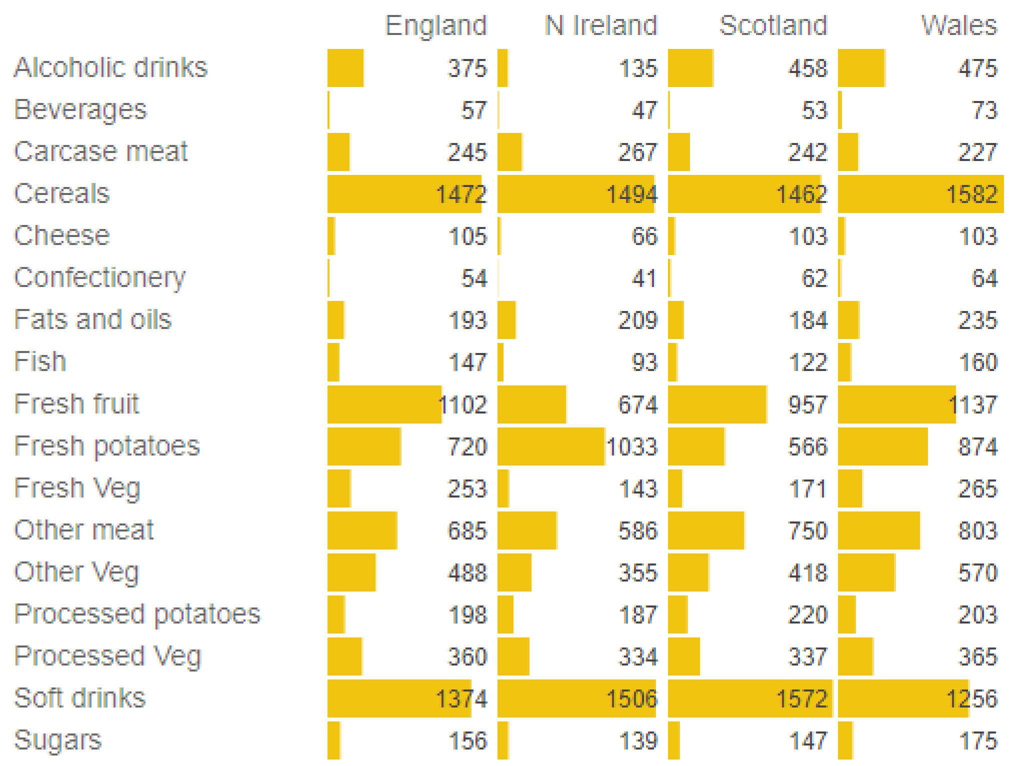
SVC Notebook

- Jupyter Notebook SVC_Example.ipynb has examples of SVC

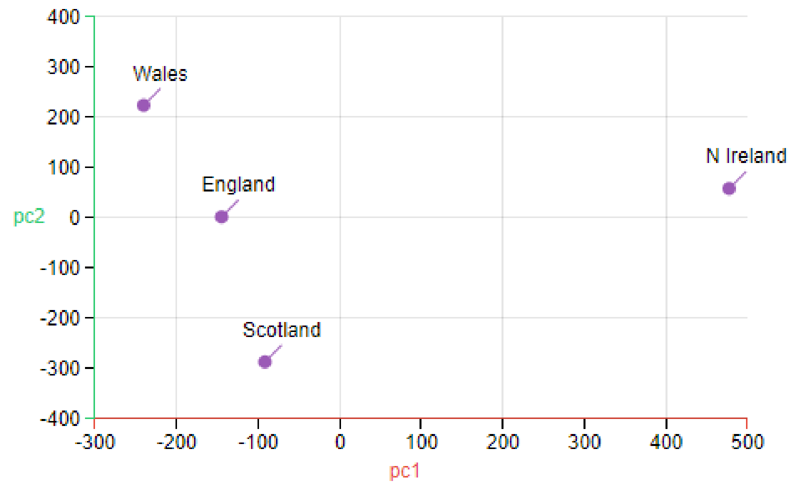
Principal Component Analysis

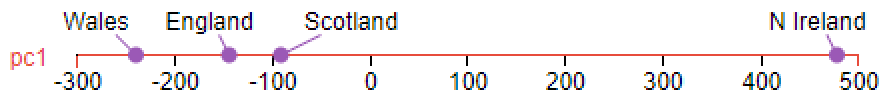
- Fast and flexible way to reduce dimensionality of data (e.g. our faces)
  - Computes eigenvectors of the data’s covariance matrix

Principal Component Analysis



	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175





PCA Notebook

- Jupyter Notebook PCA_Example.ipynb has example of PCA, and the same data with an ISOMAP (http://www-clmc.usc.edu/publications/T/tenenbaum-Science2000.pdf)

Bayesian Classifier

- Bayes Theorem $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$
- If we have a bunch of independent Ys, then:
  - $P(Class|Y1, Y2, \ldots Yn) \propto P(Class) \prod_{i=1}^{n} P(Yi|Class)$
- So we guess a class by just picking the biggest probability!

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Naïve Bayesian Classifier

$$P(Class|Y1, Y2, \ldots Yn) \propto P(Class) \prod_{i=1}^{n} P(Yi|Class)$$

- Need a probability distribution to compute $P(Yi|Class)$
  - Gaussian, compute mean and variance for each class

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Multinomial Bayesian Classifier

- Use multinomial distribution instead, useful for data with “counts” (e.g. word counts in text)

Naïve Bayesian Summary

- Good for well-separated categories
- Good for high dimensional data
- Good when naïve assumptions match (independence, distributions)
- Create fast, explainable models

Multinomial Bayesian Notebook

- Jupyter notebook Multinomial_Naive_Bayes.ipynb with Multinomial Naïve Bayes

Scikit TF-IDF

$$\text{TF}_{(w,j)} = \frac{\text{(Number of times term w appears in a document)}}{\text{(Total number of terms w in the document)}}$$

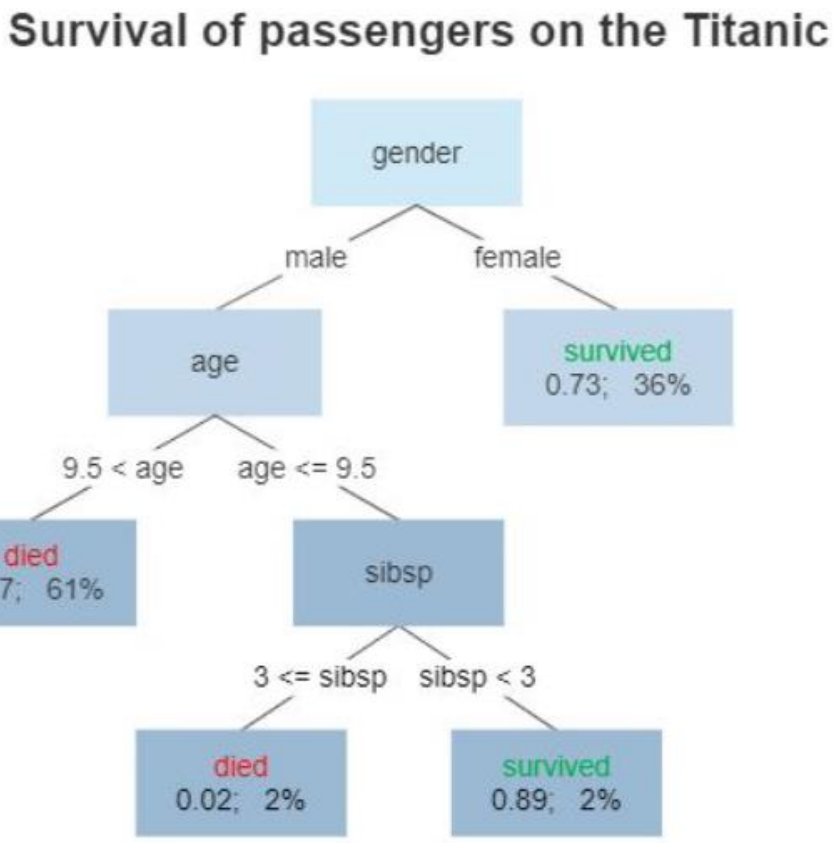
$$\text{IDF}_{(w)} = \log \frac{\text{Total number of documents}}{\text{Number of documents with term w in it}}$$

Scikit-Learn

- $\text{IDF}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

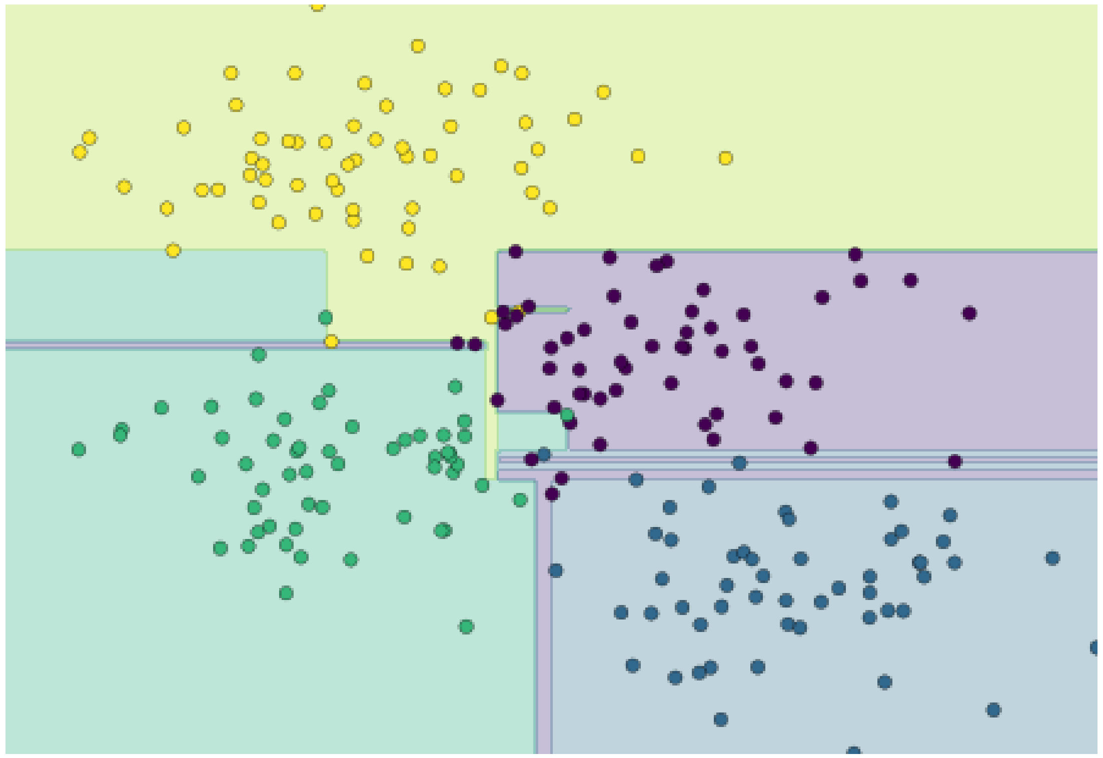
Decision Trees

- Repeatedly split space with hyperplane



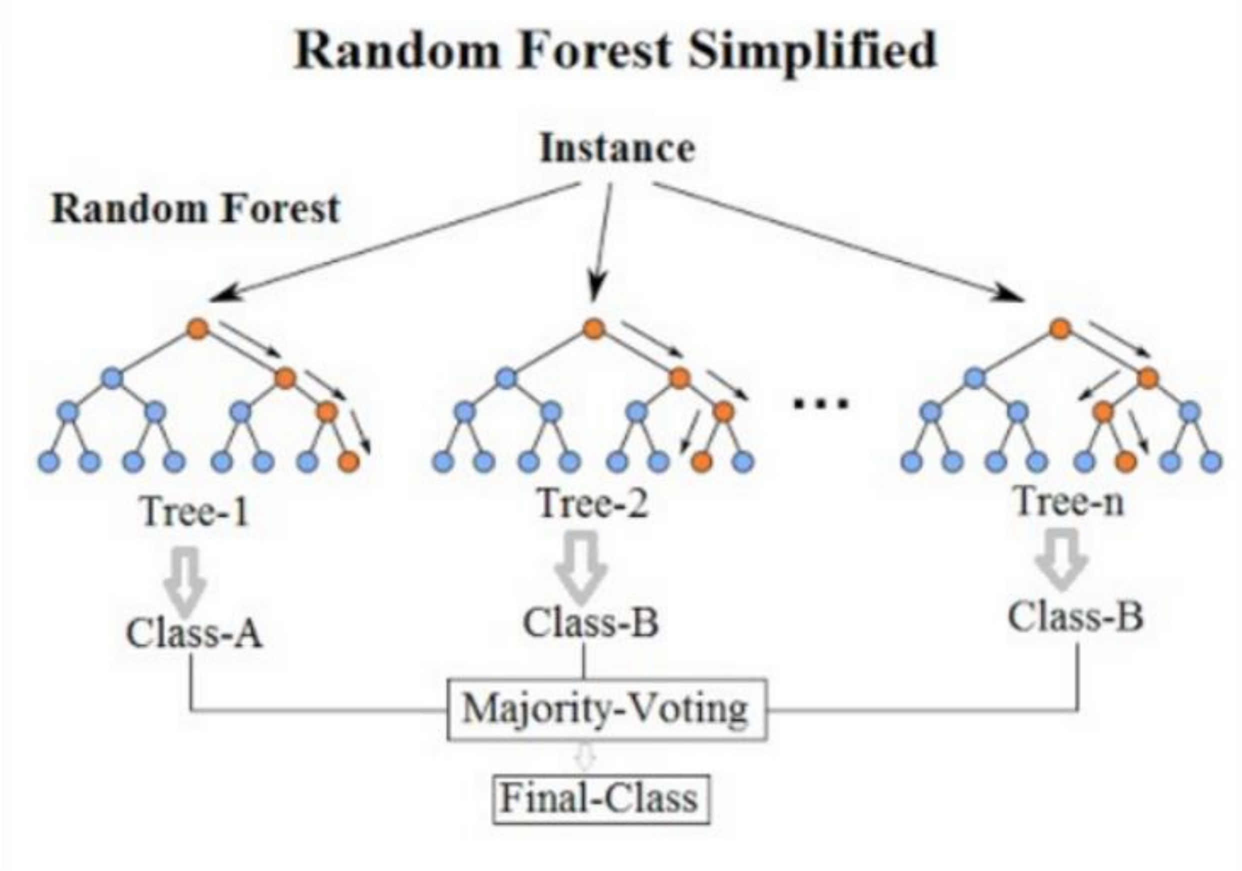
Decision Tree

- Not usually so easily explainable, may get strange behavior of automatically generated



Random Forests

Create many trees to reduce such oddities - every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features



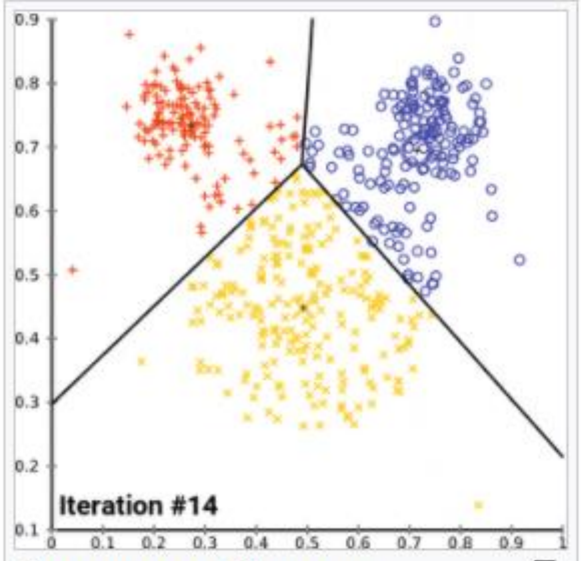
Random Forests

- Revisit digits in Jupyter Notebook Random_Forests.ipynb

K-Means Clustering

- Partition n observations into k clusters each belonging to nearest mean

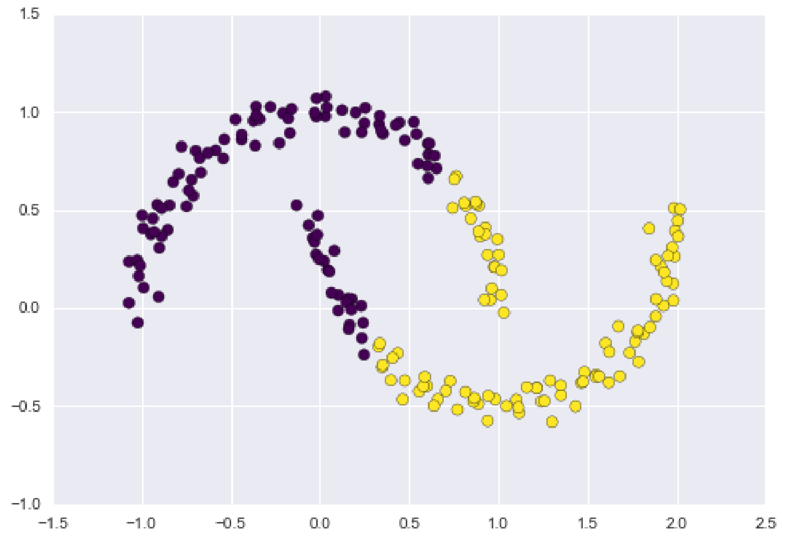
1. Select k “means”
2. Partition
3. Update means
4. Repeat 1-3 until converge



Convergence of *k*-means

K-Means Clustering

- Not necessarily optimal (depends on selection of initial “means”)
- Must know # of clusters in advance
- Might also require mapping to new space

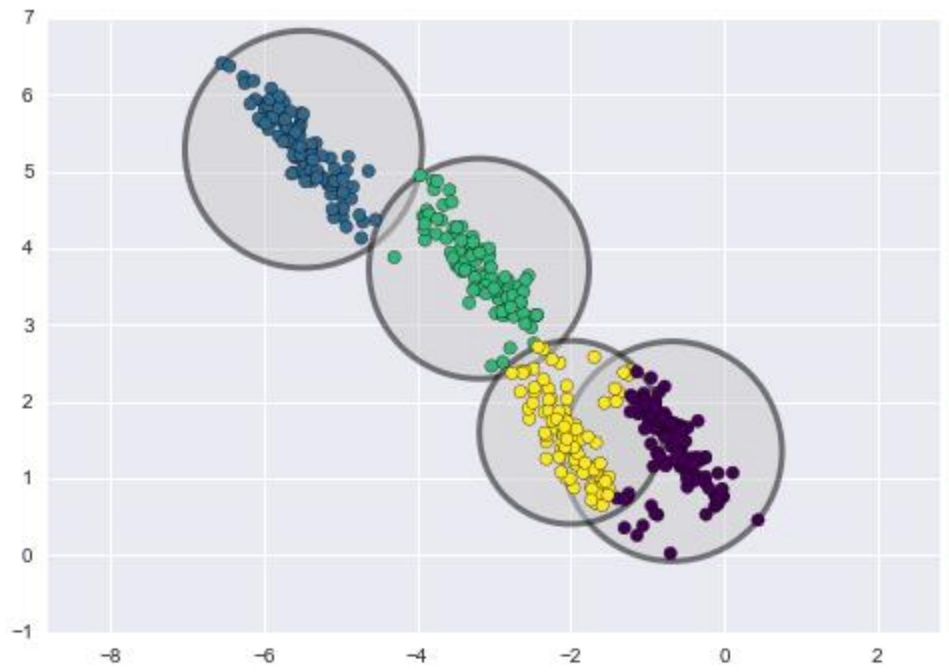


K-Means Clustering

- Revisit digits in Jupyter Notebook K_Means_Clustering.ipynb
- Uses t-distributed Stochastic Neighbor Embedding (TSNE) to visualize high-dimensional data
  - converts affinities of data points to probabilities (Gaussian joint probabilities)

Gaussian Mixture Models

- Extends k-means
  - – K-means has a lack of flexibility in cluster shape

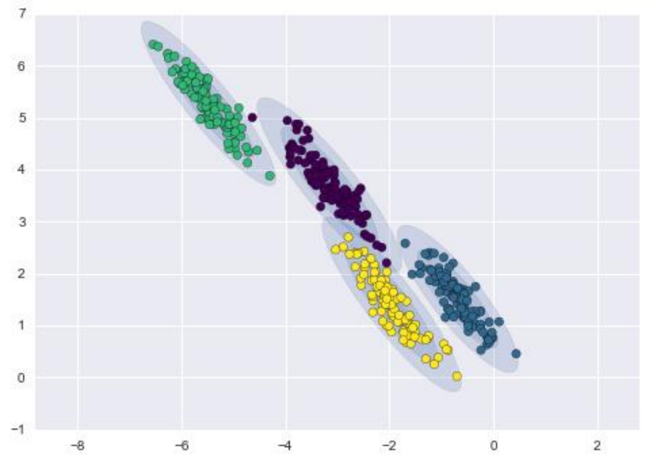


Gaussian Mixture Models

- Extends k-means
  - K-means has a lack of flexibility in cluster shape
  - K-means lacks probabilistic cluster assignment

Gaussian Mixture Models

1. Choose starting "means"
2. Repeat until convergence
   - For each point, find probability in each cluster
   - For each cluster, update location and shape based on all data points, using weights



Gaussian Mixture Modeling

- Revisit digits in Jupyter Notebook Gaussian_Mixture_Model.ipynb, now using Kernel Density Estimation
  - Uses mixture of one Gaussian per point

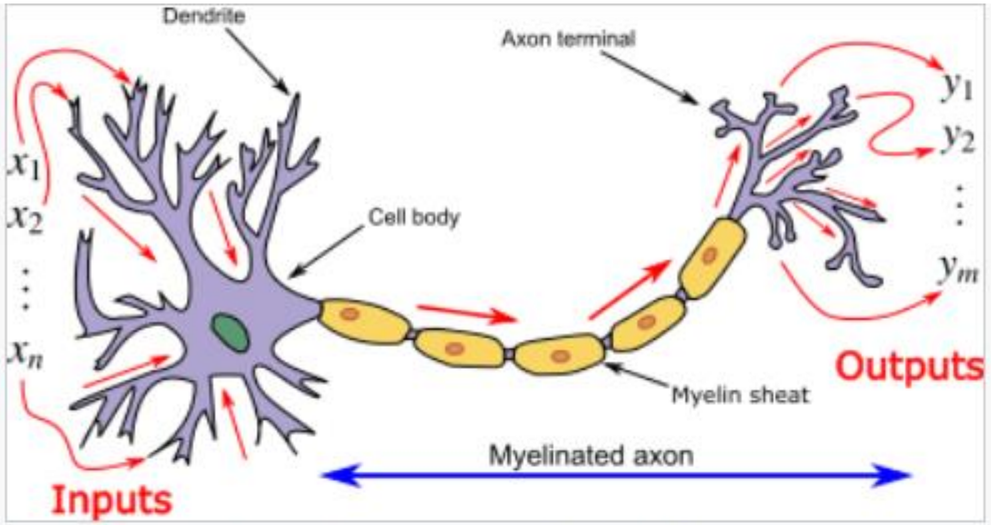
COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Neural Networks

Dr. Martin “Doc” Carlisle

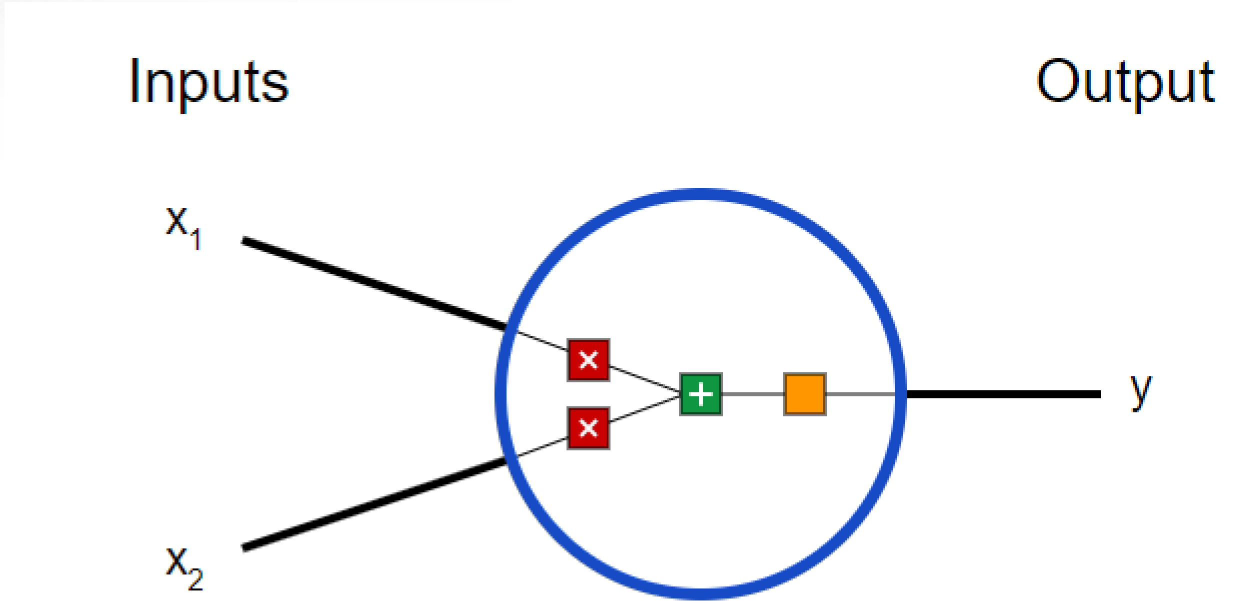
Neural Networks

Neural Networks are a machine learning technique fashioned after a mathematical model of a brain neuron



Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals

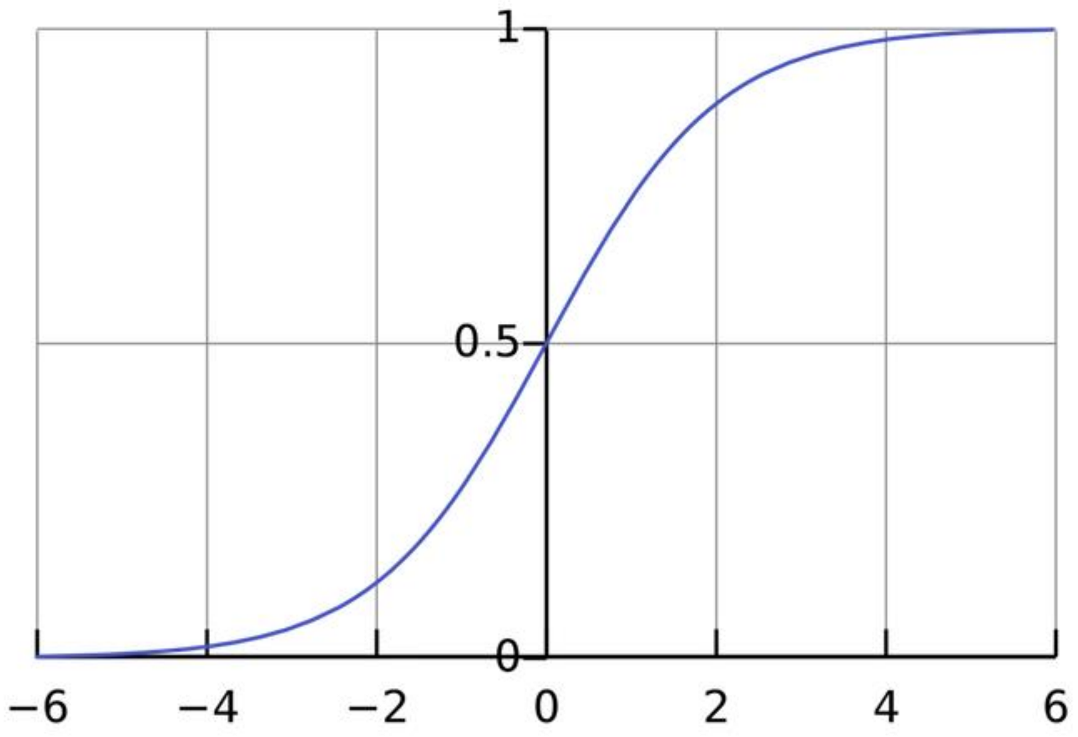
A “neuron”



Red multiplied by weights
Orange is an “activation function”

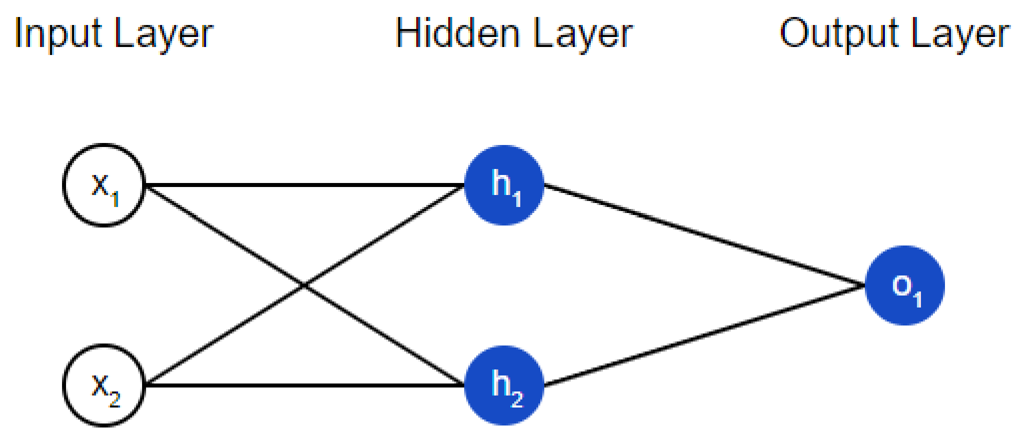
Activation function

- One example is sigmoid
  - f(x) = $\frac{1}{1+e^{-x}}$



Creating a “network”

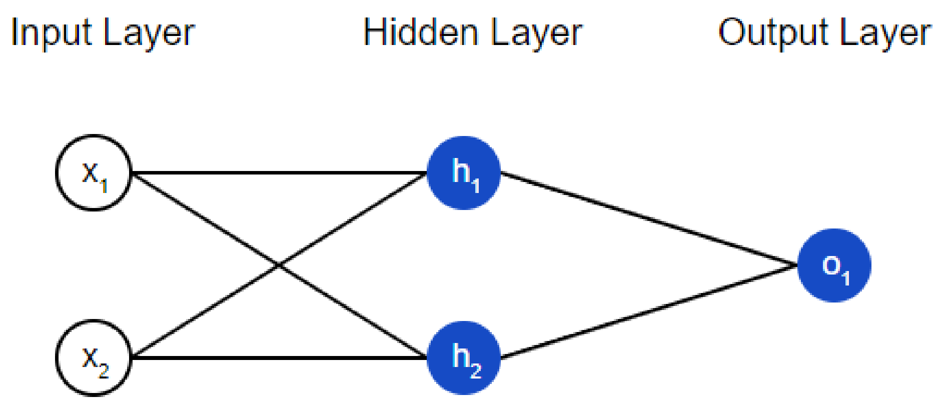
- A multi-layer architecture of neurons



- Possibly many hidden layers

At a neuron

- Weights for each input

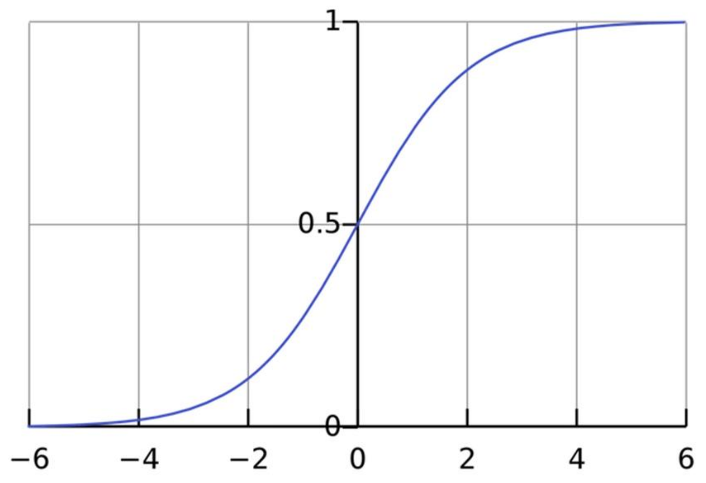


- H_1=sigmoid(w_{11}*x_1+w_{12}*x_2)
- H_2=sigmoid(w_{21}*x1+w_{22}*x_2)
- O_1=sigmoid(w_{o1}*h_1+w_{o2}*h_2)

Output layer

- We can have a neuron for each category, and we want 1 for yes, 0 for no
- Now we can go backwards with partial derivatives to update weights using the chain rule

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$



Updating weights

- We then update weights with a “learning rate”

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

Good and bad news

- Good news! We don’t need to know anything about partial differentials – pytorch will do this all for us

- Bad news! Picking a correct architecture, learning rate can be hard (and is a huge search space)

Speeding things up with a GPU

- GPUs offer data-parallelism
  - This can make NN operations *much* faster
- BUT
  - You have to explicitly put things on the GPU
  - Copying to/from the GPU is expensive

PyTorch

Useful for doing neural networks on a GPU

Tensor = NumPy ndarray, but can be on a GPU
Device = place a tensor lives (CPU, or CUDA [GPU])

WARNING!

- Copying an array to/from the GPU is expensive!
- You have to think about where you want computation to happen!

Jupyter Notebook

Go to Pytorch_Intro.ipynb at Google colab

https://colab.research.google.com/

Sample NN

- See Colab notebook for First_NN

Convolutional Neural Nets

- Aim for shift, scale and distortion invariance
  - Local receptive fields
  - Shared weights
  - Sub-sampling

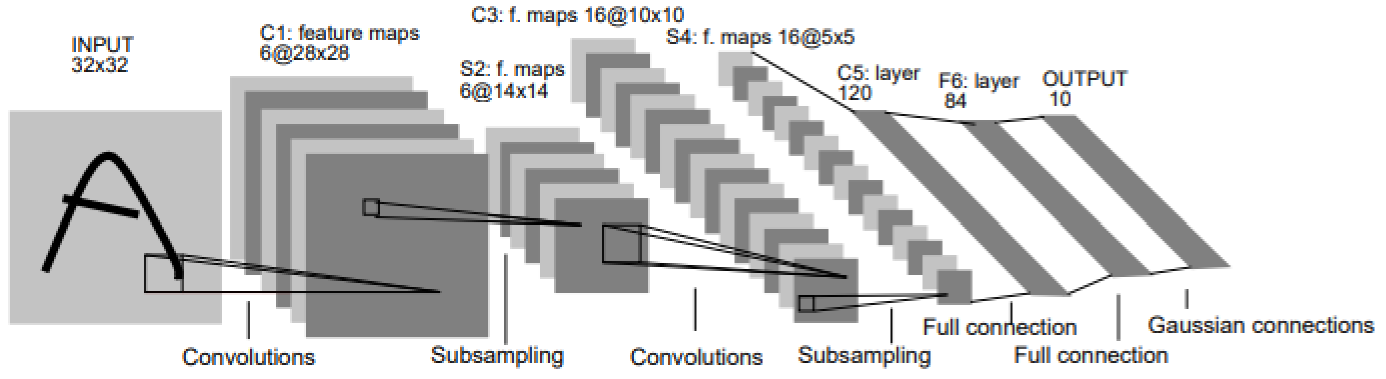


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Local receptive fields

- Extract elementary visual features
  - Oriented edges
  - End points
  - Corners
- Also, avoid explosion of weights! (224x224 image with 3 colors has 150,528 features)

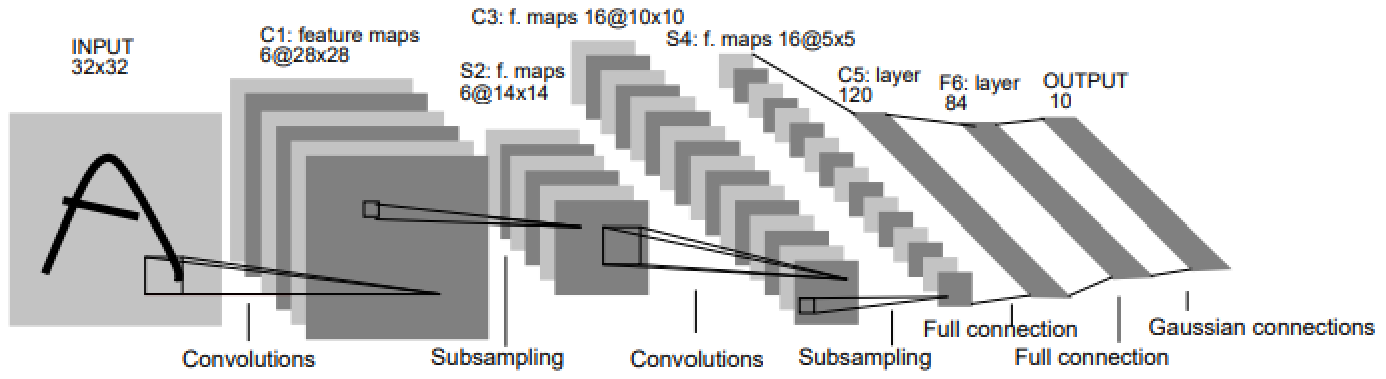


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Shared weights

- Detect same features at all possible locations in the input

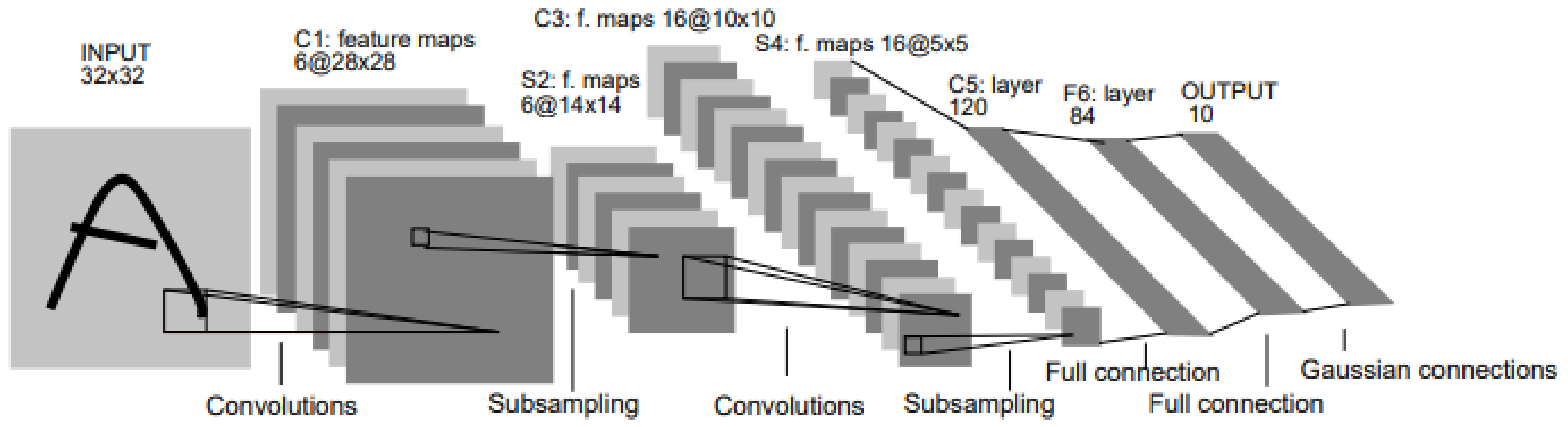


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Subsampling

- Reduces sensitivity to distortion
- Finds features relatively placed

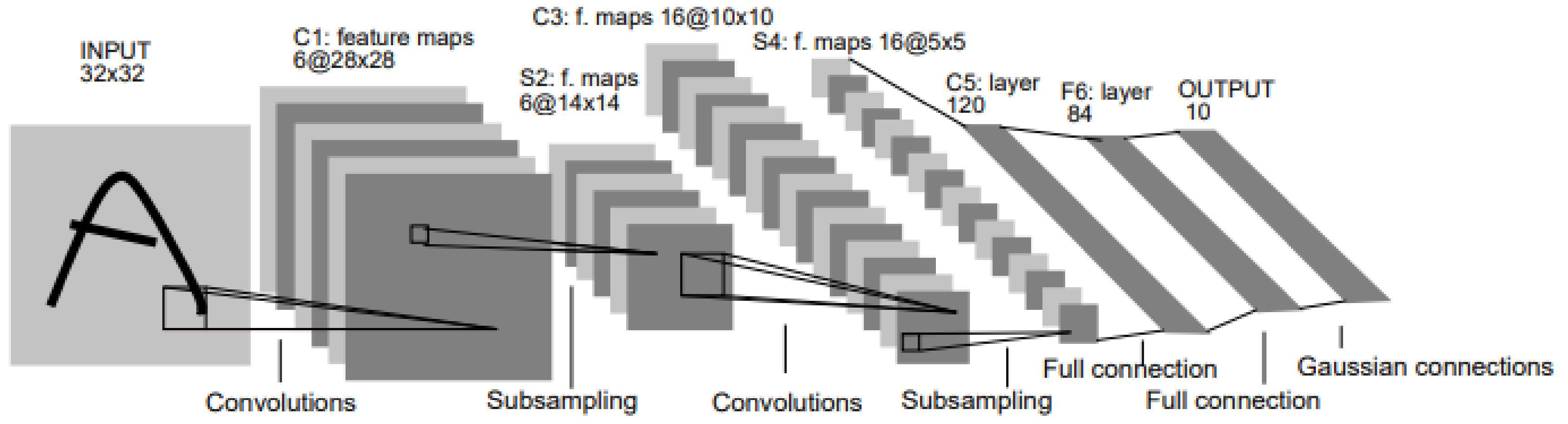


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Jupyter Notebook for CNN

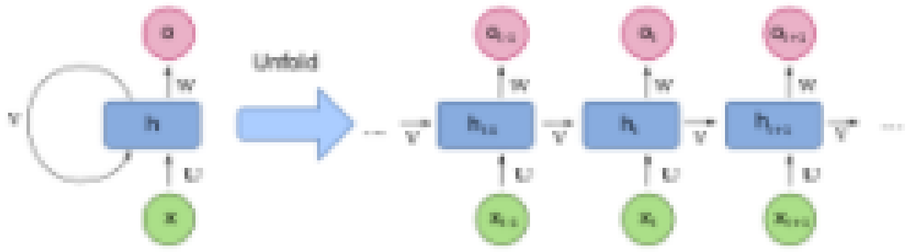
- https://github.com/erykml/medium_articles/blob/master/Computer%20Vision/lenet5_pytorch.ipynb
- lenet5_pytorch.ipynb (Google Colab and Jupyter Notebook)
- Note speed difference
- Note use of torch.no_grad()
- conda install torchvision -c pytorch

Train/Validate/Test

- Ideally, we split data into 3 parts
- Train
  - Use this to train the model and update weights
- Validate
  - Use this to prevent over-training
  - Don’t train, but evaluate hyper-parameters (learning rate, # of epochs, etc.)
- Test
  - Use only on final model run

Recurrent Neural Network

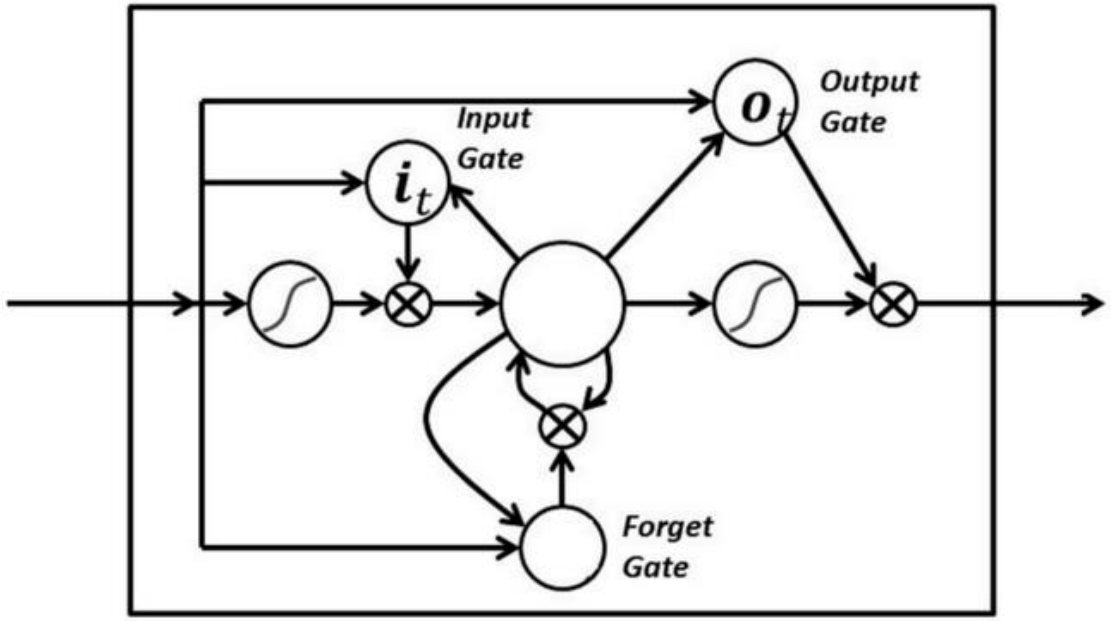
- Connections have temporal sequence
- Useful for applications where context is useful for prediction
  - Handwriting recognition (unlike zipcodes, we have a good sense what comes “afte”)
  - Speech recognition



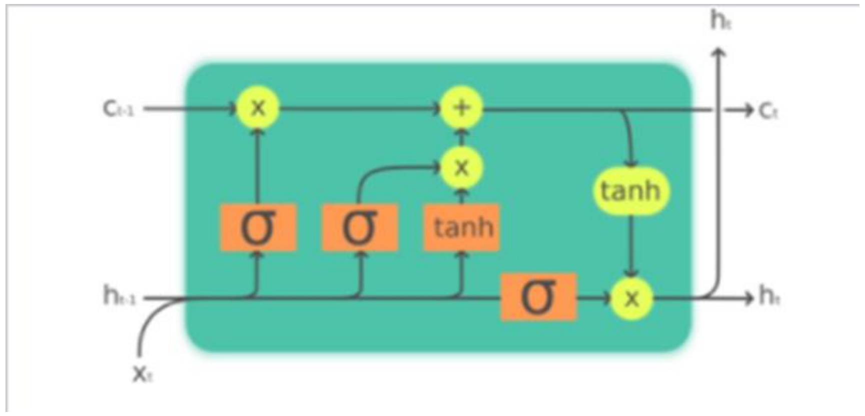
Long short-term memory (LSTM)

- “Forgets” part of previously stored memory and adds new data
- Cell gets cell and hidden state from previous timestep, input from current timestep
- Output is part of hidden state

LSTM



Simple LSTM cell (source: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5)



Example

- Password generator
  - conditional-char-rnn.ipynb

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Password Guessing with Neural Networks

Dr. Martin “Doc” Carlisle

Modeling Password Guessing

How can we help users pick good passwords?

- Simulate password-guessing algorithms
- Reject passwords that will be guessed quickly

Modeling Password Guessing

Issue:

- We want to do this fast and without using lots of storage
- Ideally in a browser

PCFGs

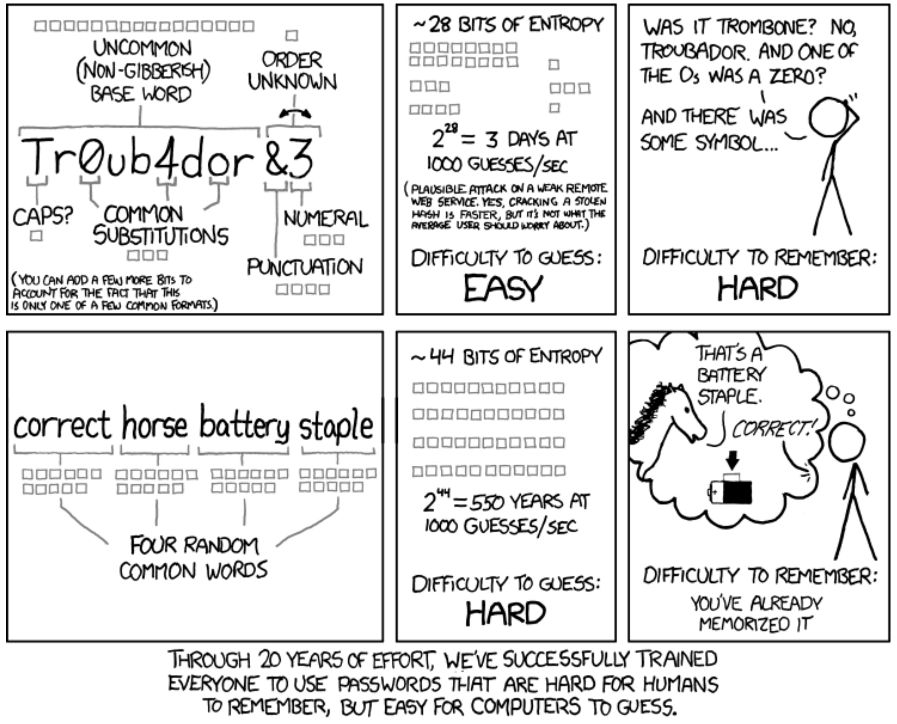
- Use context free grammars to model passwords
  - Template structure (e.g. 6 letters then 2 digits)
  - Pick probability of structure and the probability of each terminal symbol

Markov Models

- Predict probability of next character based on previous characters
  - 6-gram with additive smoothing

Mangled Word Lists

- Think “leetifying” a password
  - E.g. P@ssw0rd
- Maybe adding digits, etc.


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Neural Network guesser

Similar to Markov model, predict next character based on context

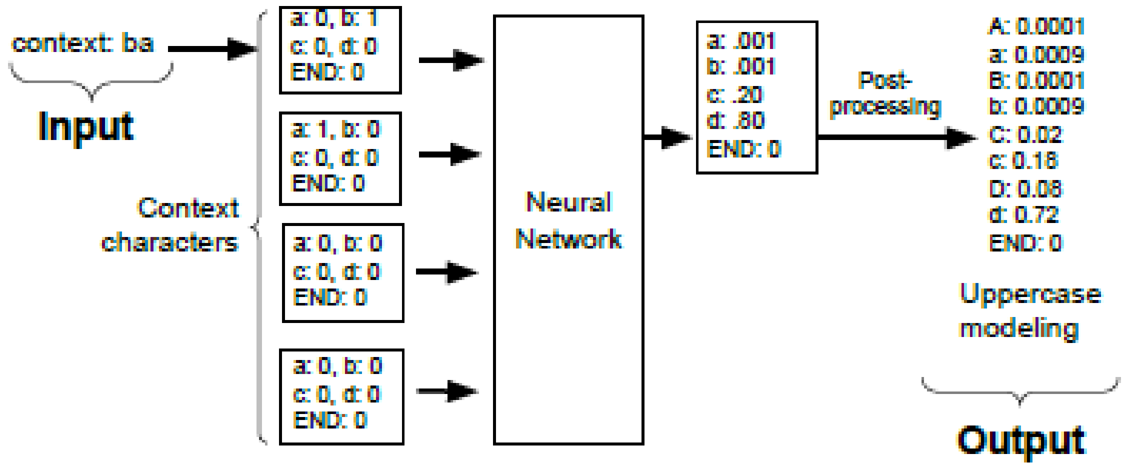


Figure 1: **An example of using a neural network to predict the next character of a password fragment.** The network is being used to predict a 'd' given the context 'ba'. This network uses four characters of context. The probabilities of each next character are the output of the network. Post processing on the network can infer probabilities of uppercase characters.

Turn probability to guess

- Enumerate all passwords who probability is above threshold via beam search
- Use Monte Carlo simulations to guess

Beam Search

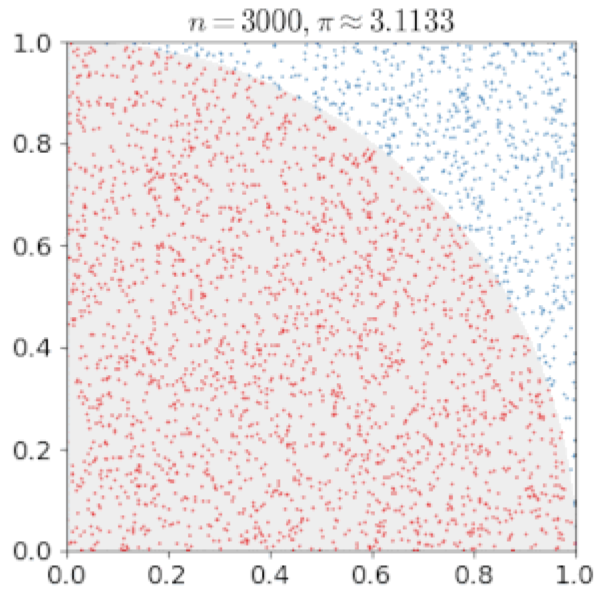
- Expand most promising node
  - Optimization of best-first search to reduce memory use
  - Greedy algorithm
  - Not optimal

Monte Carlo Simulations

- Monte Carlo methods vary, but tend to follow a particular pattern:
  - Define a domain of possible inputs
  - Generate inputs randomly from a probability distribution over the domain
  - Perform a deterministic computation on the inputs
  - Aggregate the results

Estimating Pi

- Draw a square, then inscribe a quadrant within it
- Uniformly scatter a given number of points over the square
- Count the number of points inside the quadrant, i.e. having a distance from the origin of less than 1
- The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, $\pi/4$. Multiply the result by 4 to estimate π.



Design Choices

- 1000 LSTM cells, Recurrent Neural Network
  - 1.5 weeks to train on larger data set (2016)
- Train in backwards order
  - Guess “d” from “rowssap” vs. “passwor”
- With 10 characters of context
- Model 2000 tokens (letters, syllables)
  - Model upper vs. lower separately

Design Choices

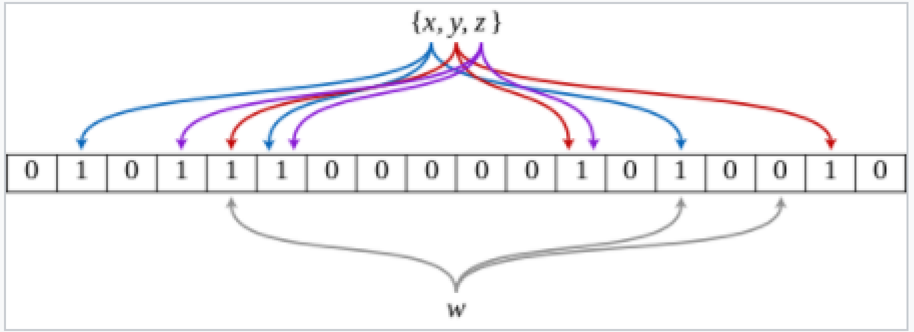
- Transference learning
  - Train with all passwords in set, then freeze lower levels of model
  - Retrain with only passwords that fit pw policy

Sending to browser!

- Quantize weights (up to 3 decimal digits)
- Fixed point encoding (save space)
  - 5.0 and 5.0 with precision of 0.005 is -1000 to 1000 with precision of 1
- ZigZag encoding 0,-1,1,-2,2,….
- Lossless compression
- Bloom filter word list (2 million most frequent passwords)
  - Top 10, top 100, etc.

Bloom filters

- Space-efficient probabilistic data structure
- Tests set membership
  - Can have false positives, but not false negs
- Hash several times with k independent hashes
  - If all 1s, is in there!
- 9.6 bits/element



An example of a Bloom filter, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, $m = 18$ and $k = 3$.

Training data

- Password Guessability Service
  - RockYou! & Yahoo leaked passwords
  - Web2, Google web, inflection dictionary
  - 33 million passwords, 5.9m natural language words
- PGS++
  - 105 million passwords

Testing Data

- Mechanical Turk
- 000webhost

Results

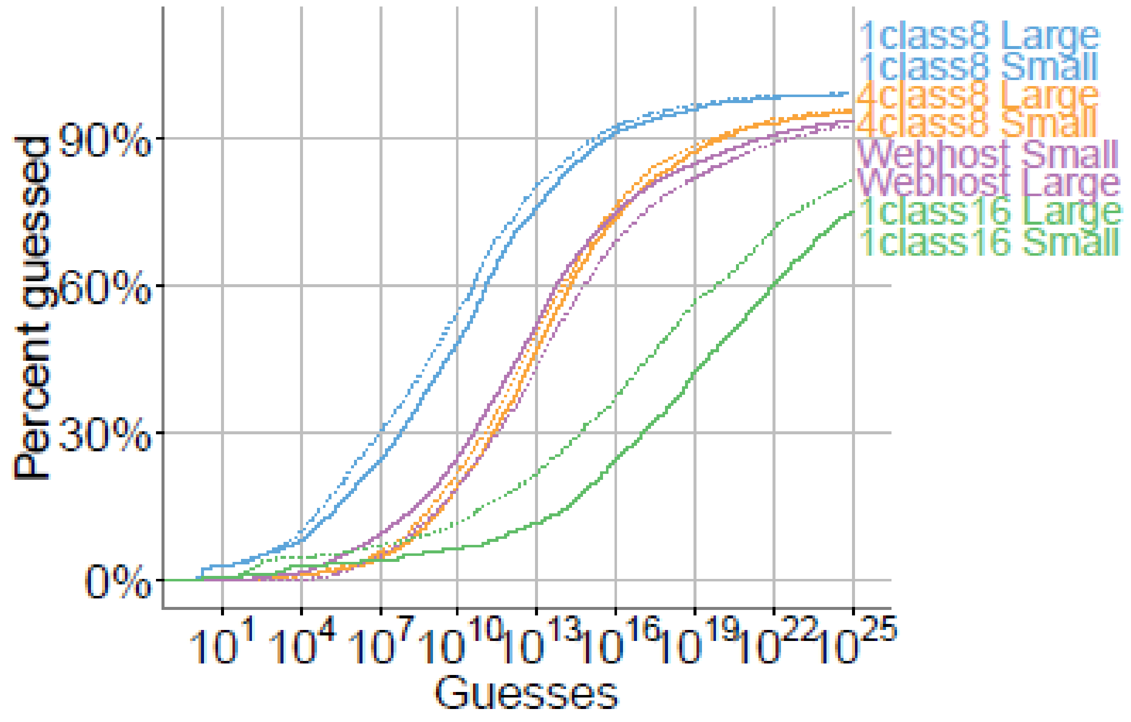
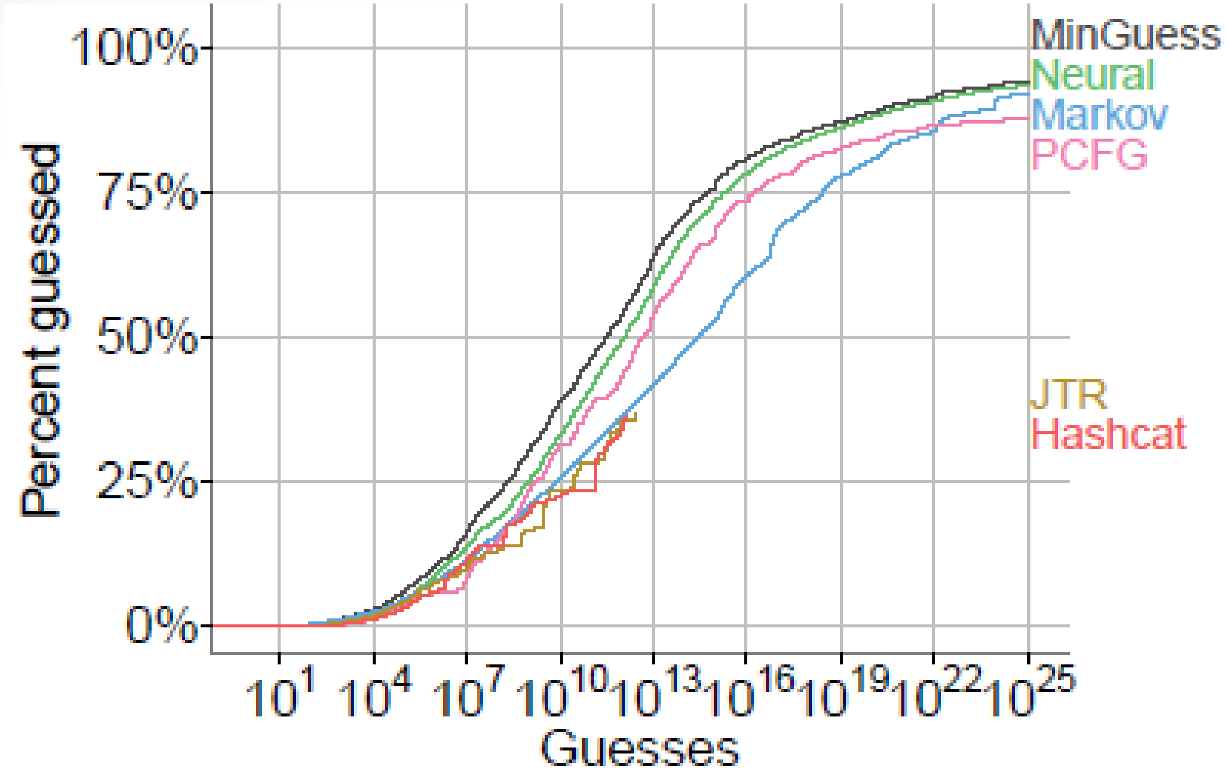


Figure 3: **Neural network size and password guessability.** Dotted lines are large networks; solid lines are small networks.

Results



(b) Webhost passwords

de Castro et al

- Example of a nice undergrad student project
  - Implements Mellicher et al with some simplifications
  - Shows understanding of LSTM, Monte Carlo simulation
  - Performs an experiment and discusses results
  - Compares their results to ZXCVBN