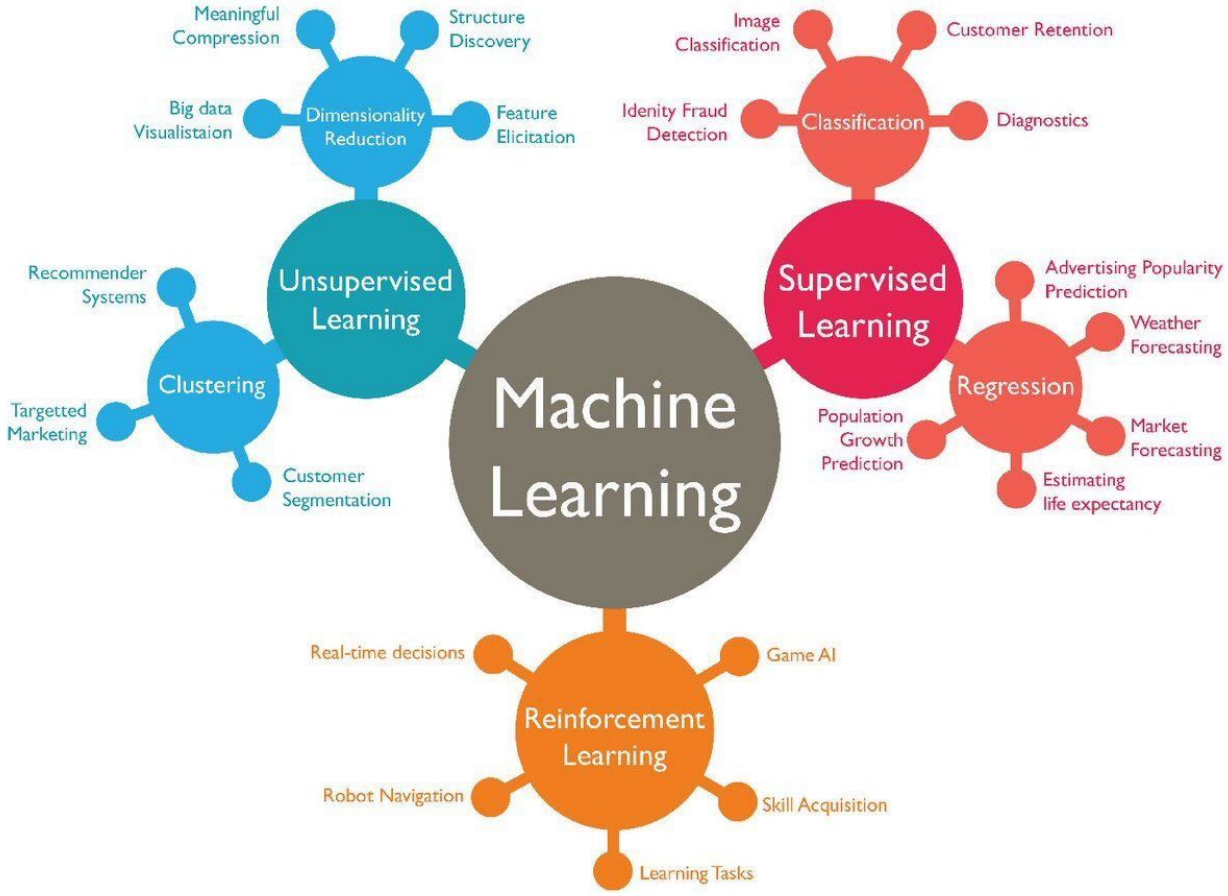
Machine Learning and Deep Learning Notes

Welcome, the following is an in depth personal notebook that I made as a quick reference to several topics in the fields of machine learning and [subfield] deep learning.

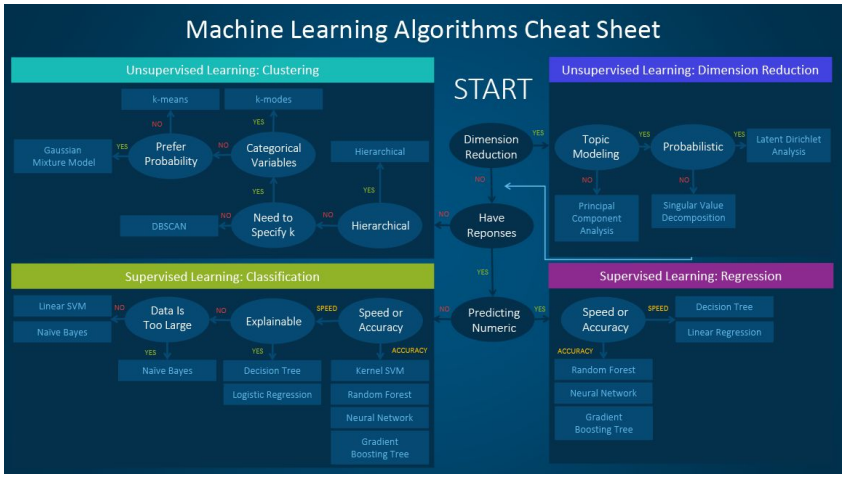
Please keep in mind that this is a work in progress, which has already grown to ~130 pages on several topics. The notebook is open for comments, please don't abuse the platform.

Many Thanks,
Ori Cohen, Ph.d.

TYPES OF MACHINE LEARNING



[A wonderful introduction into machine learning, and how to choose the right algorithm or family of algorithms for the task at hand.]



BIAS / VARIANCE

Understanding what is the next stage in DL (& ML) algorithm development: basic approach - Andrew NG on youtube

Terms: training, validation, test.
Split: training & validation 70%, test 30%
Procedure: crossfold training and validation, or further split 70% to training and validation.

BIAS - Situation 1 - doing much worse than human:
Human expert: 1% error
Training set error: 5% error (test on train)
Validation set error: 6% error (test on validation or CFV)
Conclusion: there is a BIAS between human expert and training set
Solution: 1. Train deeper or bigger\larger networks, 2. train longer, 3. May needs more data to get to the human expert level, Or 4. New model architecture.

VARIANCE - Situation 2 - validation set not close to training set error:
Human expert: 1% error
Training set error: 2% error
Validation set error: 6% error
Conclusion: there is a VARIANCE problem, i.e. OVERFITTING, between training and validation.
Solution: 1. Early stopping, 2. Regularization or 3. get more data, or 4. New model architecture.

Situation 3 - both:
Human expert: 1% error
Training set error: 5% error
Validation set error: 10% error
Conclusion: both problems occur, i.e., BIAS as and VARIANCE .
Solution: do it all.

Underfitting = Get more data
Overfitting = Early stop, regularization, reason: models detail & noise.

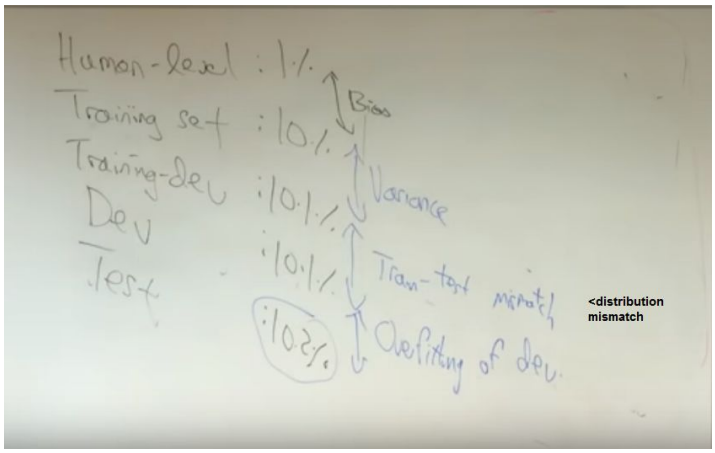
- Happens more in non parametric (and non linear) algorithms such as decision trees.

Bottom line, bigger model or more data will solve most issues.

+ In practice advice with regularized linear regression.

IMPORTANT! For Test Train efficiency when the data is from different distributions:
E.g: **TRAIN**: 50K hours of voice chatter as the train set for a DLN, **TEST:** 10H for specific voice-based problem, i.e, taxi chatter.
Best practice: better to divide the validation & test from the same distribution, i.e. the 10H set.
Reason: improving scores on validation which is from a diff distribution will not be the same quality as improving scores on a validation set originated from the actual distribution of the problem's data, i.e., 10H.
NOTE: Unlike the usual supervised learning, where all the data is from the same distribution, where we split the training to train and validation (cfv).

Situation 4: However, when there are 2 distributions it's possible to extend the division of the training set to validation_training and training, and the test to validation and test.
Split: Train, Valid_Train = 48K\2K & Valid, Test, 5K & 5K.



So situation 1 stays the same,
Situation 2 is Valid_Train error (train_dev)
Situation 3 is Valid_Test error - need more data, **data synthesis** - tweak test to be similar to train data, new architecture as a solution
Situation 4 is now Test set error - get more data

Train Test methodology -

“The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a “vault,” and be brought out only at the end of the data analysis”

- Random Split tests 66\33 - problem: variance each time we rerun.
- Multiple times random split tests - problem: samples may not be included in train\test or selected multiple times.
- Cross validation - pretty good, diff random seed results in diff mean accuracy, variance due to randomness
- Multiple cross validation - accounts for the randomness of the CV
- Statistical significance (t-test) on multi CV - are two samples drawn from the same population? (no difference). If “yes”, not significant, even if the mean and std deviations differ.

Finally, When in doubt, use k-fold cross validation (k=10) and use multiple runs of k-fold cross validation with statistical significance tests.

NORMALIZATION / SCALING

1. A comparison of normalization / scaling techniques in sklearn
2. Another great explanation on sklearn and (general) scaling - normal, min max, etc..
3. Normalization\standardize features
    - data has varying scales
    - Normalize between range 0 to 1.
        - When the algorithm you are using **does not** make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.
    - Standardize, mean of 0 and a std of 1:
        - When the algorithm assumes a gaussian dist, such as linear regression, logistic regression and linear discriminant analysis. LR, LogR, LDA

    **Generally, it is a good idea to standardize data that has a Gaussian (bell curve) distribution and normalize otherwise.

4. In general terms, we should test 0,1 or -1,1 empirically and possibly match the range to the NN gates/activation function etc.

Variables in statistics for regression (also **GLM** method):

Discrete

- **Numbers**
- **Categorical**
    - **Categorical** data are variables that contain label values rather than numeric values.
      The number of possible values is often limited to a fixed set.
    - **Categorical** variables are often called nominal.
    - labels, usually discrete values such as gender, country of origin, marital status, high-school graduate

Continuous (the opposite of discrete**):** real-number values, measured on a continuous scale: height, weight.

In order to compute a regression, categorical predictors must be re-expressed as numeric: some form of indicator variables (0/1) with a separate indicator for each level of the factor.
Discrete with many values are often treated as continuous, i.e. zone numbers - > binary

Variable types: Nominal(weather), ordinal(order var 1,2,3), interval(range),

NYC taxi pickup problem

The taxi problem **is an intro to a well known machine learning problem**, the paper will explain about feature engineering, analysis and using various regression algorithms for the purpose of solving the problem, you can use this as a base for many regression and classification problems.

A **Second study** (regression, random forest, xgboost (extreme gradient boosting tree)). Standard error estimate -- measures the distance from the estimated value to the real value R^2 error estimate- measures the distance of the estimated to the mean against the real to the mean, 1 no error, 0 lots.

**** with regression prediction it's best to create **dummy variables (i.e., binary variables - exist or doesn't exist)** from numeric variables, such as grid_number to grid_1, grid_2 etc..

SURVIVAL ANALYSIS

1. A good introduction for Survival Analysis

Structured / Unstructured data

Unstructured
Structured

BENCHMARKING

Datasets:

- EFF FF Benchmarks in AI

Hardware:

- Cpu vs GPU benchmarking for CNN\Test\LTSM\BDLTSM - google and amazon vs gpu
- Nvidia GPUs - titax Xp\1080TI\1070 on googlenet
- March\17 - Nvidia GPUs for desktop, in terms of price and cuda units, the bottom line is 1060-1080.
- Another bench up to 2013 - regarding many GPUS vs CPUs in terms of BW

Platforms

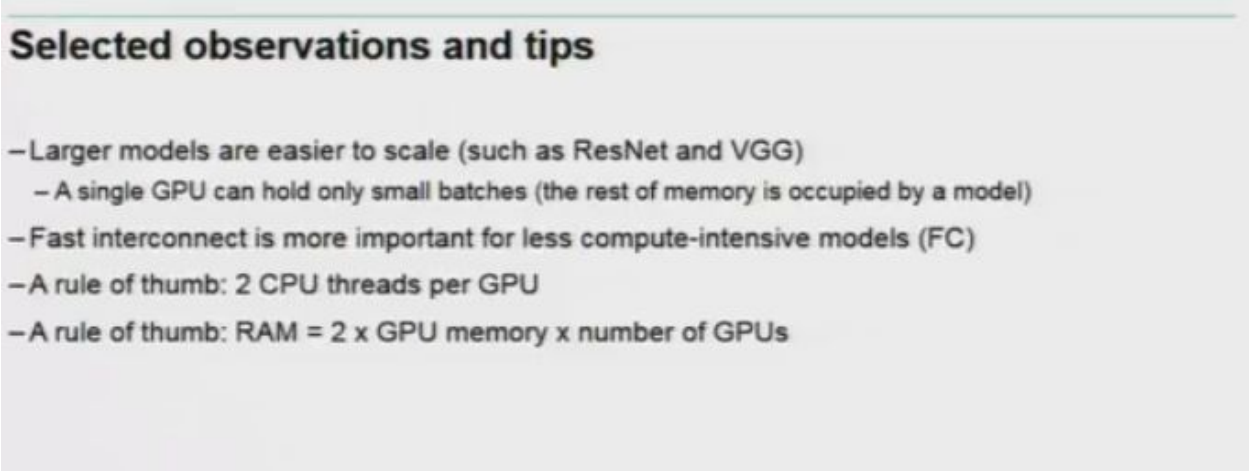
- Cntk vs tensorflow
- CNTK, TEnsor, torch, etc on cpu and gpu

Algorithms:

- Comparing accuracy, speed, memory and 2D visualization of classifiers:
  SVM, k-nearest neighbors, Random Forest, AdaBoost Classifier, Gradient Boosting, Naive, Bayes, LDA, QDA, RBMs, Logistic Regression, RBM + Logistic Regression Classifier
- LSTM vs cuDNN LS1TM - batch size of power 2 matters, the latter is faster.

Scaling networks and predicting performance of NN:

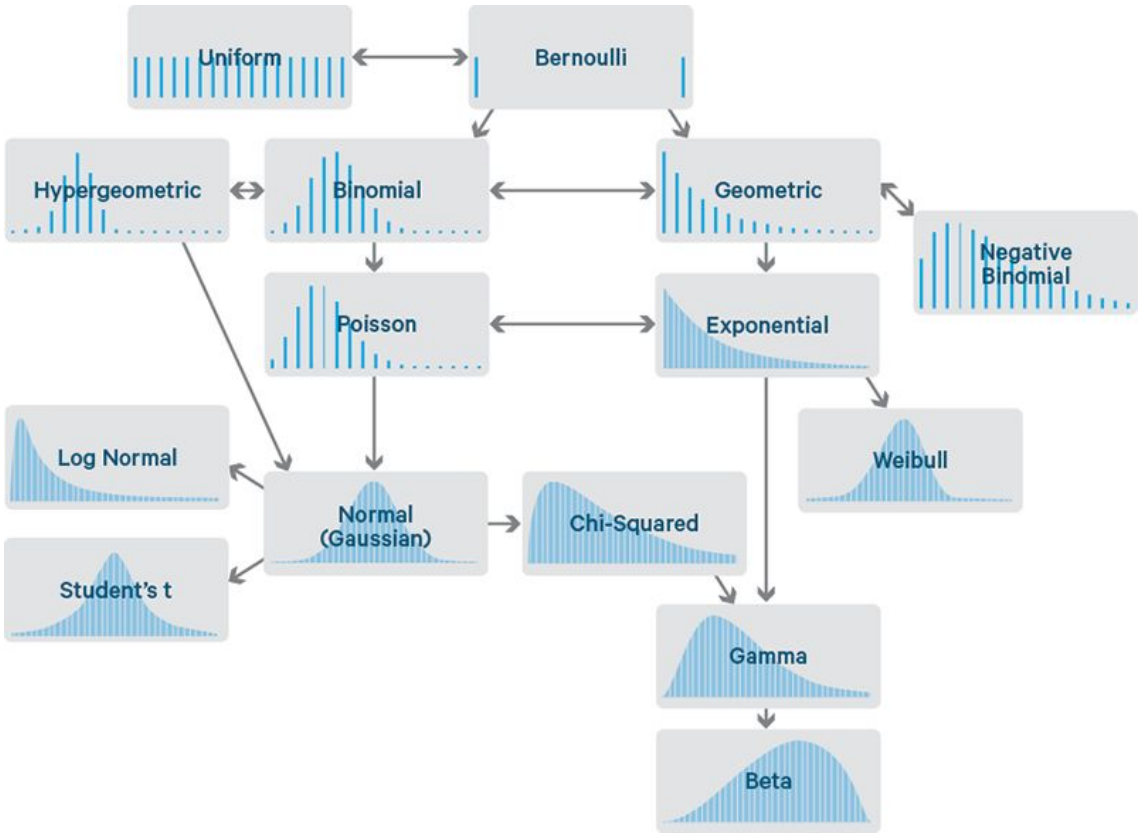
- A great overview of NN types, but the idea behind the video is to create a system that can predict train time and possibly accuracy when scaling networks using multiple GPUs, there is also a nice slide about general hardware recommendations.



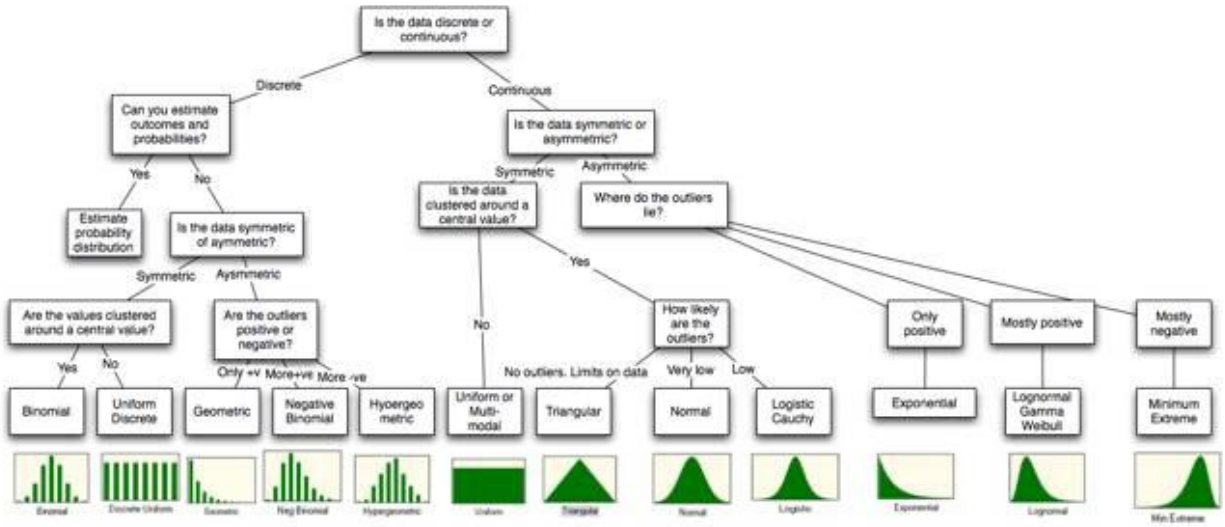
DISTRIBUTION TYPES

(What are?) probabilities in a distribution always add up to 1.

- A very good explanation



- A very wordy explanation (figure2)



Gaussian \ Normal Distribution

" if you collect data and it is not normal, "you need to collect more data"

Beautiful graphs

The normal distribution is popular for two reasons:

1. It is the most common distribution in nature (as distributions go)
2. An enormous number of statistical relationships become clear and tractable if one assumes the normal.

Sure, nothing in real life exactly matches the Normal. But it is uncanny how many things come close.

this is partly due to the Central Limit Theorem, which says that if you average enough unrelated things, you eventually get the Normal.

- the **Normal distribution** in statistics is a special world in which the math is straightforward and **all the parts fit together in a way that is easy to understand and interpret.**
- It may not exactly match the real world, but it is close enough that this one **simplifying assumption allows you to predict lots of things**, and the **predictions are often pretty reasonable.**
- statistically convenient.
- represented by basic statistics
    - **average**
    - **variance** (or standard deviation) - the average of what's left when you take away the average, but to the power of 2.

In a statistical test, you need the data to be normal to guarantee that your p-values are accurate with your given sample size.
If the data are not normal, your sample size may or may not be adequate, and it may be difficult for you to know which is true.

BOX COX

(What is the Box-Cox Power Transformation?)

- a procedure to **identify an appropriate exponent** (Lambda = l) to use **to transform data into a "normal shape."**
- The Lambda value indicates the power to which all data should be raised.

$$T(Y) = (Y^{\lambda} - 1)/\lambda$$

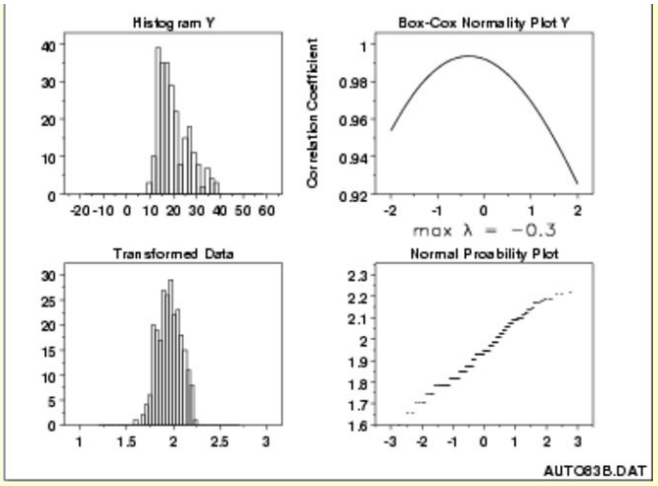
The Box-Cox transformation is a useful family of transformations.

- Many **statistical tests** and intervals are based on the **assumption of normality**.
- The assumption of normality often **leads to tests that are simple**, mathematically tractable, and powerful compared to tests that do not make the normality assumption.
- **Unfortunately, many real data sets are in fact not approximately normal.**
- However, an **appropriate transformation** of a data set can often yield a data set that does follow **approximately a normal distribution**.
- This **increases the applicability and usefulness** of statistical techniques **based on the normality assumption**.

IMPORTANT:!! After a transformation *(c)*, we need to measure of the normality of the resulting transformation (d) .

- One measure is to compute the correlation coefficient of a normal probability plot **=> (d).**
- The correlation is computed between the vertical and horizontal axis variables of the probability plot and is a convenient measure of the linearity of the probability plot
- In other words: **the more linear the probability plot, the better a normal distribution fits the data!**

*NOTE: another useful link that explains it with figures, but i did not read it.



The histogram in the upper left-hand corner shows a data set that has significant right skewness (and so does not follow a normal distribution). The Box-Cox normality plot shows that the maximum value of the correlation coefficient is at λ = -0.3. The histogram of the data after applying the Box-Cox transformation with λ = -0.3 shows a data set for which the normality assumption is reasonable. This is verified with a normal probability plot of the transformed data.

GUARANTEED NORMALITY?

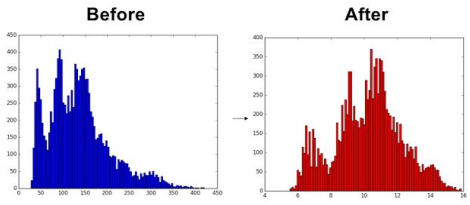
- NO!
- This is because it actually does not really check for normality;
- the method checks for the smallest standard deviation.
- The assumption is that among all transformations with Lambda values between -5 and +5, transformed data has the highest likelihood – but not a guarantee – to be normally distributed when standard deviation is the smallest.
- it is absolutely necessary to always check the transformed data for normality using a probability plot. **(d)**

+ Additionally, the Box-Cox Power transformation only works if all the data is positive and greater than 0.
+ achieved easily by adding a constant 'c' to all data such that it all becomes positive before it is transformed. The transformation equation is then:

COMMON TRANSFORMATION FORMULAS (based on the actual formula)

Common Box-Cox Transformations	
Lambda value (λ)	Transformed data (Y')
-3	$Y^{-3} = 1/Y^3$
-2	$Y^{-2} = 1/Y^2$
-1	$Y^{-1} = 1/Y^1$
-0.5	$Y^{-0.5} = 1/(\sqrt{Y})$
0	log(Y)**
0.5	$Y^{0.5} = \sqrt{Y}$
1	$Y^1 = Y$
2	Y^2
3	Y^3

Finally: An awesome tutorial in python with code examples, there is also another code example here



* Maybe there is a slight problem in the python vs R code, details here, but needs investigating.

MANN-WHITNEY U TEST

(what is?) -the **Mann–Whitney *U* test** is a nonparametric test of the null hypothesis that it is equally likely that a randomly selected value from one sample will be less than or greater than a randomly selected value from a second sample.

In other words: This test can be used to determine whether **two *independent*** samples were selected **from populations having the same distribution**.

Unlike the *t*-test it does not require the assumption of normal distributions. It is nearly as efficient as the *t*-test on normal distributions.

Kernel Density Estimation

This tutorial actually explains why we should use KDE over a Histogram, it explains the cons of histograms and how KDE helps solve some issue that we usually encounter in 'Sparse' histograms where the distribution is hard to figure out.

+ Supposedly a better implementation of KDE than SCIPY

How to use KDE? A tutorial about kernel density and how to use it in python. Has several good graphs and shows use cases.

Video tutorials about Kernel Density:

1. KDE
2. Non parametric Kernel Regression Estimation
3. Non parametric Sieve Estimation
4. Semi- nonparametric estimation

Udacity Video Tutorial - pretty good

1. Comparison and benchmarks of KDE algo's
2. SK LEARN
3. Gaussian KDE in scipy, version 2

STATIONARY TIME SERIES

What is? A time series without a trend or seasonality, in other words non-stationary has a trend or seasonality

There are ways to remove the trend and seasonality, i.e., take the difference between time points.

1. T+1 - T
2. Bigger lag to support seasonal changes
3. pandas.diff()
4. Plot a histogram, plot a log(X) as well.
5. Test for the unit root null hypothesis - i.e., use the Augmented dickey fuller test to determine if two samples originate in a stationary or a non-stationary (seasonal/trend) time series

PROBABILITY AND STATISTICS

(Difference between?)

- **I.e, *Probability* deals with predicting the likelihood of future events, while *statistics* involves the analysis of the frequency of past events.**
- The **problems** considered by probability and statistics are **inverse to each other**.
- In probability theory we *consider some underlying process which has some randomness or uncertainty modeled by random variables, and we figure out what happens.*

=> Underlying process + randomness and random variables -> what happens next?

- In statistics we *observe something that has happened, and try to figure out what underlying process would explain those observations.*

=> observe what happened -> what is the underlying process?

- **Finally, probability** theory is mainly concerned with the **deductive** part, **statistics** with the **inductive** part of modeling processes with uncertainty

(another angle) The main difference between probability and statistics has to do with knowledge

- what are the known facts? Inherent in both probability and statistics is a population,
- every individual we are interested in studying, and a sample, consisting of the individuals that are selected from the population.
- **in probability:** would start with us knowing everything about the composition of a population, and then would ask, "What is the likelihood that a selection, or sample, from the population, has certain characteristics?"
- **In statistics**: we have no knowledge about the types of socks in the drawer. we infer properties about the population on the basis of a random sample.

Some calculations to get you into probability:

- Finding out the probability of an event
- Of two consecutive events (multiplication)
- Of several events (sum)
- Etc..

The *opposite* or *complement* of an event *A* is the event [not *A*] (that is, the event of *A* not occurring), often denoted as $\overline{A}, A^\complement, \neg A$, or $\sim A$; its probability is given by *P*(not *A*) = 1 − *P*(*A*).[29] As an example, the chance of not rolling a six on a six-sided die is 1 – (chance of rolling a six) $= 1 - \frac{1}{6} = \frac{5}{6}$. See Complementary event for a more complete treatment.

If two events *A* and *B* occur on a single performance of an experiment, this is called the intersection or joint probability of *A* and *B*, denoted as $P(A \cap B)$.

Independent events [edit]

If two events, *A* and *B* are independent then the joint probability is

$$P(A \text{ and } B) = P(A \cap B) = P(A)P(B),$$

for example, if two coins are flipped the chance of both being heads is $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.[30]

Mutually exclusive events [edit]

If either event *A* or event *B* occurs on a single performance of an experiment this is called the union of the events *A* and *B* denoted as $P(A \cup B)$. If two events are mutually exclusive then the probability of either occurring is

$$P(A \text{ or } B) = P(A \cup B) = P(A) + P(B).$$

For example, the chance of rolling a 1 or 2 on a six-sided die is
$P(1 \text{ or } 2) = P(1) + P(2) = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}.$

Not mutually exclusive events [edit]

If the events are not mutually exclusive then

$$P\left(A \text{ or } B\right) = P\left(A\right) + P\left(B\right) - P\left(A \text{ and } B\right).$$

For example, when drawing a single card at random from a regular deck of cards, the chance of getting a heart or a face card (J,Q,K) (or one that is both) is $\frac{13}{52} + \frac{12}{52} - \frac{3}{52} = \frac{11}{26}$, because of the 52 cards of a deck 13 are hearts, 12 are face cards, and 3 are both: here the possibilities included in the "3 that are both" are included in each of the "13 hearts" and the "12 face cards" but should only be counted once.

Conditional probability [edit]

Conditional probability is the probability of some event *A*, given the occurrence of some other event *B*. Conditional probability is written $P(A \mid B)$, and is read "the probability of *A*, given *B*". It is defined by[31]

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

If $P(B) = 0$ then $P(A \mid B)$ is formally undefined by this expression. However, it is possible to define a conditional probability for some zero-probability events using a σ-algebra of such events (such as those arising from a continuous random variable).[*citation needed*]

For example, in a bag of 2 red balls and 2 blue balls (4 balls in total), the probability of taking a red ball is $1/2$; however, when taking a second ball, the probability of it being either a red ball or a blue ball depends on the ball previously taken, such as, if a red ball was taken, the probability of picking a red ball again would be $1/3$ since only 1 red and 2 blue balls would have been remaining.

Inverse probability [edit]

In probability theory and applications, **Bayes' rule** relates the odds of event A_1 to event A_2, before (prior to) and after (posterior to) conditioning on another event B. The odds on A_1 to event A_2 is simply the ratio of the probabilities of the two events. When arbitrarily many events A are of interest, not just two, the rule can be rephrased as **posterior is proportional to prior times likelihood**, $P(A|B) \propto P(A)P(B|A)$ where the proportionality symbol means that the left hand side is proportional to (i.e., equals a constant times) the right hand side as A varies, for fixed or given B (Lee, 2012; Bertsch McGrayne, 2012). In this form it goes back to Laplace (1774) and to Cournot (1843); see Fienberg (2005). See Inverse probability and Bayes' rule.

Summary of probabilities

Event	Probability
A	$P(A) \in [0,1]$
not A	$P(A^\complement) = 1 - P(A)$
A or B	$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ $P(A \cup B) = P(A) + P(B)$ if A and B are mutually exclusive
A and B	$P(A \cap B) = P(A\|B)P(B) = P(B\|A)P(A)$ $P(A \cap B) = P(A)P(B)$ if A and B are independent
A given B	$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B\|A)P(A)}{P(B)}$

BAYES, BAYESIAN BELIEF NETWORKS

Introduction To BBS - a very good blog post
A complementing SLIDE presentation that shows how to build the network's tables
A very nice presentation regarding BBS

MARKOV MODELS

Random vs Stochastic (here and here):

- A **variable** is **'random'**.
- A **process** is **'stochastic'**.
  Apart from this difference the two words are synonyms

In other words:

- A **random vector** is a generalization of a **single random variables** to many.
- A **stochastic process** is a sequence of **random variables**, or a **sequence of random vectors** (and then you have a vector-stochastic process).

(What is a Markov Model?) A Markov Model is a stochastic(random) model which models temporal or sequential data, i.e., data that are ordered.

- It provides a way to model the dependencies of current information (e.g. weather) with previous information.
- It is composed of states, transition scheme between states, and emission of outputs (discrete or continuous).
- Several **goals** can be accomplished by using Markov models:
  - Learn **statistics** of **sequential data.**
  - Do **prediction** or estimation.
  - Recognize **patterns**.

(sunny cloudy explanation) Markov Chains is a probabilistic process, that **relies on the current state to predict the next state.**

- to be effective the current state has to be dependent on the previous state in some way
- if it looks cloudy outside, the next state we expect is rain.
- If the rain starts to subside into cloudiness, the next state will most likely be sunny.
- **Not every process has the Markov Property**, such as the Lottery, this weeks winning numbers have no dependence to the previous weeks winning numbers.

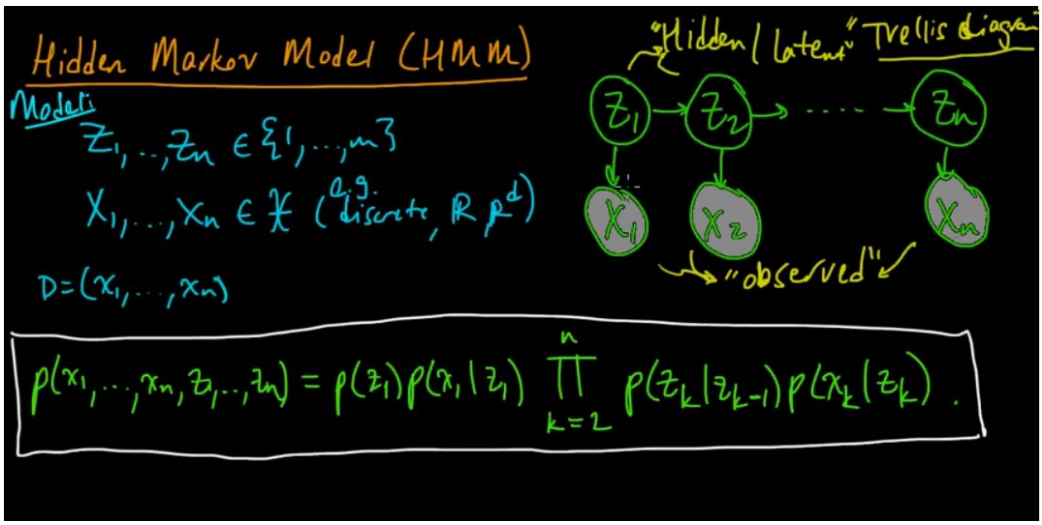
1. They show how to build an order 1 markov table of probabilities, predicting the next state given the current.
2. Then it shows the state diagram built from this table.
3. Then how to build a transition matrix from the 3 states, i.e., from the probabilities in the table
4. Then how to calculate the next state using the “current state vector” doing vec*matrix multiplications.
5. Then it talks about the setting always into the rain prediction, and the solution is using two last states in a bigger table of order 2. **He is not really telling us why the**

probabilities don't change if we add more states, it stays the same as in order 1, just repeating.

HIDDEN MARKOV MODEL

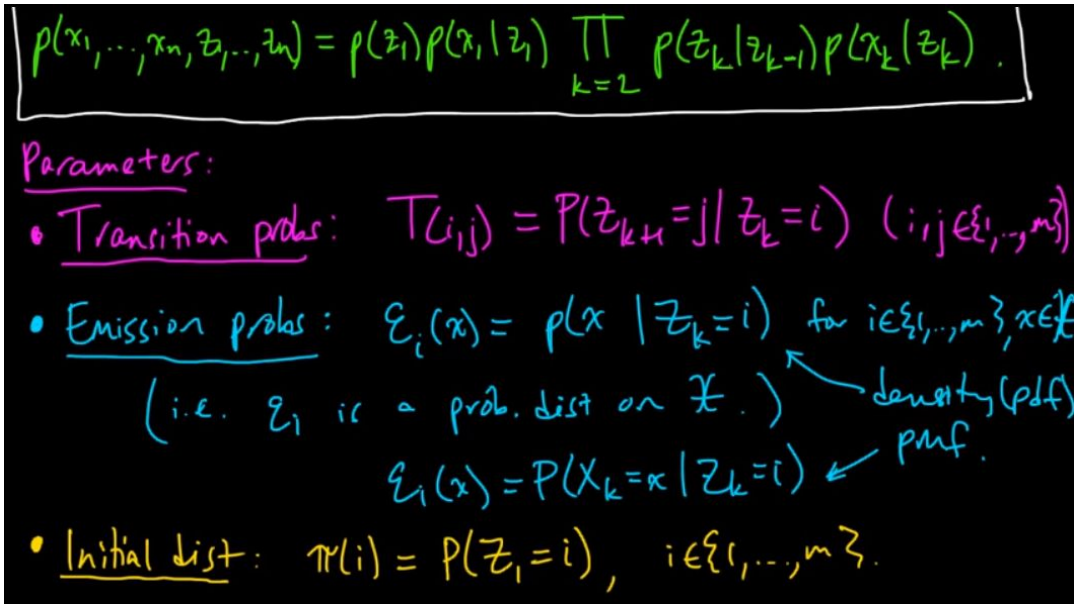
(what is? And why HIDDEN?) - the idea is that there are things that you CAN OBSERVE and there are things that you CAN'T OBSERVE. From the things you OBSERVE you want to INFER the things you CAN'T OBSERVE (HIDDEN). I.e., **you play against someone else in a game, you don't see their choice of action, but you see the result.**

This youtube video part1 - explains about the hidden markov model. It shows the visual representation of the model and how we go from that the formula:



It breaks down the formula to:

- **transition probability formula** - the probability of going from Zk to Zk+1
- **emission probability formula** - the probability of going from Zk to Xk
- **(Pi) Initial distribution -** the probability of Z1=i for i=1..m

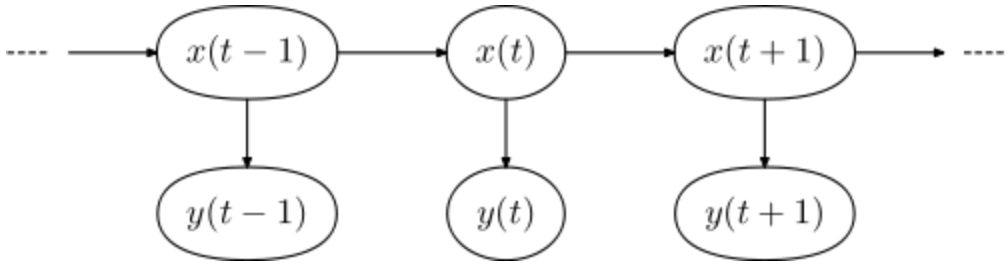


In part2 of the video:

* HMM in weka, with github, working on 7.3, not on 9.1

1. Probably the **simplest** explanation of Markov Models and HMM as a “game” - link
2. This video explains that building blocks of the needed knowledge in HMM, starting probabilities P0, transitions and emissions (state probabilities)
3. This post, explains HMM and ties our understanding.

A cute explanation on quora:



This is the iconic image of a Hidden Markov Model. There is some state (x) that changes with time (markov). And you want to estimate or track it. Unfortunately, you cannot directly observe this state (hidden). That's the hidden part. But, you can observe something correlated with the state (y).

OBSERVED DATA -> INFER -> what you CANT OBSERVE (HIDDEN).

Given a history of observations, say Y_1=w, Y_2=f, Y_3=c, we want to compute the posterior distribution that you are happy at step 3. That is, we want to estimate:

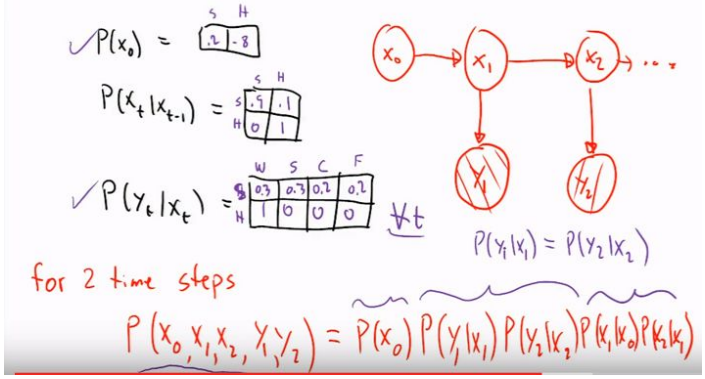
$$P(X_3=h \mid Y_1=w, Y_2=f, Y_3=c)$$

Considering this model:

- **where P(X0) is the initial state for happy or sad**
- **Where P(Xt | X t-1) is the transition model from time-1 to time**
- **Where P(Yt | Xt) is the observation model for happy and sad (X) in 4 situations (w, sad, crying, facebook)**

Dynamic model

*In general, we assume we have an **initial** distribution **$P(X_0)$**, a **transition** model **$P(X_t | X_{t-1})$**, and an **observation** model **$P(Y_t | X_t)$**.*



for 2 time steps

$$P(x_0, x_1, x_2, y_1, y_2) = P(x_0)\,P(y_1|x_1)\,P(y_2|x_2)\,P(x_1|x_0)\,P(x_2|x_1)$$

INFORMATION GAIN

Entropy - **lack of order** or **lack of predictability**

Formula for **2 classes:**

Entropy:

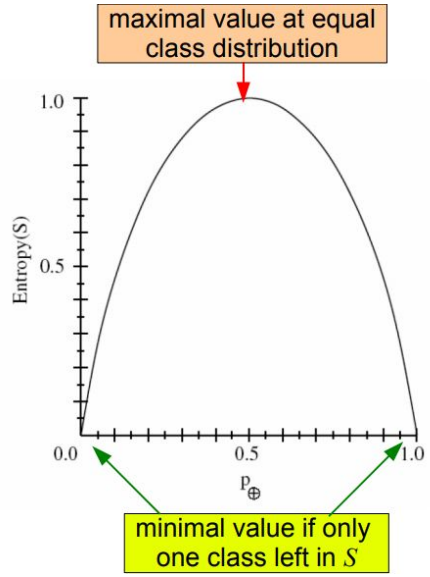
$$E(S) = -p_{\oplus} \cdot \log_2 p_{\oplus} - p_{\ominus} \cdot \log_2 p_{\ominus}$$

NOTE: Entropy can be generalized as a formula for **N > 2 classes:**

$$E(S) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n = -\sum_{i=1}^{n} p_i \log p_i$$

(We want to grow a simple tree) **awesome pdf tutorial**→ a good attribute prefers attributes that split the data so that each successor node is as pure as possible

- i.e., the distribution of examples in each node is so that it mostly contains examples of a single class
- In other words: We want a measure that prefers attributes that have a high degree of „order“:
- Maximum order: All examples are of the same class
- Minimum order: All classes are equally likely → Entropy is a measure for (un-)orderedness Another interpretation:
- Entropy is the amount of information that is contained
- all examples of the same class → no information



Entropy is the amount of **<u>unorderedness</u> in the class distribution of S**
IMAGE above:

- Maximal value when the equal class distribution
- Minimal value when only one class is in S

So basically if we have the **outlook attribute** and it has **3 categories,** we calculate the entropy for E(feature=category) for all 3.

Example: Attribute Outlook

- Outlook = sunny: 3 examples yes, 2 examples no

$$E(\text{Outlook}=\text{sunny})=-\frac{2}{5}\log\left(\frac{2}{5}\right)-\frac{3}{5}\log\left(\frac{3}{5}\right)=0.971$$

- Outlook = overcast: 4 examples yes, 0 examples no

$$E(\text{Outlook}=\text{overcast})=-1\log(1)-0\log(0)=0$$

Note: this is normally undefined. Here: = 0

- Outlook = rainy : 2 examples yes, 3 examples no

$$E(\text{Outlook}=\text{rainy})=-\frac{3}{5}\log\left(\frac{3}{5}\right)-\frac{2}{5}\log\left(\frac{2}{5}\right)=0.971$$

INFORMATION: The I(S,A) formula below.
What we actually want is the **average entropy of the entire split**, that **corresponds to an entire attribute,** i.e., OUTLOOK (sunny & overcast & rainy)

Average Entropy / Information

- **Problem:**
  - Entropy only computes the quality of a single (sub-)set of examples
    - corresponds to a single value
  - How can we compute the quality of the entire split?
    - corresponds to an entire attribute
- **Solution:**
  - Compute the weighted average over all sets resulting from the split
    - weighted by their size

$$I(S,A)=\sum_i \frac{|S_i|}{|S|}\cdot E(S_i)$$

- **Example:**
  - Average entropy for attribute *Outlook*:

$$I(\text{Outlook})=\frac{5}{14}\cdot 0.971+\frac{4}{14}\cdot 0+\frac{5}{14}\cdot 0.971=0.693$$

Information Gain: is actually what we gain by subtracting information from the entropy.
In other words we find the attributes that maximizes that difference, *in other other words, the attribute that* ***reduces the unorderness / lack of order / lack of predictability.***

The ***BIGGER GAIN is selected.***

Information Gain

- When an attribute A splits the set S into subsets S_i
  - we compute the average entropy
  - and compare the sum to the entropy of the original set S

Information Gain for Attribute A

$$Gain(S,A) = E(S) - I(S,A) = E(S) - \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)$$

- The attribute that maximizes the difference is selected
  - i.e., the attribute that reduces the unorderedness most!
- **Note**:
  - maximizing information gain is equivalent to minimizing average entropy, because $E(S)$ is constant for all attributes A

There are some properties to Entropy that influence INFO GAIN (?):

Properties of Entropy

- Entropy is the only function that satisfies all of the following three properties
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property:*
    - p, q, r are classes in set S, and T are examples of class $t = q \vee r$

$$E_{p,q,r}(S) = E_{p,t}(S) + \frac{|T|}{|S|} \cdot E_{q,r}(T)$$

→ decisions can be made in several stages

- Simplification of computation of average entropy (information):

$$I(S,[2,3,4]) = -\tfrac{2}{9}\cdot\log(\tfrac{2}{9}) - \tfrac{3}{9}\cdot\log(\tfrac{3}{9}) - \tfrac{4}{9}\cdot\log(\tfrac{4}{9})$$
$$= -\tfrac{1}{9}\left(2\cdot\log(2) + 3\cdot\log(3) + 4\cdot\log(4) - 9\cdot\log(9)\right)$$

There are some disadvantages with INFO GAIN, done use it when an attribute has many number values, such as “day” (date wise) 05/07, 06/07, 07/07..31/07 etc.

Information gain is biased towards choosing attributes with a large number of values and causes:

- Overfitting
- fragmentation

Highly-branching attributes

- Problematic: attributes with a large number of values
  - extreme case: each example has its own value
    - e.g. example ID; Day attribute in weather data
- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values
- This may cause several problems:
  - *Overfitting*
    - selection of an attribute that is non-optimal for prediction
  - *Fragmentation*
    - data are fragmented into (too) many small sets

We measure **Intrinsic information of an attribute,** i.e., **Attributes with higher intrinsic information are less useful.**

We define **Gain Ratio** as info-gain with **less bias toward multi value attributes, ie., “days”**
NOTE: Day attribute would still **win with the Gain Ratio, Nevertheless: Gain ratio is more reliable than Information Gain**

Intrinsic Information of an Attribute

- Intrinsic information of a split
  - entropy of distribution of instances into branches
  - i.e. how much information do we need to tell which branch an instance belongs to

$$IntI(S,A) = -\sum_i \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right)$$

- Example:
  - Intrinsic information of Day attribute:

$$IntI(\text{Day}) = 14 \times \left(-\tfrac{1}{14} \cdot \log\left(\tfrac{1}{14}\right)\right) = 3.807$$

- Observation:
  - Attributes with higher intrinsic information are less useful

Gain Ratio

- modification of the information gain that reduces its bias towards multi-valued attributes
- takes number and size of branches into account when choosing an attribute
  - corrects the information gain by taking the *intrinsic information* of a split into account
- Definition of Gain Ratio:

$$GR(S,A) = \frac{Gain(S,A)}{IntI(S,A)}$$

- Example:
  - Gain Ratio of Day attribute

$$GR(\text{Day}) = \frac{0.940}{3{,}807} = 0.246$$

Therefore, we define the alternative, which is the **GINI INDEX**. It measures **impurity**, we define the **average Gini, and the Gini Gain.**

Gini Index

- Many alternative measures to Information Gain
- Most popular altermative: Gini index
  - used in e.g., in CART (Classification And Regression Trees)
  - impurity measure (instead of entropy)

$$Gini(S) = 1 - \sum_i p_i^2$$

  - average Gini index (instead of average entropy / information)

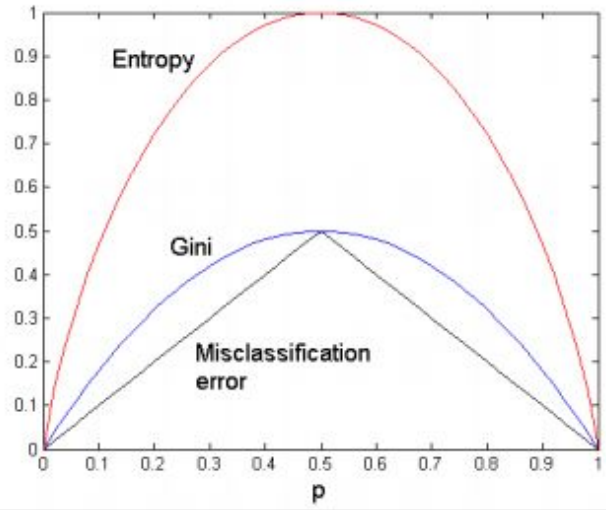
$$Gini(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

  - Gini Gain
    - could be defined analogously to information gain
    - but typically avg. Gini index is minimized instead of maximizing Gini gain

27 © J. Fürnkranz

Comparison among Splitting Criteria

For a 2-class problem:



FINALLY, further reading about decision trees and examples of INFOGAIN and GINI here.

Feature Selection

A series of good articles that explain about several techniques for feature selection

1. Univariate and independent features
2. Linear models and regularization, doing feature ranking
3. Random forests and feature ranking
4. Stability selection and recursive feature elimination (RFE). are wrapper methods in sklearn for the purpose of feature selection. RFE in sklearn
5. Kernel feature selection via conditional covariance minimization (netanel d.)

MULTI LABEL CLASSIFICATION

(what is?) Multilabel classification is a classification problem where multiple target labels can be assigned to each observation instead of only one like in multiclass classification.
Two different approaches exist for multilabel classification:

- *Problem transformation methods* try to transform the multilabel classification into binary or multiclass classification problems.
- *Algorithm adaptation methods* adapt multiclass algorithms so they can be applied directly to the problem.

I.e., the Two approaches are:

- Use a classifier that does multi label
- Use any classifier with a wrapper that compares each two labels

Okay PDF that explains about multi label classification.

An awesome Paper that explains all of these methods in detail, also available here!
PT1: for each sample select one label, remove all others.
PT2: remove every sample which has multi labels.
PT3: for every combo of labels create a single-label, i.e. A&B, A&C etc..
PT4: (most common) create L datasets, for each label learn a binary representation, i.e., is it there or not.
PT5: duplicate each sample with only one of its labels
PT6: read the paper

There are other approaches for doing it within algorithms, they rely on the ideas PT3\4\5\6 implemented in the algorithms, or other tricks.

They also introduce **Label cardinality** and **label density**.

REGRESSION ALGORITHMS

1. CART - classification and regression tree, basically the diff between classification and regression trees - instead of IG we use sum squared error
2. SVR - regression based svm, with kernel only.
3. NNR- regression based NN, one output node
4. LOGREG - Logistic regression - is used as a classification algo to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Output is BINARY. I.e., If the likelihood of killing the bug is > 0.5 it is assumed dead, if it is < 0.5 it is assumed alive.
    - Assumes binary outcome
    - Assumes no outliers
    - Assumes no intercorrelations among predictors (inputs?)

Regression Measurements:

1. R^2 - several reasons it can be too high.
    a. Too many variables
    b. Overfitting
    c. Time series - seasonality trends can cause this

KERNEL REGRESSION

Gaussian Kernel Regression does–it takes a weighted average of the surrounding points

- **variance**, sigma^2. Informally, this parameter will **control the smoothness** of your approximated function.
- **Smaller values** of sigma will cause the function to **overfit** the data points, while **larger values** will cause it to **underfit**
- **There is a proposed method to find sigma in the post!**
- Gaussian Kernel Regression is **equivalent to creating an RBF Network** with the following properties: - described in the post

IMBALANCED DATASETS

(the BEST resource and a great api for python) with visual samples - it actually works well on clustering.

Systematic Investigation of imbalance effects in CNN's, with several observations. This is crucial when training networks, because in real life you dont always get a balanced DS.
They recommend the following:

1. (i) the effect of class imbalance on classification performance is **detrimental**;
2. (ii) the method of addressing class imbalance that emerged as dominant in almost all analyzed scenarios was **oversampling**;
3. (iii) **oversampling** should be applied to the level that totally **eliminates the imbalance**, whereas undersampling can perform better when the imbalance is only removed to some extent;
4. (iv) as opposed to some classical machine learning models, oversampling **does not necessarily cause overfitting of CNNs;**
5. (v) **thresholding** should be applied to compensate for **prior class probabilities** when overall number of properly classified cases is of interest.

General Rules:

1. Many samples - undersampling
2. Few samples - over sampling
3. Consider random and non-random schemes
4. Different sample rations, instead of 1:1 (proof? papers?)

Balancing data sets (wiki, scikit learn & examples in SKLEARN):

1. Oversampling the minority class
    a. (Random) duplication of samples
    b. SMOTE (in weka + needs to be installed & paper) - find k nearest neighbours, New_Sample = (random num in [0,1]) * vec(ki,current_sample)
        - **(in weka)** The nearestNeighbors parameter says how many nearest neighbor instances (surrounding the currently considered instance) are used to build an inbetween synthetic instance. The default value is 5. Thus the attributes of 5 nearest neighbors of a real existing instance are used to compute a new synthetic one.
        - **(in weka)** The percentage parameter says how many synthetic instances are created based on the number of the class with less instances (by default - you can also use the majority class by setting the -Coption). The default value is 100. This means if you have 25 instances in your minority class, again 25 instances are created synthetically from these (using their nearest neighbours' values). With 200% 50 synthetic instances are created and so on.

    c. ADASYN - shifts the classification boundary to the minority class, synthetic data generated for majority class.
2. Undersampling the majority class
    a. Remove samples
    b. Cluster centroids - replaces a cluster of samples (k-means) with a centroid.
    c. Tomek links - cleans overlapping samples between classes in the majority class.
    d. Penalizing the majority class during training
3. Combined over and under (hybrid) - i.e., SMOTE and tomek/ENN
4. Ensemble sampling
    a. EasyEnsemble
    b. BalanceCascade
5. Dont balance, try algorithms that perform well with unbalanced DS
    a. Decision trees - C4.5\5\CART\Random Forest
    b. SVM
6. Penalize Models -
    a. added costs for misclassification on the minority class during training such as penalized-SVM
    b. a CostSensitiveClassifier meta classifier in Weka that wraps classifiers and applies a custom penalty matrix for miss classification.
    c. complex

CLUSTERING

KNN intuition 2 - classify a new sample by looking at the majority vote of its K-nearest neighbours. k=1 special case. Even amount of classes needs an odd K that is not a multiply of the amount of classes in order to break ties.

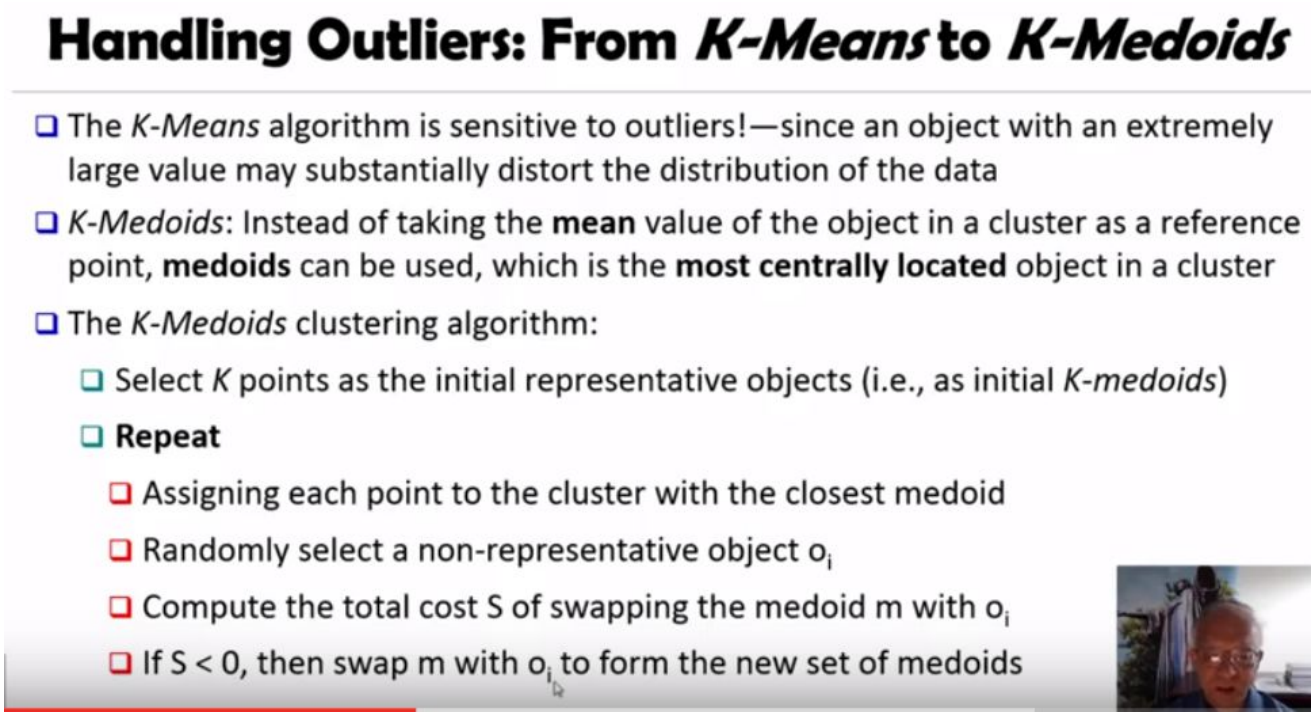
Kmeans:

- Sensitive to outliers, can skew results (because we rely on the mean)

K-mediods - basically k-means with a most center object rather than a center virtual point that was based on mean distance from all points, we keep choosing medoids samples based on minimised SSE

- k-medoid is a classical partitioning technique of clustering that clusters the data set of n objects into k clusters known *a priori*.
- It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.
- A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster.
- Does Not scale to many samples, its O(K*n-K)^2
- Randomized resampling can assure efficiency and quality.

From youtube (okay video)



X-means

X-means(paper):

1. Theory behind bic calculation with a formula.
2. Code: Calculate bic in k-means

Splitting idea 1: One at a time. The first idea would be to pick one centroid, produce a new centroid nearby, run K-means to completion and see if the resulting model scores better. If it does, accept the new centroid. If it doesn't, return to the previous structure. But this will need $O(K_{\max})$

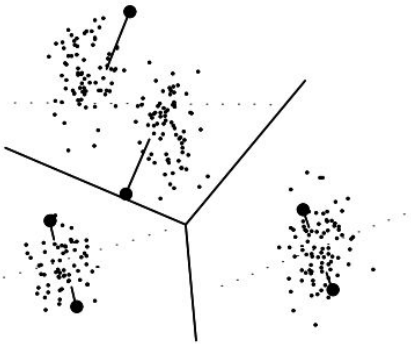


Figure 3: The first step of parallel local 2-means. The line coming out of each centroid shows where it moves to.

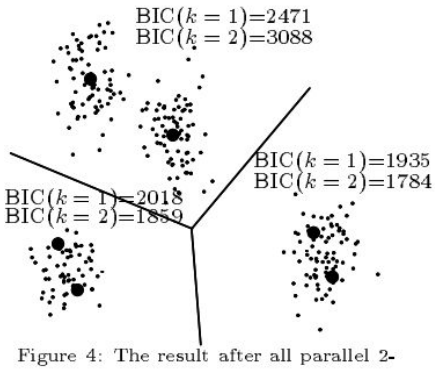


Figure 4: The result after all parallel 2-means have terminated.

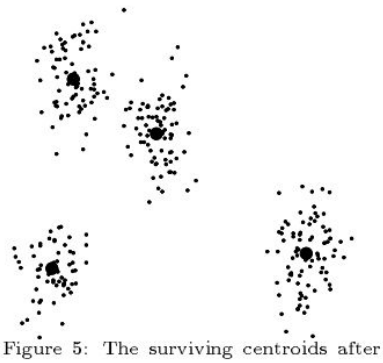


Figure 5: The surviving centroids after all the local model scoring tests.

G-means:

G-means Improves on X-means in the paper: The G-means algorithm starts with a small number of k-means centers, and grows the number of centers. Each iteration of the algorithm splits into two those centers whose data appear not to come from a Gaussian distribution using the **Anderson Darling test**. Between each round of splitting, we run k-means on the entire dataset and all the centers to refine the current solution. We can initialize with just k = 1, or we can choose some larger value of k if we have some prior knowledge about the range of k. G-means repeatedly makes decisions based on a statistical test for the data assigned to each enter. If the data currently assigned to a k-means center appear to be Gaussian, then we want to represent that data with only one center.

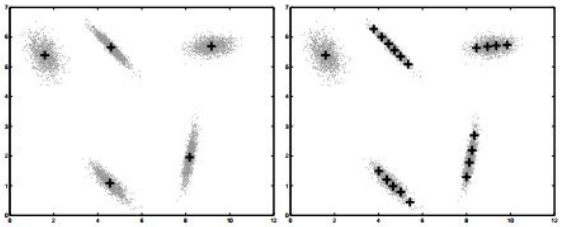


Figure 4: 2-d synthetic dataset with 5 true clusters. On the left, G-means correctly chooses 5 centers and deals well with non-spherical data. On the right, the BIC causes X-means to overfit the data, choosing 20 unevenly distributed clusters.

DBSCAN

a DBSCAN visualization - very good!

HDBSCAN*

(what is?) HDBSCAN is a clustering algorithm developed by Campello, Moulavi, and Sander. It extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters.

- Github code
- (great) Documentation with examples, for clustering, outlier detection, comparison, benchmarking and analysis!
- (jupytr example) - take a look and see how to use it, usage examples are also in the docs and github

What are the algorithm's steps:

1. Transform the space according to the density/sparsity.
2. Build the minimum spanning tree of the distance weighted graph.
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

OPTICS

(What is?) **Ordering points to identify the clustering structure** (**OPTICS**) is an algorithm for finding density-based[1] clusters in spatial data

- Its basic idea is similar to DBSCAN,[3]
- it addresses one of DBSCAN's major weaknesses: **the problem of detecting meaningful clusters in data of varying density.**
- (How?) the points of the database are (linearly) ordered such that points which are spatially closest become neighbors in the ordering.
- a special distance is stored for each point that represents the density that needs to be accepted for a cluster in order to have both points belong to the same cluster. (This is represented as a dendrogram.)

ANOMALY DETECTION

“whether a new observation belongs to the same distribution as existing observations (it is an inlier), or should be considered as different (it is an outlier). “

=> Often, this ability is used to clean real data sets

Two important distinction must be made:

novelty detection:

The training data is **not polluted by outliers**, and we are interested in **detecting anomalies** in new observations.

outlier detection:

The training data **contains outliers**, and we need to **fit the central mode of the training data, ignoring the deviant observations**.

A comparison of One-class SVM versus Elliptic Envelope versus Isolation Forest versus LOF in sklearn. (The examples below illustrate how the performance of the covariance.EllipticEnvelope degrades as the data is less and less unimodal. The svm.OneClassSVM works better on data with multiple modes and ensemble.IsolationForest andneighbors.LocalOutlierFactor perform well in every cases.)

ISOLATION FOREST

The best resource to explain isolation forest - the basic idea is that for an anomaly (in the example) only 4 partitions are needed, for a regular point in the middle of a distribution, you need many many more.

Isolation Forest -Isolating observations:

- randomly selecting a feature
- randomly selecting a split value between the maximum and minimum values of the selected feature.

Recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.
Random partitioning produces noticeable shorter paths for anomalies.

=> when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

the paper is pretty good too - In the training stage, iTrees are constructed by recursively partitioning the given training set until instances are isolated or a specific tree height is reached of which results a partial model.
Note that the tree height limit l is automatically set by the sub-sampling size ψ: l = ceiling(log2 ψ), which is approximately the average tree height [7].
The rationale of growing trees up to the average tree height is that we are only interested in data points that have shorter-than average path lengths, as those points are more likely to be anomalies

LOCAL OUTLIER FACTOR

- LOF computes a score (called local outlier factor) reflecting the degree of abnormality of the observations.
- **It measures the local density deviation of a given data point with respect to its neighbors. The idea is to detect the samples that have a substantially lower density than their neighbors.**
- In practice the local density is obtained from the **k-nearest neighbors.**
- The LOF score of an observation is equal to the ratio of the average local density of his k-nearest neighbors, and its own local density:
    - **a normal instance is expected to have a local density** similar to that of its neighbors,
    - **while abnormal data are expected to have much smaller local density.**

ELLIPTIC ENVELOPE

1. We assume that the **regular data come from a known distribution** (e.g. data are Gaussian distributed).
2. From this assumption, we generally try to **define the "shape" of the data**,
3. And can define **outlying observations as observations which stand far** enough from the fit shape.

ONE CLASS SVM

It looks like there are two such methods, - The 2nd one: The algorithm obtains a spherical boundary, in feature space, around the data. The volume of this hypersphere is minimized, to minimize the effect of incorporating outliers in the solution.
The resulting hypersphere is characterized by a center and a radius R>0 as distance from the center to (any support vector on) the boundary, of which the volume R2 will be minimized.

SVM

SVM - Definition:

- For Optimal **2-class classifier**.
- Extended for **regression** and **clustering problems (1 class).**
- **Kernel-based**
    - **maps** feature vectors into a higher-dimensional space using a kernel function
    - **builds** an **optimal linear discriminating function** in this space (linear?) or an **optimal hyper-plane** (RBF?) that fits the training data
- In case of SVM, the **kernel is not defined explicitly**.
- **A distance needs to be defined** between any 2 points in the hyper-space.
- The solution is optimal, **the margin is maximal.** between the separating hyper-plane and the nearest feature vectors
- **The feature vectors that are the closest to the hyper-plane are called support vectors, which means that the position of other vectors does not affect the hyper-plane (the decision function).**
- The model produced by support vector classification (as described above) depends only on a **subset of the training data**, because the cost function for building the model does not care about training points that lie beyond the margin.

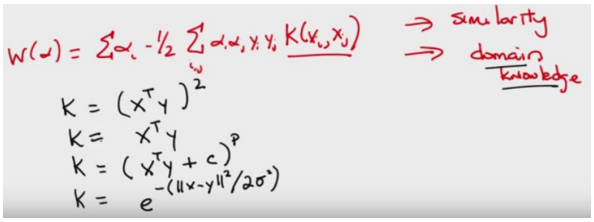
***MULTI CLASS SVM - one against all, one against one, and Direct Acyclic Graph SVM (one against one with DAG). bottom line One Against One in LIBSVM.

A few good explanation about SVM, formulas, figures, C, gamma, etc.

Math of SVM on youtube:

- very good number-based example #1
- Very good but lengthy and chatty example with make-sense math #2 - udacity
    - Linear - maximize the margin, optimal solution, only a few close points are really needed the others are zeroes by the alphas (alpha says "pay attention to this variable") in the quadratic programming equation. XtX is a similarity function (pairs of points that relate to each other in output labels and how similar to one another, Xi's point in the same direction) y1y2 are the labels. Therefore further points are not needed. But the similarity is important here(?)
    - Non-linear - e.g. circle inside a circle, needs to map to a higher plane, a measure of similarity as XtX is important. We use this similarity idea to map into a higher plane, but we choose the higher plane for the purpose of a final function that behaves likes a known function, such as (A+B)^2. It turns out that (q1,q2,root(2)q1q2) is engineered with that root(2) thing for the purpose of making the multiplication of X^tY, which turns out to be (X^tY)^2. We can substitute this formula (X^tY)^2 instead of the X^tX in the quadratic equation to do that for us.This is the kernel trick

that maps the inner class to one side and the outer circle class to the other and passes a plane in between them.
    - Similarity is defined intuitively as all the points in one class vs the other.. I think
    - A general kernel K=(X^tY + C)^p is a polynomial kernel that can define the above function and others.
    - Quadratic eq with possible kernels including the polynomial.



    - **Most importantly the kernel function is our domain knowledge. (?) IMO we should choose a kernel that fits our feature data.**
    - The output of K is a number(?)
    - Infinite dimensions - possible as well.
    - Mercer condition - it acts like a distance\similar so that is the "rule" of which a kernel needs to follow.
- Super good lecture on MIT OPEN COURSE WARE - expands on the quadratic equations that were introduced in the previous course above.

Regularization and influence - (basically **punishment** for overfitting and **raising** the non- linear class points **higher and lower**)

- ***How does regularization look like in SVM - controlling 'C'***
- ***The best explanation about Gamma (and C) in SVM!***

SVR - Definition Support Vector Regression.:

- The method of SVM can be extended to solve regression problems.
- Similar to SVM, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

LibSVM vs LibLinear - using many kernel transforms to turn a non-linear problem into a linear problem beforehand.
From the link above, it seems like liblinear is very much the same thing, without those kernel transforms. So, as they say, in cases where the kernel transforms are not needed (they mention document classification), it will be faster.

- libsvm (SMO) implementation
    - kernel (n^2)
    - Linear SVM (n^3)
- liblinear - optimized to deal with linear classification without kernels
    - Complexity O(n)
    - does not support kernel SVMs.

    - Scores higher

n is the number of samples in the training dataset.

Conclusion: In practice libsvm becomes painfully slow at 10k samples. Hence for medium to large scale datasets use liblinear and forget about libsvm (or maybe have a look at approximate kernel SVM solvers such as LaSVM, which saves training time and memory usage for large scale datasets).

KERNELS

What are kernels in SVM - intuition and example

- allows us to do certain calculations faster which otherwise would involve computations in higher dimensional space.
- K(x, y) = <f(x), f(y)>. Here K is the kernel function, x, y are n dimensional inputs. f is a map from n-dimension to m-dimension space. < x,y> denotes the dot product. usually m is much larger than n.
- normally calculating <f(x), f(y)> requires us to calculate f(x), f(y) first, and then do the dot product. These two computation steps can be quite expensive as they involve manipulations in m dimensional space, where m can be a large number.
- Result is ONLY a scalar, i..e., 1-dim space.
- We **dont** need to do that calc if we use a clever kernel.

Example:
Simple Example: x = (x1, x2, x3); y = (y1, y2, y3). Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3), the kernel is K(x, y) = (<x, y>)^2.
Let's plug in some numbers to make this more intuitive: suppose x = (1, 2, 3); y = (4, 5, 6). Then:
f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9) and f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)
<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024 i.e., 1*16+2*20+*3*24..
A lot of algebra. Mainly because f is a mapping from 3-dimensional to 9 dimensional space.

With a kernel its faster.
K(x, y) = (4 + 10 + 18) ^2 = 32^2 = 1024

A kernel is a magical shortcut to calculate even infinite dimensions!

Relation to SVM?:

- The idea of SVM is that y = w phi(x) +b, where w is the weight, phi is the feature vector, and b is the bias.
- if y> 0, then we classify datum to class 1, else to class 0.
- We want to find a set of weight and bias such that the margin is maximized.
- Previous answers mention that kernel makes data linearly separable for SVM. I think a more precise way to put this is, **kernels do not make the the data linearly separable**.
- The feature vector **phi(x) makes the data linearly separable**. **Kernel is to make the calculation process faster and easier**, especially when the feature vector **phi is of very high dimension (for example, x1, x2, x3, ..., x_D^n, x1^2, x2^2, ..., x_D^2).**
- Why it can also be understood as a measure of similarity: if we put the definition of kernel above, **<f(x), f(y)>**, in the context of SVM and feature vectors, it becomes **<phi(x), phi(y)>. The inner product means the projection of phi(x) onto phi(y**). **or colloquially, how much overlap do x and y have in their feature space. In other words, how similar they are.**

Kernels:

- **SVM::LINEAR Linear kernel. No mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option.** $K(x_i, x_j) = x_i^T x_j$.
- **SVM::RBF Radial basis function (RBF), a good choice in most cases.** $K(x_i, x_j) = e^{-\gamma ||x_i - x_j||^2}, \gamma > 0$.

Intuition for regularization in SVM

Grid search for SVM Hyper parameters - in openCV. Example in log space

- I.e., (for example, C = 2^-5 , 2 ^-3 , . . . , 2^15 , γ = 2^-15 , 2 ^-13 , . . . , 2^3).
- There are heuristic methods that skip some search options
- However, no need for heuristics, computation-time is small, grid can be paralleled and we dont skip parameters.
- Controlling search complexity using two tier grid, coarse grid and then fine tune.

Overfitting advice for SVM:

- Regularization parameter C - penalty example
- In non linear kernels:
    - Kernel choice
    - Kernel parameters
- RBF - gamma, low and high values are far and near influence
    - Great Tutorial at LIBSVM
    - Reasonable first choice
    - when the **relation between class labels and attributes is nonlinear**.
    - Special case of C can make this similar to **linear kernel (only! After finding C and gamma)**
    - Certain parameters makes it behave like the **sigmoid kernel**.
    - Less hyperparameters than RBF kernel.
    - 0 <Kij <1 unlike other kernels where the degree is 0<k<infinity
    - Sigmoid is not valid under some parameters.
    - DON'T USE when the #features is very large, use linear.

Use cases for RBF kernel:

- **Number of instances << number of features**. I.e, 38 instances over 7000 features. **RBF=LINEAR** When the number of features is large, we may not need to use RBF over Linear and vice versa (After finding C and gamma)
- **Number of Instances & features is VERY LARGE.** I.e, 20K samples X 20K features. Similar performance with libsvm and liblinear, liblinear is faster by 150 times. Rule of thumb is to use for document classification.
- **Number of instances >> number of features.** Usually high dimensional mapping using non linear kernel. If we insist on liblinear, -s 2 leads to faster training.

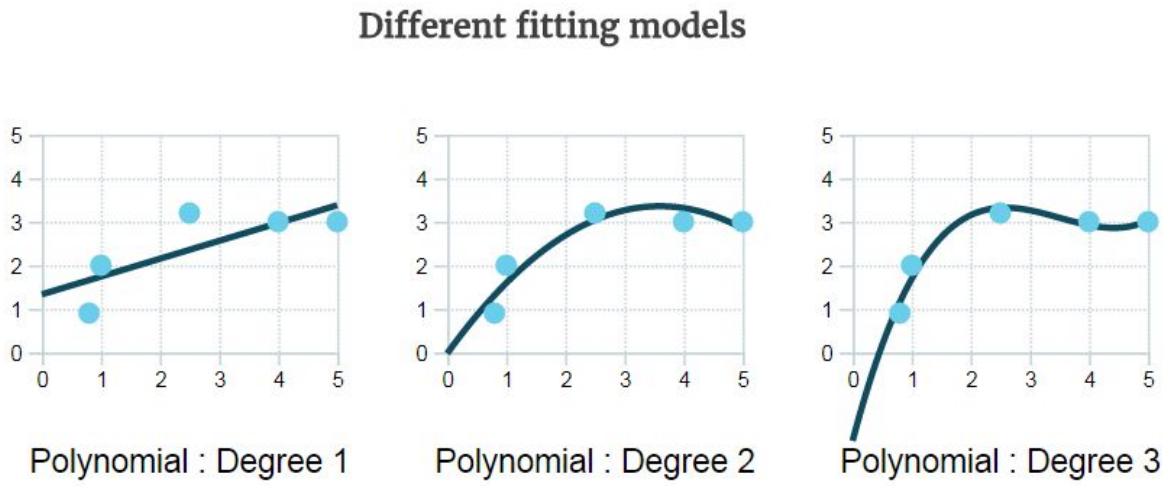
Kdnuggets: When to use DL over SVM and other algorithms. Computationally expensive for a very small boost in accuracy.

REGULARIZATION

(what is?) Regularization (in linear regression) - to find the best model we define a loss or cost function that describes how well the model fits the data, and try minimize it. For a complex model that fits even the noise, i.e., over fitted, we penalize it by adding a complexity term that would add BIGGER LOSS for more complex models.

- Bigger lambda -> high complexity models (**deg 3**) are ruled out, **more punishment.**
- Smaller lambda -> models with high training error are rules out. I.e., linear model on non linear data?, i.e., **deg 1**.
- Optimal is in between (**deg 2**)

L1 - for sparse models,
L1 vs L2, some formula
Rehearsal on vector normalization - for l1,l2,l3,l4 etc, what is the norm? (absolute value in certain cases)

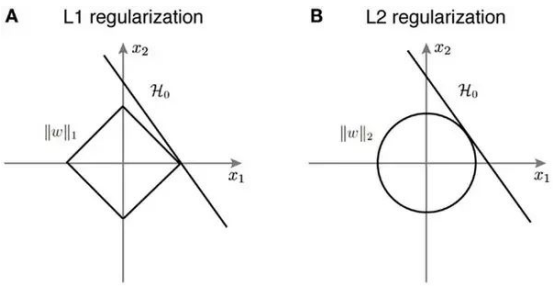


(Difference between? And features of) L1 vs L2 as loss function and regularization.

- **L1 -** moves the regressor faster, feature selection by sparsing coefficients (zeroing them), with sparse algorithms it is computationally efficient, with others no, so use L2.
- **L2 -** moves slower, doesn't sparse, computationally efficient.

Why does L1 lead to sparity?

- **Intuition + some mathematical info**
- L1 & L2 regularization add constraints to the optimization problem. The curve H0 is the hypothesis. The solution is a set of points where the H0 meets the constraints.
- In L2 the the hypothesis is tangential to the ||w||_2. The point of intersection has both x1 and x2 components. On the other hand, in L1, due to the nature of ||w||_1, the viable solutions are limited to the corners of the axis, i.e.,

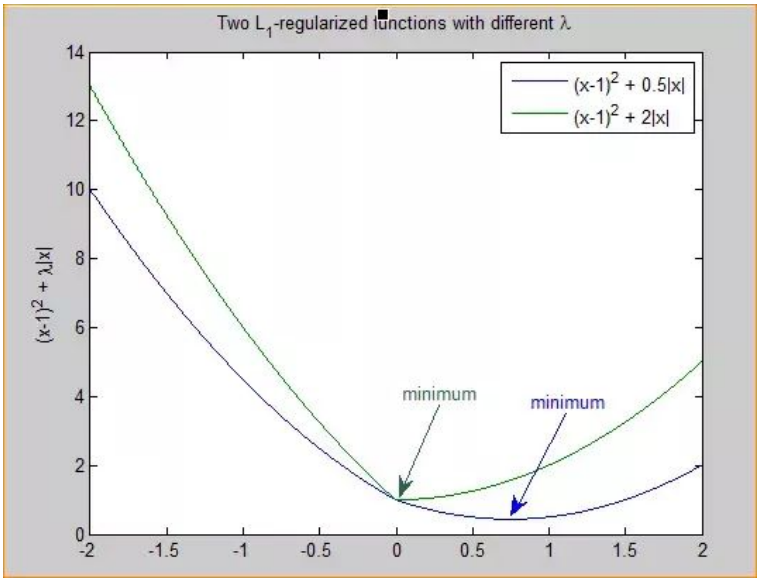


x1. So that the value of x2 = 0. This means that the solution has eliminated the role of x2 leading to sparsity.

- This can be extended to a higher dimensions and you can see why L1 regularization leads to solutions to the optimization problem where many of the variables have value 0.
- **In other words, L1 regularization leads to sparsity.**
- Also considered feature selection - although with LibSVM the recommendation is to feature select prior to using the SVM and use L2 instead.

L1 sparsity - intuition #2

- For simplicity, let's just consider the 1-dimensional case.
- **L2:**
  - L2-regularized loss function F(x)=f(x)+λ // x // ^2 is smooth.
  - This means that the optimum is the stationary point (0-derivative point).
  - The stationary point of F can get very small when you increase λ, but it will still **won't be 0 unless f′(0)=0.**
- **L1:**
  - regularized loss function F(x)=f(x)+λ // x // is **non-smooth, i.e., a min knee of 0.**
  - It's not differentiable at 0.
  - Optimization theory says that the optimum of a function is either the point with 0-derivative or one of the irregularities (corners, kinks, etc.). So, it's possible that the optimal point of F is 0 even if 0 isn't the stationary point of f.
  - **In fact, it would be 0 if λ is large enough (stronger regularization effect). Below is a graphical illustration.**



In multi-dimensional settings: if a feature is not important, the loss contributed by it is small and hence the (non-differentiable) regularization effect would turn it off.

Intuition + formulation, which is pretty good:

Consider the vector $\vec{x} = (1, \varepsilon) \in \mathbb{R}^2$ where $\varepsilon > 0$ is small. The l_1 and l_2 norms of $\vec{x}$, respectively, are given by

$$||\vec{x}||_1 = 1 + \varepsilon, \;\; ||\vec{x}||_2^2 = 1 + \varepsilon^2$$

Now say that, as part of some regularization procedure, we are going to reduce the magnitude of one of the elements of $\vec{x}$ by $\delta \leq \varepsilon$. If we change x_1 to $1 - \delta$, the resulting norms are

$$||\vec{x} - (\delta, 0)||_1 = 1 - \delta + \varepsilon, \;\; ||\vec{x} - (\delta, 0)||_2^2 = 1 - 2\delta + \delta^2 + \varepsilon^2$$

On the other hand, reducing x_2 by δ gives norms

$$||\vec{x} - (0, \delta)||_1 = 1 - \delta + \varepsilon, \;\; ||\vec{x} - (0, \delta)||_2^2 = 1 - 2\varepsilon\delta + \delta^2 + \varepsilon^2$$

The thing to notice here is that, for an l_2 penalty, regularizing the larger term x_1 results in a much greater reduction in norm than doing so to the smaller term $x_2 \approx 0$. For the l_1 penalty, however, the reduction is the same. Thus, when penalizing a model using the l_2 norm, it is highly unlikely that anything will ever be set to zero, since the reduction in l_2 norm going from ε to 0 is almost nonexistent when ε is small. On the other hand, the reduction in l_1 norm is always equal to δ, regardless of the quantity being penalized.

Another way to think of it: it's not so much that l_1 penalties encourage sparsity, but that l_2 penalties in some sense **discourage** sparsity by yielding diminishing returns as elements are moved closer to zero.

(did not watch) but here is andrew ng talks about cost functions.

L2 regularization equivalent to Gaussian prior

Let us imagine that you want to infer some parameter β from some observed input-output pairs $(x_1, y_1) \ldots, (x_N, y_N)$. Let as assume that the outputs are linearly related to the inputs via β and that the data are corrupted by some noise ϵ:

$$y_n = \beta x_n + \epsilon,$$

where ϵ is Gaussian noise with mean 0 and variance σ^2. This gives raise to a Gaussian likelihood:

$$\prod_{n=1}^{N} \mathcal{N}(y_n | \beta x_n, \sigma^2).$$

Let us regularise parameter β by imposing the Gaussian prior $\mathcal{N}(\beta | 0, \lambda^{-1})$,

where λ is a strictly positive scalar. Hence, combining the likelihood and the prior we simply have:

$$\prod_{n=1}^{N} \mathcal{N}(y_n | \beta x_n, \sigma^2) \mathcal{N}(\beta | 0, \lambda^{-1}).$$

Let us take the logarithm of the above expression. Dropping some constants we get:

$$\sum_{n=1}^{N} -\frac{1}{\sigma^2}(y_n - \beta x_n)^2 - \lambda \beta^2 + \text{const.}$$

If we maximise the above expression with respect to β we get the so called maximum a-posteriori estimate for β, or MAP estimate for short. In this expression it becomes apparent why the Gaussian prior can be interpreted as a L2 regularisation term.

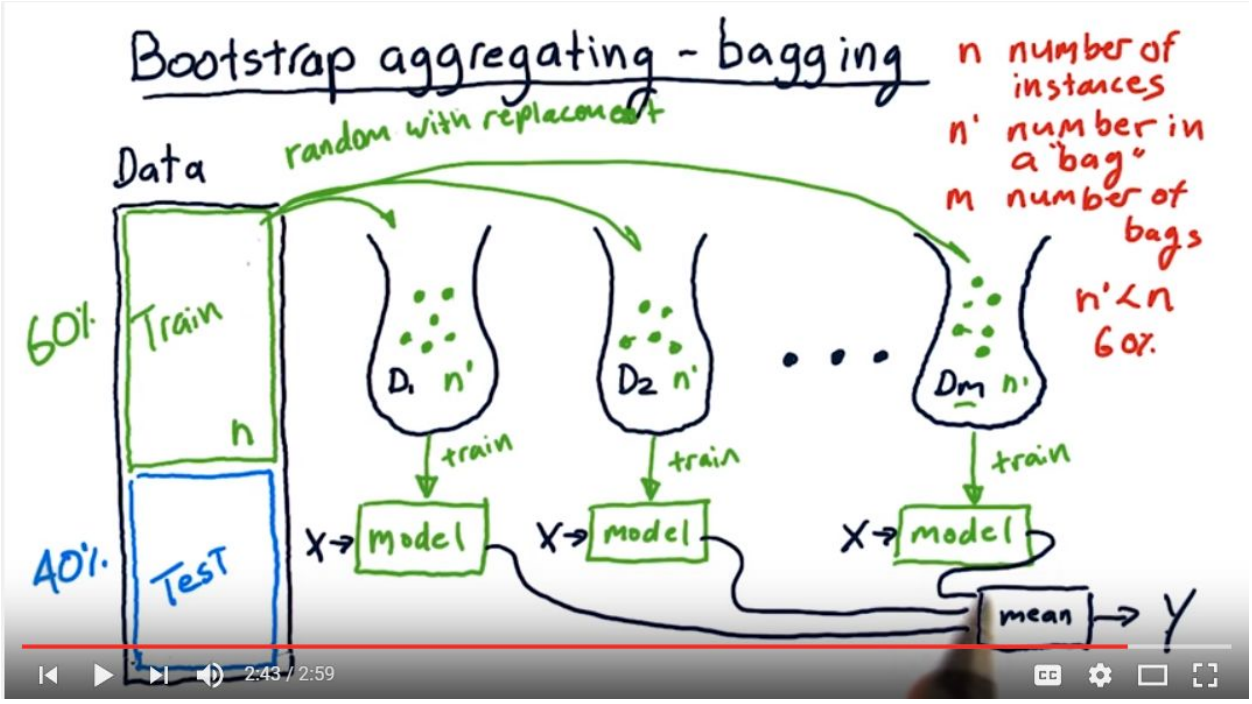
L1 regularization equivalent to a Laplacean Prior(same link as above) - "Similarly the relationship between L1 norm and the Laplace prior can be undestood in the same fashion. Take instead of a Gaussian prior, a Laplace prior combine it with your likelihood and take the logarithm."

How does regularization look like in SVM - controlling 'C'

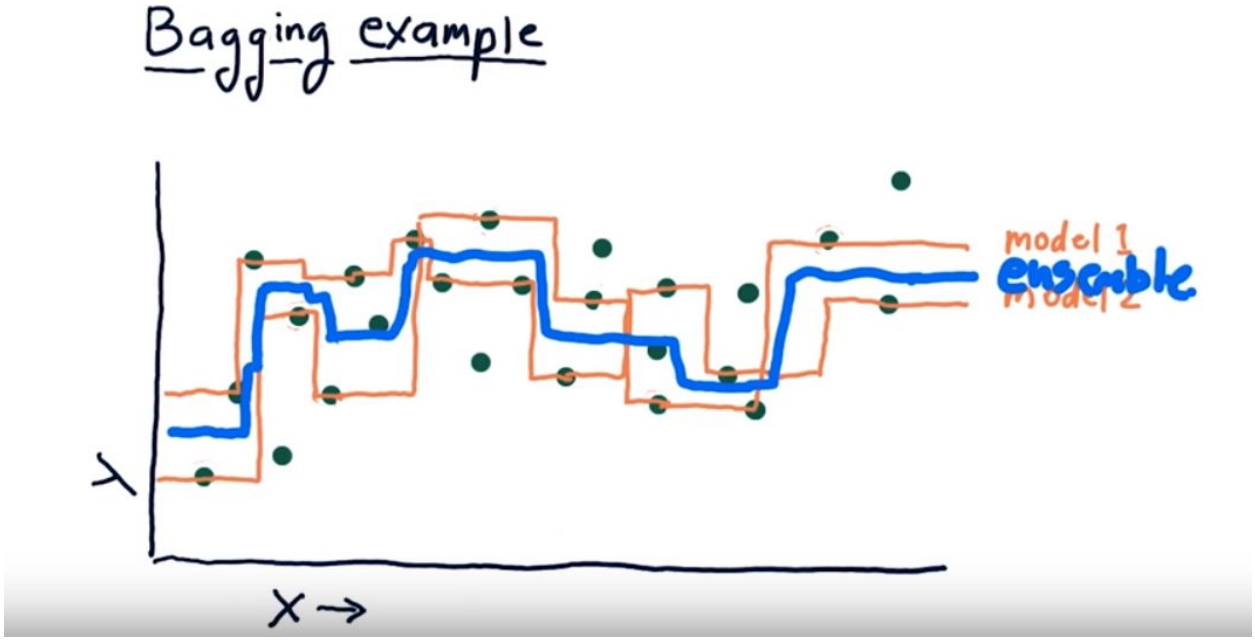
Ensembles in WEKA - bagging (random sample selection, multi classifier training), random forest (random feature selection for each tree, multi tree training), boosting(creating stumps, each new stump tries to fix the previous error, at last combining results using new data, each model is assigned a skill weight and accounted for in the end), voting(majority vote, any set of algorithms within weka, results combined via mean or some other way), stacking(same as voting but combining predictions using a meta model is used).

BAGGING - bootstrap aggregating

Bagging - best example so far, create m bags, put n'<n samples (60% of n) in each bag - with replacement which means that the same sample can be selected twice or more, query from the test (x) each of the m models, calculate mean, this is the classification.



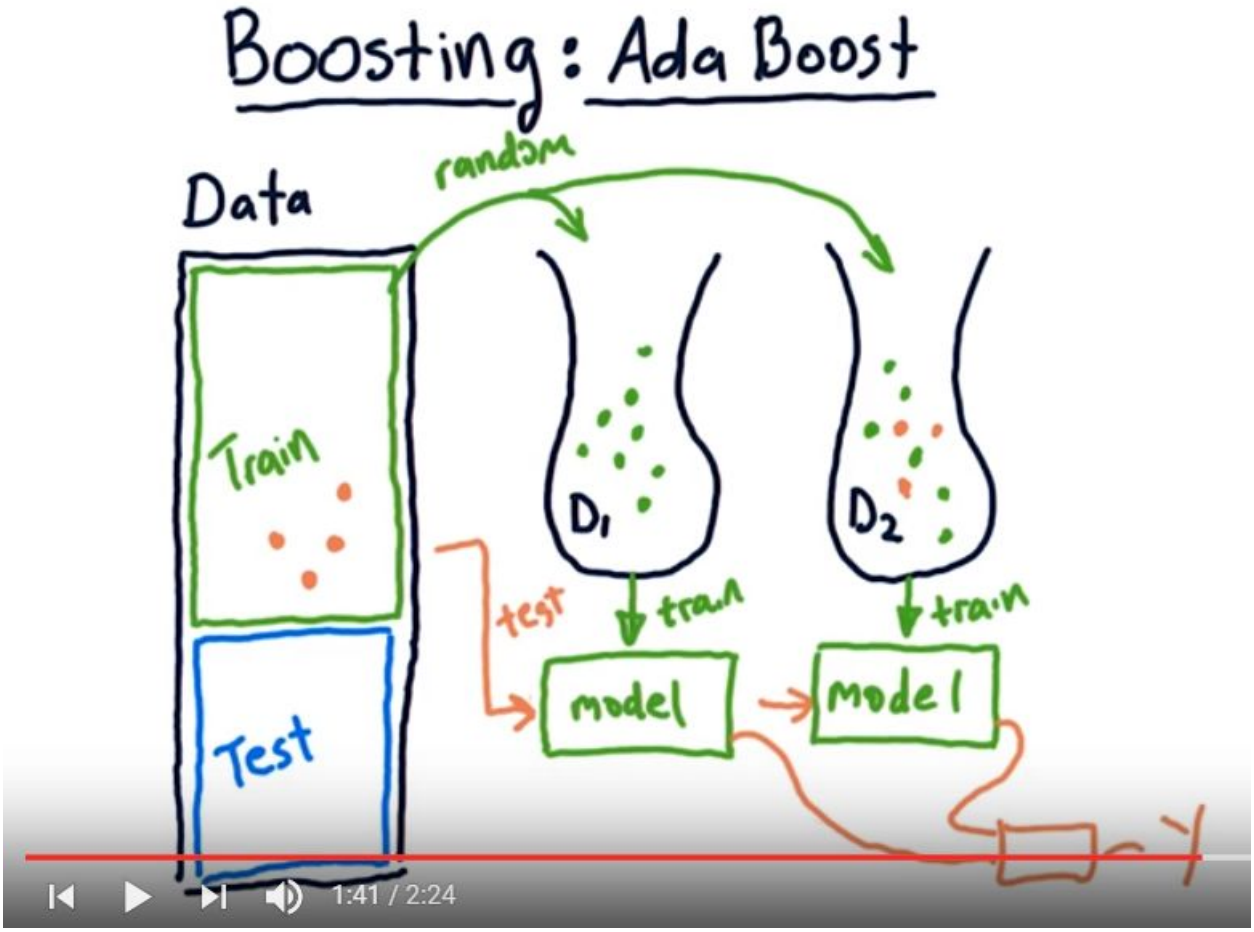
Overfitting - not an issue with bagging, as the mean of the models actually averages or **smoothes** the "curves". Even if all of them are overfitted.



BOOSTING

Adaboost: similar to bagging, create a system that chooses from samples that were modelled poorly before.

1. create bag_1 with n' features <n with replacement, create the model_1, test on ALL train.
2. Create bag_2 with n' features with replacement, but add a bias for selecting from the samples that were wrongly classified by the model_1. Create a model_2. Average results from model_1 and model_2. I.e., who was classified correctly or not.
3. Create bag_3 with n' features with replacement, but add a bias for selecting from the samples that were wrongly classified by the model_1+2. Create a model_3. Average results from model_1, 2 & 3 I.e., who was classified correctly or not. Iterate onward.
4. Create bag_m with n' features with replacement, but add a bias for selecting from the samples that were wrongly classified by the previous steps.



XGBOOST

- What is XGBOOST? XGBoost is an optimized distributed gradient boosting system designed to be highly **efficient**, **flexible** and **portable** **#2nd link**
- Authors Youtube lecture.
- GIT here
- How to code tutorial, short and makes sense, with info about the parameters.
    - Threads
    - Rounds
    - Tree height
    - Loss function
    - Error
    - Cross fold.

- Beautiful Video Class about XGBOOST - mostly practical in jupyter but with some insight about the theory.
- Machine learning mastery - slides, video, lots of info.

R Installation in Weka, then XGBOOST in weka through R
Parameters for weka mlr class.xgboost.

- https://cran.r-project.org/web/packages/xgboost/xgboost.pdf
- Here is an example configuration for multi-class classification:
- weka.classifiers.mlr.MLRClassifier -learner "**nrounds = 10, max_depth = 2, eta = 0.5, nthread = 2"**
- classif.xgboost -params "nrounds = 1000, max_depth = 4, eta = 0.05, nthread = 5, objective = \"multi:softprob\"

Copy: nrounds = 10, max_depth = 2, eta = 0.5, nthread = 2

Special case of random forest using XGBOOST:

```
#Random Forest™ - 1000 trees
bst <- xgboost(data = train$data, label = train$label, max_depth = 4,
num_parallel_tree = 1000, subsample = 0.5, colsample_bytree =0.5, nrounds = 1,
objective = "binary:logistic")

#Boosting - 3 rounds
bst <- xgboost(data = train$data, label = train$label, max_depth = 4, nrounds = 3,
objective = "binary:logistic")
```

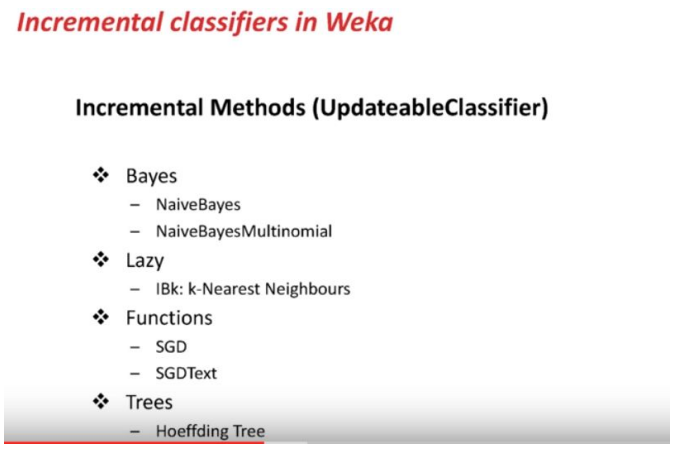
RF1000: - max_depth = 4, num_parallel_tree = 1000, subsample = 0.5, colsample_bytree =0.5, nrounds = 1, nthread = 2
XG: nrounds = 10, max_depth = 4, eta = 0.5, nthread = 2

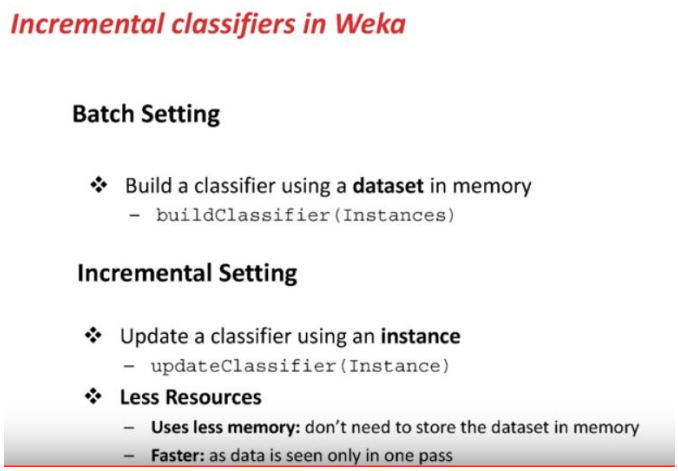
Q-LEARN

Q-Learning

- Markov chain problem, (state, action, new state, reward)
- Lots of Exploration in the beginning, then exploitation
- Returns optimal policy.
- Refer to youtube here

Incremental Learning





- **HOEFFDING TREE IS STATE OF THE ART**
- Using incremental learning for time series, i.e., sentiment from twitter api streat, learning from smiley in weka.
- And time series in weka, with some ideas about dealing with time series data, especially date and time.

TF-IDF

TF-IDF - how important is a word to a document in a corpus

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

Frequency of word in doc / all words in document (normalized bcz docs have diff sizes)

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

measures how important a term is

TF-IDF is TF*IDF

DIMENSIONALITY REDUCTION METHODS

- A small blog post about PCA, AE & TSNE in tensorflow

PCA

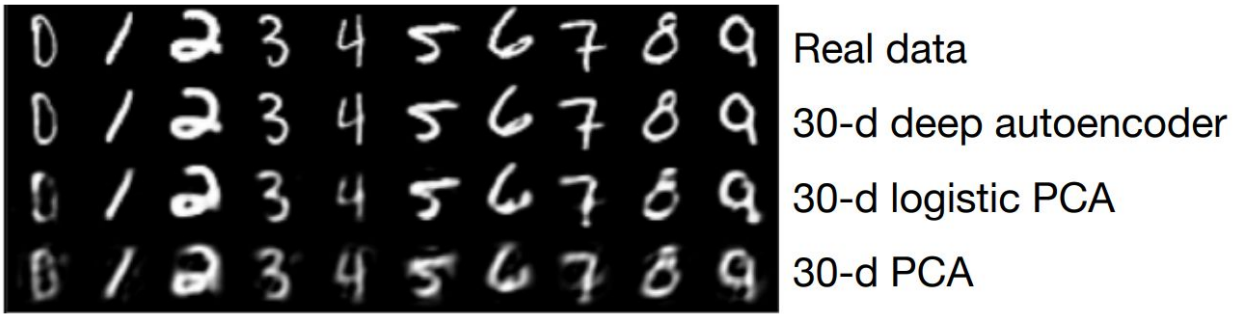
1. **Machine learning mastery:**
    a. **Expected value, variance, covariance**
    b. **PCA (remove the mean from A, calculate cov(A), calculate eig(cov), A*eigK = PCA)**
    c. **EigenDecomposition - what is an eigen vector - simply put its a vector that satisfies A*v = lambda*v, how to use eig() and how to confirm an eigenvector/eigenvalue and reconstruct the original A matrix.**
    d. **SVD**
    e. **What is missing is how the EigenDecomposition is calculated.**
1. **(did not read)** What is PCA?
2. **(did not read)** What is a covariance matrix?
3. **(did not read)** Variance covariance matrix
4. Visualization of the first PCA vectors, it is unclear what he is trying to show.
5. A very nice introductory tutorial on how to use PCA
6. ** An in-depth tutorial on PCA (paper)
7. ** yet another tutorial paper on PCA (looks good)
8. How to use PCA in Cross validation and for train\test split. (bottom line, do it on the train only.)
9. Another tutorial paper - looks decent

SVD

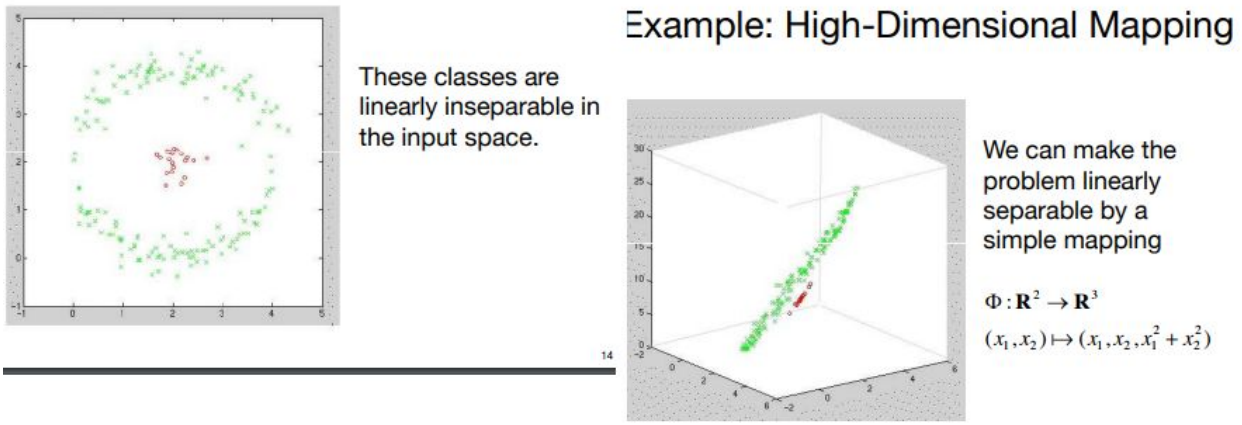
1. An explanation about SVD's formulas.

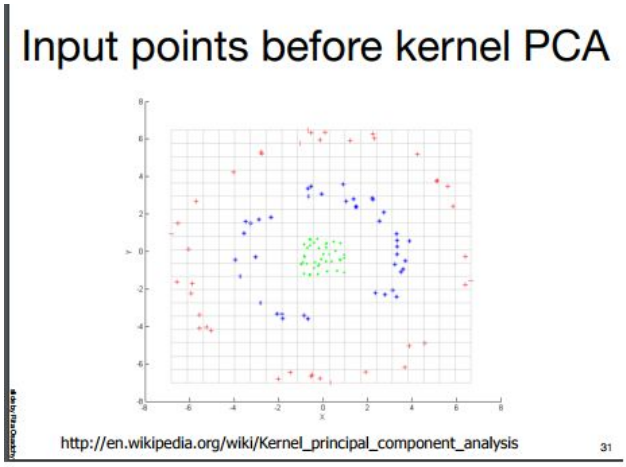
KPCA

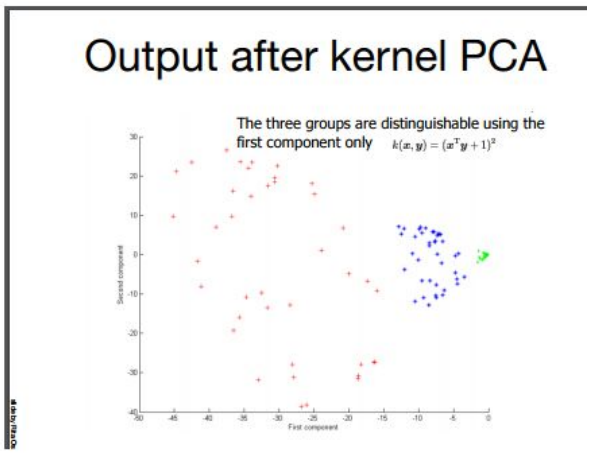
1. First they say that Autoencoder is PCA based on their equation, i.e. minimize the reconstruction error formula.



2. Then they say that PCA cant separate certain non-linear situations (circle within a circle), therefore they introduce kernel based PCA (using the kernel trick - like svm) which mapps the space to another linearly separable space, and performs PCA on it,







3. Finally, showing results how KPCA works well on noisy images, compared to PCA.

Example: De-noising images

Original data

Data corrupted with Gaussian noise

Result after linear PCA

Result after kernel PCA, Gaussian kernel

LSA

LSA is quite simple, you just use SVD to perform dimensionality reduction on the tf-idf vectors–that's really all there is to it! And LSA CLUSTERING
Note: As far as I can see LSA is PCA with tf-idf because the algorithm is intended for text.

PCA vs LSA: (intuition1, intuition2)

- reduction of the dimensionality
- noise reduction
- incorporating relations between terms into the representation.
- SVD and PCA and "total least-squares" (and several other names) are the same thing. It computes the orthogonal transform that decorrelates the variables and keeps the ones with the largest variance. There are two numerical approaches: one by SVD of the (centered) data matrix, and one by Eigen decomposition of this matrix "squared" (covariance).

(LDA) Linear discriminant analysis

(Not to be confused with the other LDA) - **Linear Discriminant Analysis (LDA)** is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. **The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting** ("curse of dimensionality") **and also reduce computational costs**.

PCA vs LDA:
Both Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are **linear transformation techniques used for dimensionality reduction**.

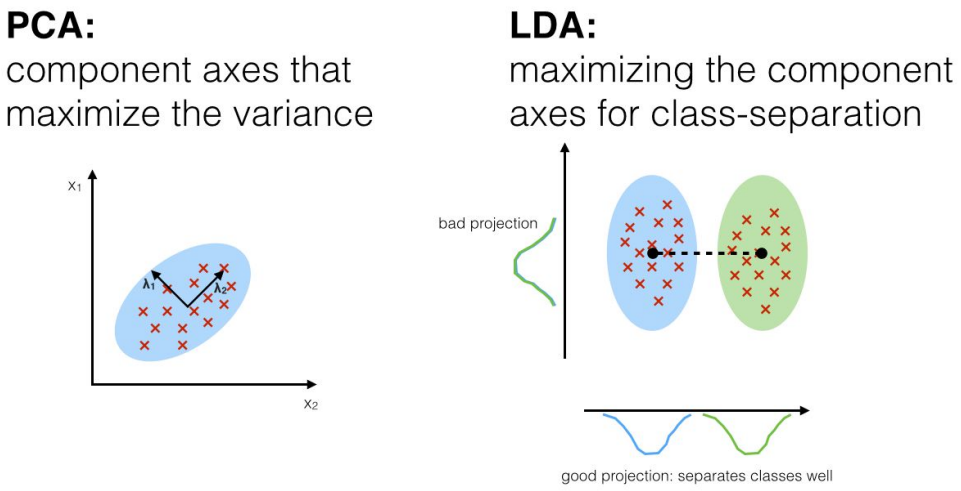
- PCA can be described as an "**unsupervised**" algorithm, since it "ignores" class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset.
- In contrast to PCA, **LDA is "supervised**" and computes the directions ("linear discriminants") that will represent the axes that maximize the separation between multiple classes.

Although it might sound intuitive that LDA is superior to PCA for a multi-class classification task where the class labels are known, this might not always the case.

For example, comparisons between classification accuracies for image recognition after using PCA or LDA show that :

- **PCA tends to outperform LDA if the number of samples per class is relatively small** (PCA vs. LDA, A.M. Martinez et al., 2001).
- **In practice, it is also not uncommon to use both LDA and PCA in combination**:

Best Practice: **PCA for dimensionality reduction can be followed by an LDA.** But before we skip to the results of the respective linear transformations, let us quickly recapitulate the purposes of PCA and LDA: PCA finds the axes with maximum variance for the whole data set where LDA tries to find the axes for best class separability. **In practice, often a LDA is done followed by a PCA for dimensionality reduction.**



**** To fully understand the details please follow the LDA link to the original and very informative article**

***** TODO: need some benchmarking for PCA\LDA\LSA\ETC..**

(not LDA) Latent Dirichlet Allocation

LDA here is a bad acronym, as it's already taken by the above algorithm!
Latent Dirichlet allocation (LDA) - This algorithm takes a group of documents (anything that is made of up text), and returns a number of topics (which are made up of a number of words) most relevant to these documents.

- LDA is an example of topic modelling
- **?- can this be used for any set of features, not just text?**

In case LDA groups together two topics, we can influence the algorithm in a way that makes those two topics separable - this is called Semi Supervised Guided LDA

ICA

1. While PCA is global, it finds global variables (with images we get eigen faces, good for reconstruction) that maximizes variance in orthogonal directions, and is not influenced by the TRANSPOSE of the data matrix.
2. On the other hand, ICA is local and finds local variables (with images we get eyes ears, mouth, basically edges!, etc), ICA will result differently on TRANSPOSED matrices, unlike PCA, its also “directional” - consider the “cocktail party” problem. On documents, ICA gives topics.
3. It helps, similarly to PCA, to help us analyze our data.

Sparse info on ICA with security returns.

MANIFOLD

Many manifold methods used to visualize high dimensional data.
Comparing manifold methods

T-SNE

1. Misreading T-SNE, this is a very important read.
2. In contrary to what it says on sklearn's website, TSNE is not suited ONLY for visualization, you can also use it for data reduction
3. "t-Distributed Stochastic Neighbor Embedding (t-SNE) is a (prize-winning) technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets."
4. Comparing PCA and TSNE, then pushing PCA to TSNE and seeing what happens (as recommended in SKLEARN
5. TSNE + AUTOENCODER example

TREES

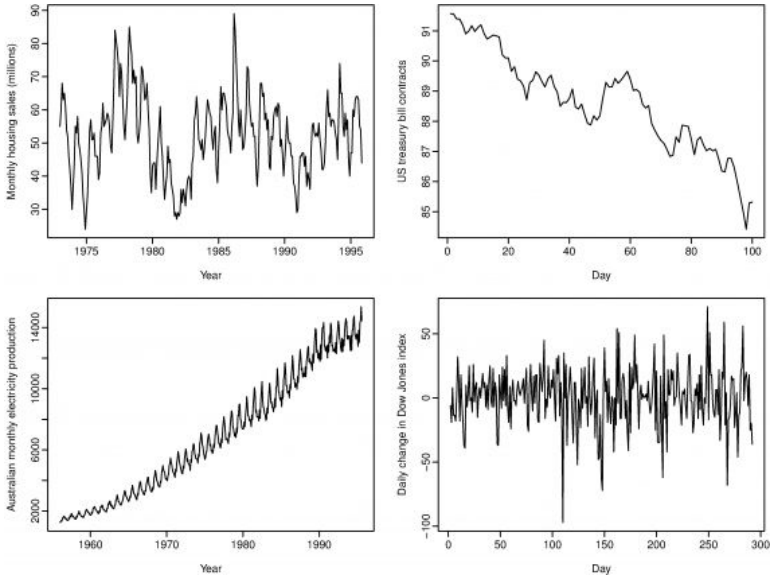
CART TREES - explains about the similarities and how to measure. which is the best split? based on SSE and GINI (good info about gini here).

- For classification the Gini cost function is used which provides an indication of how “pure” the leaf nodes are (how mixed the training data assigned to each node is). Gini = sum(pk * (1 – pk))
- Early stop - 1 sample per node is overfitting, 5-10 are good
- Pruning - evaluate what happens if the lead nodes are removed, if there is a big drop, we need it.

TIME SERIES

A great introduction into time series - **“The approach is to come up with a list of features that captures the temporal aspects so that the auto correlation information is not lost.”** basically tells us to take sequence features and create (auto)-correlated new variables using a time window, i.e., **“Time series forecasts as regression that factor in autocorrelation as well.”.** we can transform raw features into other type of features that explain the relationship in time between features. we measure success using loss functions, MAE RMSE MAPE RMSEP AC-ERROR-RATE

Interesting idea on how to define ‘time series’ dummy variables that utilize beginning\end of certain holiday events, including important information on what NOT to filter even if it seems insignificant, such as zero sales that may indicate some relationship to many sales the following day.



Time series patterns:

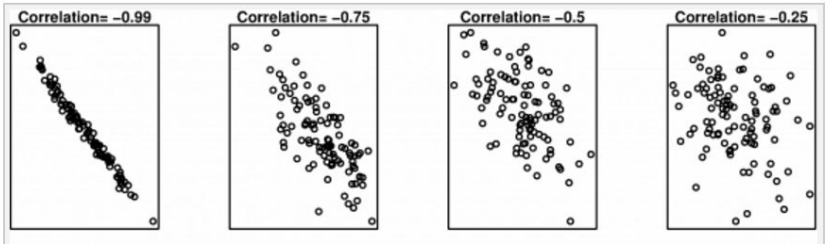
- A **trend (a,b,c)** exists when there is a long-term increase or decrease in the data.
- A **seasonal (a - big waves)** pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. The monthly sales induced by the change in cost at the end of the calendar year.
- A **cycle (a)** occurs when the data exhibit rises and falls that are not of a fixed period - **sometimes years.**

Some statistical measures (mean, median, percentiles, iqr, std dev, bivariate statistics - correlation between variables)

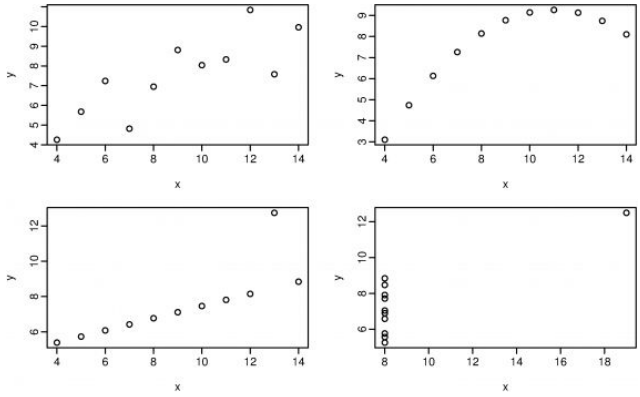
Bivariate Formula: this correlation measures the **extent of a linear relationship between two variables**. high number = high correlation between two variable. The value of **r** always lies between -1 and 1 with negative values indicating a negative relationship and positive values

indicating a positive relationship. Negative = decreasing, positive = increasing.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}},$$

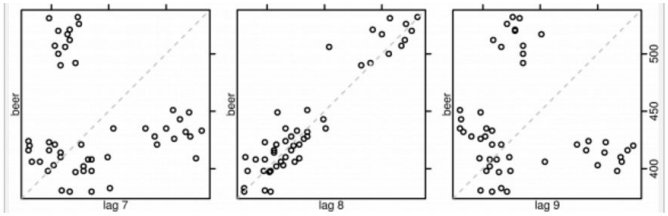


But correlation can LIE, the following has 0.8 correlation for all of the graphs:



Autocorrelation measures the linear relationship between lagged values of a time series.
L8 is correlated, and has a high measure of 0.83

- White-noise has autocorrelation of 0.



$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2},$$

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
-0.126	-0.650	-0.094	0.863	-0.099	-0.642	-0.098	0.834	-0.116

Forecasting methods:

- **Average**: Forecasts of all future values are equal to the mean of the historical data.
- **Naive**: Forecasts are simply set to be the value of the last observation.

- **Seasonal Naive**: forecast to be equal to the last observed value from the same season of the year
- **Drift:** A variation on the naïve method is to allow the forecasts to increase or decrease over time, the drift is set to be the average change seen in the historical data.

Data Transformations:

- Log
- Box cox
- Back transform
- Calendrical adjustments
- Inflation adjustment

Evaluate forecast accuracy:

$$\text{Mean absolute error: MAE} = \text{mean}(|e_i|),$$
$$\text{Root mean squared error: RMSE} = \sqrt{\text{mean}(e_i^2)}.$$
$$\text{Mean absolute percentage error: MAPE} = \text{mean}(|p_i|).$$
$$\text{sMAPE} = \text{mean}\left(200|y_i - \hat{y}_i|/(y_i + \hat{y}_i)\right).$$

- **Dummy variables:** sunday, monday, tues,wed,thurs, friday. **NO SATURDAY!**
- notice that only six dummy variables are needed to code seven categories. That is because the seventh category (in this case Sunday) is specified when the dummy variables are all set to zero. Many beginners will try to add a seventh dummy variable for the seventh category. This is known as the "dummy variable trap" because it will cause the regression to fail.
- **Outliers:** If there is an outlier in the data, rather than omit it, you can use a dummy variable to remove its effect. In this case, the dummy variable takes value one for that observation and zero everywhere else.
- **Public holidays:** For daily data, the effect of public holidays can be accounted for by including a dummy variable predictor taking value one on public holidays and zero elsewhere.
- **Easter:** is different from most holidays because it is not held on the same date each year and the effect can last for several days. In this case, a dummy variable can be used with value one where any part of the holiday falls in the particular time period and zero otherwise.
- **Trading days:** The number of trading days in a month can vary considerably and can have a substantial effect on sales data. To allow for this, the number of trading days in each month can be included as a predictor. An alternative that allows for the effects of

different days of the week has the following predictors. # Mondays in month;# Tuesdays in month;# Sundays in month.
- **Advertising**: $advertising for previous month;$advertising for two months previously

Rolling window analysis: “compute parameter estimates over a rolling window of a fixed size through the sample. If the parameters are truly constant over the entire sample, then the estimates over the rolling windows should not be too different. If the parameters change at some point during the sample, then the rolling estimates should capture this instability”

Moving average window - estimate the trend cycle
- 3-5-7-9? If its too large its going to flatten the curve, too low its going to be similar to the actual curve.
- two tier moving average, first 4 then 2 on the resulted moving average.

Visual example of ARIMA algorithm - captures the time series trend or forecast.

(the components of time series) -
1. Level. The baseline value for the series if it were a straight line.
2. Trend. The optional and often linear increasing or decreasing behavior of the series over time.
3. Seasonality. The optional repeating patterns or cycles of behavior over time.
4. Noise. The optional variability in the observations that cannot be explained by the model.

All time series have a level, most have noise, and the trend and seasonality are optional.

One step forecast using a window of “1” and a typical sample **“time, measure1, measure2”:**
- linear/nonlinear classifiers: predict a single output value - using the t-1 previous line, i.e., **“measure1 t, measure 2 t, measure 1 t+1, measure 2 t+1 (as the class)”**
- Neural networks: predict multiple output values, i.e., **“measure1 t, measure 2 t, measure 1 t+1(class1), measure 2 t+1(class2)”**

One-Step Forecast: This is where the next time step (t+1) is predicted.
Multi-Step Forecast: This is where two or more future time steps are to be predicted.

Multi-step forecast using a window of “1” and a typical sample “time, measure1”, i.e., using the current value input we label it as the two future input labels:
- **“measure1 t, measure1 t+1(class) , measure1 t+2(class1)”**

This article explains about ML Methods for Sequential Supervised Learning - Six methods that have been applied to solve sequential supervised learning problems:

1. sliding-window methods - converts a sequential supervised problem into a classical supervised problem
2. recurrent sliding windows
3. hidden Markov models
4. maximum entropy Markov models
5. input-output Markov models
6. conditional random fields
7. graph transformer networks

LTSM for time series: There are three types of gates within a unit:

- **Forget Gate**: conditionally decides what information to throw away from the block.
- **Input Gate**: conditionally decides which values from the input to update the memory state.
- **Output Gate**: conditionally decides what to output based on input and the memory of the block.

JUPYTER

%%time - measures cell time execution

(Debugging in Jupyter, how?) - put a one liner before the code and query the variables inside a function.

(how does reshape work?) - a shape of (2,4,6) is like a tree of 2->4 and each one has more leaves 4->6.
As far as i can tell, reshape effectively flattens the tree and divide it again to a new tree, but the total amount of inputs needs to stay the same. 2*4*6 = 4*2*3*2 for example
code:

```
import numpy
rng = numpy.random.RandomState(234)
a = rng.randn(2,3,10)
print(a.shape)
print(a)

b = numpy.reshape(a, (3,5,-1))
print(b.shape)
print (b)
```

***** A tutorial for Google Colaboratory - free Tesla K80 with Jup-notebook**
Jupyter on Amazon AWS
How to add extensions to jupyter: extensions

NUMPY

Fast vector calculation, a benchmark between list, map, vectorize. Vectorize wins. The idea is to use vectorize and a function that does something that may involve if conditions on a vector, and do it as fast as possible.

Pandas

Great introductory tutorial about using pandas, loading, loading from zip, seeing the table's features, accessing rows & columns, boolean operations, calculating on a whole row\column with a simple function and on two columns even, dealing with time\date parsing.

The fastest way to select rows by columns, by using masked values (benchmarked):

```
def mask_with_values(df): mask = df['A'].values == 'foo' return df[mask]
```

Accessing dataframe rows, columns and cells- by name, by index, by python methods.

How to inject headers into a headless CSV file -
Dealing with time series in pandas,

Create a new column based on a (boolean or not) column and calculation:

- Using python (map)
- Using numpy
- using a function (not as pretty)

Given a DataFrame, the **shift**() function can be used to create copies of columns that are pushed forward (rows of **NaN** values added to the front) or pulled back (rows of NaN values added to the end).

```
df['t'] = [x for x in range(10)]
df['t-1'] = df['t'].shift(1)
df['t-1'] = df['t'].shift(-1)
```

Row and column sum in pandas and numpy

NEURAL NETS

- **NN in general - 5 introductions tutorials.**
- **Segmentation examples**

MLP: fully connected, input, hidden layers, output. Gradient on the backprop takes a lot of time to calculate. Has vanishing gradient problem, because of multiplications when it reaches the first layers the loss correction is very small (0.1*0.1*01 = 0.001), therefore the early layers train slower than the last ones, and the early ones capture the basics structures so they are the more important ones.

AutoEncoder - unsupervised, drives the input through fully connected layers, sometime reducing their neurons amount, then does the reverse and expands the layer's size to get to the input (images are multiplied by the transpose matrix, many times over), Comparing the predicted output to the input, correcting the cost using gradient descent and redoing it, until the networks learns the output.

- Convolutional auto encoder
- Denoiser auto encoder - masking areas in order to create an encoder that understands noisy images
- Variational autoencoder - doesnt rely on distance between pixels, rather it maps them to a function (gaussian), eventually the DS should be explained by this mapping, uses 2 new layers added to the network. Gaussian will create blurry images, but similar. Please note that it also works with CNN.

WORD2VEC - trick based on autoencode, we keep only the hidden layer , Part 2

RBM- restricted (no 2 nodes share a connection) boltzman machine
An Autoencoder of features, tries to encode its own structure.
Works best on pics, video, voice, sensor data. 2 layers, visible and hidden, error and bias calculated via KL Divergence.

- Also known as a shallow network.
- Two layers, input and output, goes back and forth until it learns its output.

DBN - deep belief networks, similar structure to multi layer perceptron. fully connected, input, hidden(s), output layers. Can be thought of as stacks of RBM. training using GPU optimization, accurate and needs smaller labelled data set to complete the training.
Solves the 'vanishing gradient' problem, imagine a fully connected network, advancing each 2 layers step by step until each boltzman network (2 layers) learns the output, keeps advancing until finished.. Each layer learns the entire input.
Next step is to fine tune using a labelled test set, improves performance and alters the net. So basically using labeled samples we fine tune and associate features and pattern with a name. Weights and biases are altered slightly and there is also an increase in performance. Unlike CNN which learns features then high level features.

Accurate and reasonable in time, unlike fully connected that has the vanishing gradient problem.

Transfer Learning = like Inception in Tensor flow, use a prebuilt network to solve many problems that "work" similarly to the original network.

- CS course definition - also very good explanation of the common use cases:
    - Feature extraction from the CNN part (removing the fully connected layer)
    - Fine-tuning, everything or partial selection of the hidden layers, mainly good to keep low level neurons that know what edges and color blobs are, but not dog breeds or something not as general.
- CNN checkpoints for many problems with transfer learning. Has several releveant references
- Such as this "How transferable are features in deep neural networks? "
- (the indian guy on facebook) IMDB transfer learning using cnn vgg and word2vec, the word2vec is interesting, the cnn part is very informative. With python code, keras.

CNN, Convolutional Neural Net (this link explains CNN quite well, 2nd tutorial - both explain about convolution, padding, relu - sparsity, max and avg pooling):

- **Common Layers:** input->convolution->relu activation->pooling to reduce dimensionality **** ->fully connected layer
- ****repeat several times over as this discover patterns but needs another layer -> fully connected layer
- Then we connect at the end a fully connected layer (fcl) to classify data samples.
- Good for face detection, images etc.
- Requires lots of data, not always possible in a real world situation
- Relu is quite resistant to vanishing gradient & allows for deactivating neurons and for sparsity.

RNN - what is RNN by Andrej Karpathy - The Unreasonable Effectiveness of Recurrent Neural Networks, basically a lot of information about RNNs and their usage cases

- basic NN node with a loop, previous output is merged with current input. for the purpose of remembering history, for time series, to predict the next X based on the previous Y.
- 1 to N = frame captioning
- N to 1 = classification
- N to N = predict frames in a movie
- N\2 with time delay to N\2 = predict supply and demand
- Vanishing gradient is 100 times worse.
- Gate networks like LSTM solves vanishing gradient.

SNN - SELU activation function is inside not outside, results converge better.

Probably useful for feedforward networks

DEEP REINFORCEMENT LEARNING COURSE (for motion planning)or
DEEP RL COURSE (Q-LEARNING?) - using unlabeled data, reward, and probably a CNN to solve games beyond human level.

A brief survey of DL for Reinforcement learning

WIKI has many types of RNN networks (unread)

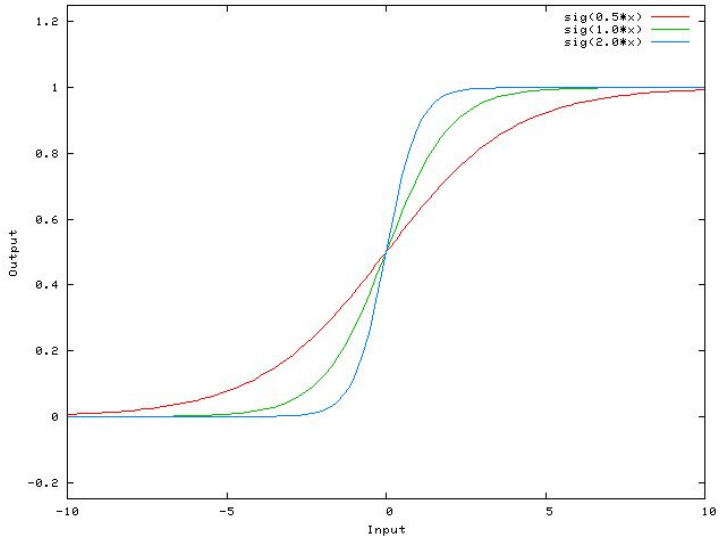
Unread and potentially good tutorials:

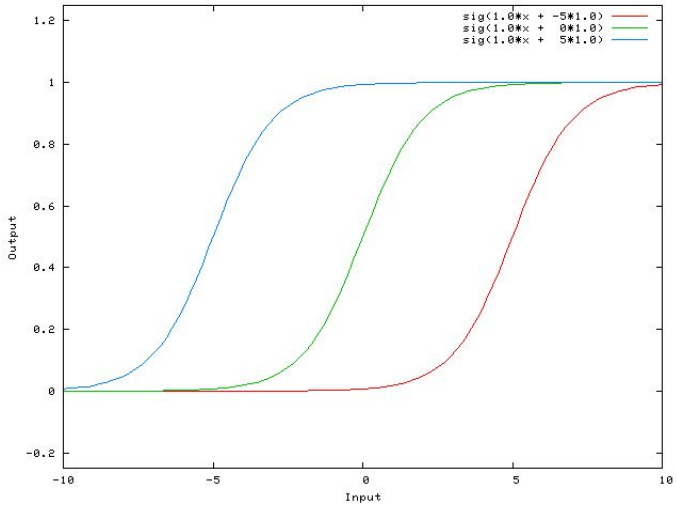
1. deep learning python

EXAMPLES of Using NN on images:
Deep image prior / denoiser/ high res/ remove artifacts/ etc..

BIAS

The role of bias in NN - similarly to the ‘b’ in linear regression.





AUTOENCODERS

1. How to use AE for dimensionality reduction + code - using keras' functional API
2. Keras.io blog post about AE's
3. Examples of vanilla, multi layer, CNN and sparse AE's
4. Another example of CNN-AE
5. Another AE tutorial
6. Hinton's coursera course on PCA vs AE, basically some info about what PCA does - maximizing variance and projecting and then what AE does and can do to achieve similar but non-linear dense representations
7. A great tutorial on how does the clusters look like after applying PCA/ICA/AE
8. Another great presentation on PCA vs AE, summarized in the KPCA section of this notebook. +another one +StackExchange

AE for anomaly detection, fraud detection

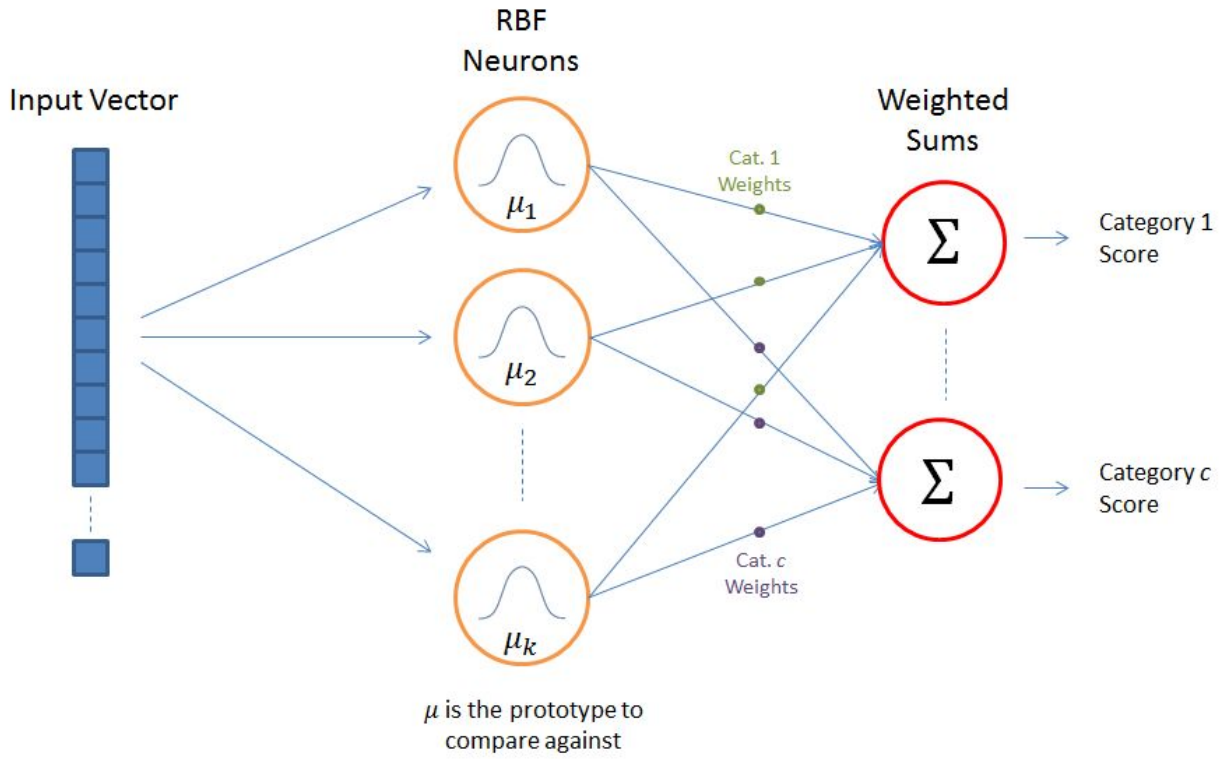
Variational AE:

1. Unread - Simple explanation
2. Pixel art VAE
3. Unread - another VAE
4. Pixel GAN VAE
5. Disentangled VAE - improves VAE

Radial Basis Function Network (RBFN) Tutorial

The RBFN approach is more intuitive than the MLP.

- An RBFN performs classification by **measuring the input's similarity to examples from the training set.**
- Each RBFN neuron **stores a "prototype**", which is just **one of the examples from the training set.**
- When we want to **classify a new input**, each neuron **computes the Euclidean distance between the input and its prototype**.
- Roughly speaking, if the input more closely resembles the class A prototypes than the class B prototypes, it is classified as class A.



Bayesian Neural Network (BNN)

BNN - (what is?) Bayesian neural network (BNN) according to Uber - **architecture that more accurately forecasts time series predictions and uncertainty estimations at scale.** "how Uber has successfully applied this model to large-scale time series anomaly detection, enabling better accommodate rider demand during high-traffic intervals."
Under the BNN framework, prediction uncertainty can be categorized into three types:

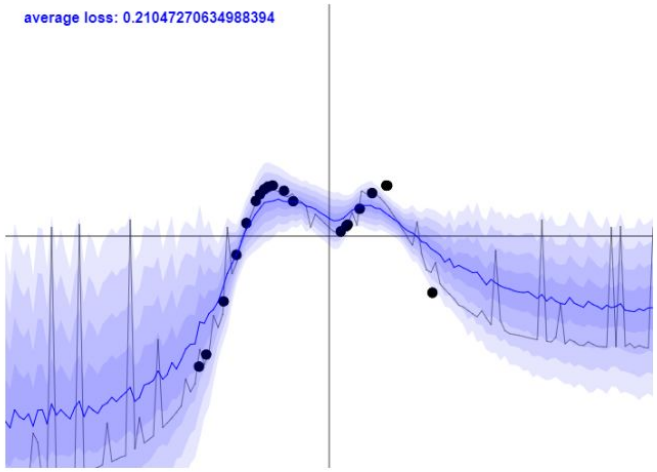
1. **Model uncertainty** captures our ignorance of the model parameters and **can be reduced as more samples are collected.**
2. **model misspecification**
3. **inherent noise** captures the uncertainty in the data generation process and **is irreducible**.

Note: in a series of articles, uber explains about time series and leads to a BNN architecture.

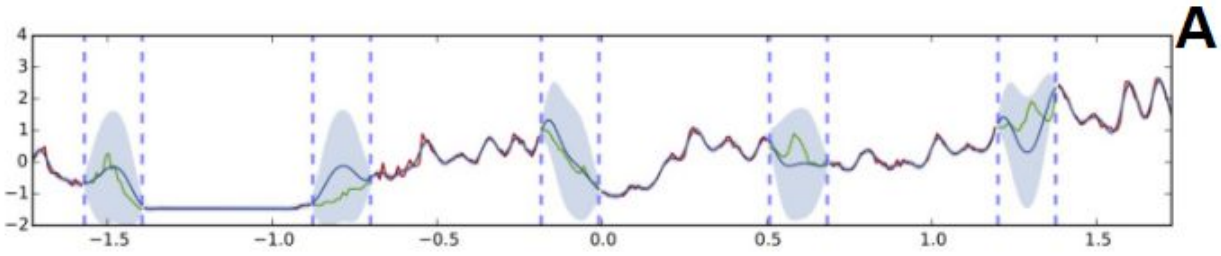
1. Neural networks - training on multi-signal raw data, training X and Y are window-based and the window size(lag) is determined in advance.
   Vanilla LSTM did not work properly, therefore an architecture of

Regarding point 1: '**run prediction with dropout 100 times**'
Why do we need a confidence measure when we have a softmax probability layer? The blog post explains, for example, that with a CNN of apples, oranges, cat and dogs, a non related example such as a frog image may influence the network to decide its an apple, therefore we can't rely on the probability as a confidence measure. The '**run prediction with dropout 100 times**' should give us a confidence measure because it draws each weight from a bernoulli distribution.
"By applying dropout to all the weight layers in a neural network, we are essentially drawing each weight from a Bernoulli distribution. In practice, this mean that we can sample from the distribution by running several forward passes through the network. This is referred to as Monte Carlo dropout."

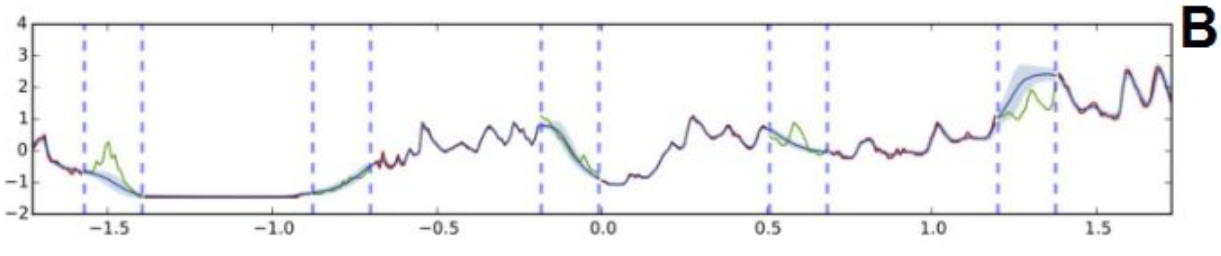
Taken from Yarin Gal's blog post . In this figure we see how sporadic is the signal from a forward pass (black line) compared to a much cleaner signal from 100 dropout passes.



Is it applicable for time series? In the figure below he tried to predict the missing signal between each two dotted lines, A is a bad estimation, but with a dropout layer we can see that in most cases the signal is better predicted.

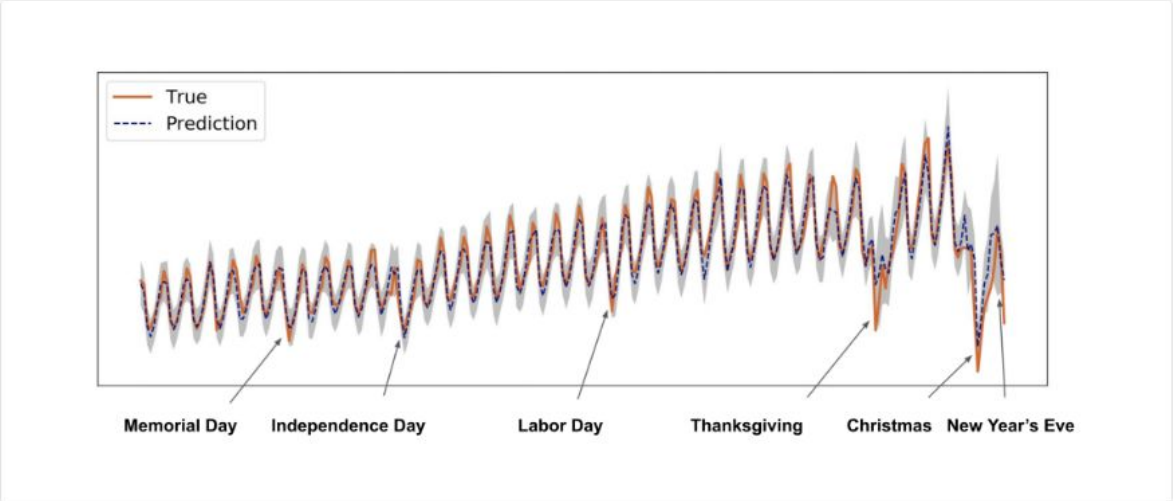


Gaussian process with SE covariance function



Dropout network using uncertainty information on the same dataset (5 hidden layers, ReLU non-linearity)

Going back to uber, they are actually using this idea to predict time series with LSTM, using encoder decoder framework.

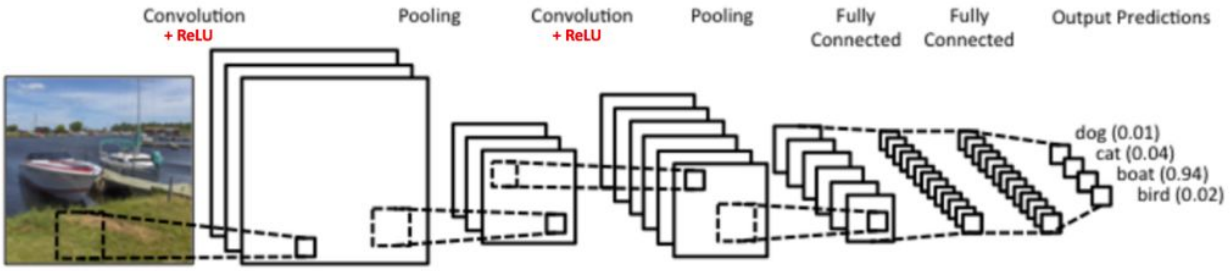


Note: this is probably applicable in other types of networks.

Phd Thesis by Yarin, he talks about uncertainty in Neural networks and using BNNs. he may have proved this thesis, but I did not read it. This blog post links to his full Phd.

Old note: The idea behind uncertainty is (paper here) that in order to trust your network's classification, you drop some of the neurons during prediction, you do this ~100 times and you average the results. Intuitively this will give you confidence in your classification and increase your classification accuracy, because only a partial part of your network participated in the classification, randomly, 100 times. **Please note that Softmax doesn't give you certainty.**

CONVOLUTIONAL NEURAL NET



([an excellent and thorough explanation about LeNet](#)) -

- **Convolution Layer** primary purpose is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.
- **ReLU** (more in the activation chapter) - The purpose of ReLU is to introduce non-linearity in our ConvNet
- **Spatial Pooling** (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.
- **Dense / Fully Connected -** a traditional Multi Layer Perceptron that **uses a softmax activation** function in the output layer to classify. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for **classifying the input image into various classes** based on the training dataset.

The overall training process of the Convolutional Network may be summarized as below:

- Step1: We **initialize** all **filters** and **parameters** / **weights** with random values
- Step2: The network **takes a single training image** as input, **goes through the forward** propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
    - Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
    - **Since weights are randomly assigned for the first training example, output probabilities are also random**.
- Step3: **Calculate the total error** at the output layer (summation over all 4 classes)
    - (L2) Total Error = $\sum$ ½ (target probability – output probability) ²
- Step4: Use **Backpropagation** to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values **to minimize the output error.**

    - **The weights are adjusted in proportion to their contribution to the total error.**
    - When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
    - This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
    - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.
- Step5: **Repeat steps 2-4 with all images in the training set.**

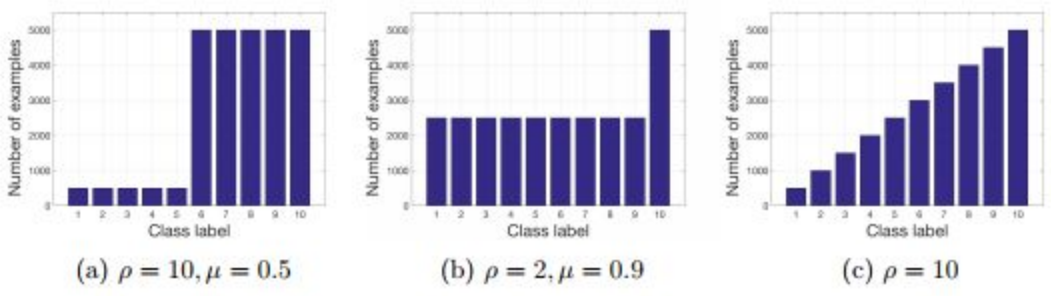
The above steps train the ConvNet – this essentially means that all the weights and parameters of the ConvNet have now been optimized to correctly classify images from the training set.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

A study that deals with class imbalance in CNN's - we systematically investigate the impact of class imbalance on classification performance of convolutional neural networks (CNNs) and compare frequently used methods to address the issue

1. Over sampling
2. Undersampling
3. Thresholding probabilities (ROC?)
4. Cost sensitive classification -different cost to misclassification
5. One class - novelty detection. This is a concept learning technique that recognizes positive instances rather than discriminating between two classes

Using several imbalance scenarios, on several known data sets, such as MNIST



(a) $\rho = 10, \mu = 0.5$ (b) $\rho = 2, \mu = 0.9$ (c) $\rho = 10$

The results indication (loosely) that oversampling is usually better in most cases, and doesn't cause overfitting in CNNs.

Graph Convolutional Networks

Explain here, with some examples

CAPSULE NEURAL NETS:

The solution to CNN's shortcomings, where features can be identified without relations to each other in an image, i.e. changing the location of body parts will not affect the classification, and changing the orientation of the image will. The promise of capsule nets is that these two issues are solved.

Recurrent Neural Net (RNN)

RNN - a basic NN node with a loop, previous output is merged with current input (using tanh?), for the purpose of remembering history, for time series - to predict the next X based on the previous Y.

(**What is RNN**?) by Andrej Karpathy - The Unreasonable Effectiveness of Recurrent Neural Networks, basically a lot of information about RNNs and their usage cases 1 to N = frame captioning

- N to 1 = classification
- N to N = predict frames in a movie
- N\2 with time delay to N\2 = predict supply and demand
- Vanishing gradient is 100 times worse.
- Gate networks like LSTM solves vanishing gradient.

(**how to initialize**?) Benchmarking RNN networks for text - don't worry about initialization, use normalization and GRU for big networks.

** Experimental improvements:
Ref - "Simplified RNN, with pytorch implementation" - changing the underlying mechanism in RNNs for the purpose of parallelizing calculation, seems to work nicely in terms of speed, not sure about state of the art results. Controversy regarding said work, author claims he already mentioned these ideas (QRNN) first, a year before, however it seems like his ideas have also been reviewed as incremental (PixelRNN). Its probably best to read all 3 papers in chronological order and use the most optimal solution.

NN-Sequence Analysis

(did not read) A causal framework for explaining the predictions of black-box sequence-to-sequence models - can this be applied to other time series prediction?

NN Most Common Usages

This is based on Francois Chollet's code sample from his book, which relies on Keras. Although probably transferable to other packages.

Layer	Type	Activation	Activation Purpose	Network Loss	N' optimizer	ref
Dense	Hidden	ReLu	Introduce non linearity, solve issues that sig\tanh have.			
Dense	Output	SoftMax	Probability, Sums to 1, **Classification** between several categories	categorical_ crossentropy	rmsprop	2.1

Troubleshooting Neural Nets

(37 reasons, 10 more) - copy pasted and rewritten here for convenience, it's pretty thorough, but long and extensive, you should have some sort of intuition and not go through all of these. The following list is has much more insight and information in the article itself.
The author of the original article suggests to turn everything off and then start building your network step by step, i.e., “a divide and conquer ‘debug’ method”.

Dataset Issues:
1. Check your input data - for stupid mistakes
2. Try random input - if the error behaves the same on random data, there is a problem in the net. Debug layer by layer
3. Check the data loader - input data is possibly broken. Check the input layer.
4. Make sure input is connected to output - do samples have correct labels, even after shuffling?
5. Is the relationship between input and output too random? - the input are not sufficiently related to the output. Its pretty amorphic, just look at the data.
6. Is there too much noise in the dataset? - badly labelled datasets.
7. Shuffle the dataset - useful to counteract order in the DS, always shuffle input and labels together.
8. Reduce class imbalance - imbalance datasets may add a bias to class prediction. Balance your class, your loss, do something.
9. Do you have enough training examples? - training from scratch? ~1000 images per class, ~probably similar numbers for other types of samples.
10. Make sure your batches don’t contain a single label - this is probably something you wont notice and will waste a lot of time figuring out! In certain cases shuffle the DS to prevent batches from having the same label.
11. Reduce batch size - This paper points out that having a very large batch can reduce the generalization ability of the model. However, please note that I found other references that claim a too small batch will impact performance.
12. Test on well known Datasets

Data Normalization/Augmentation:
12. Standardize the features - zero mean and unit variance, sounds like normalization.
13. Do you have too much data augmentation?
Augmentation has a regularizing effect. Too much of this combined with other forms of regularization (weight L2, dropout, etc.) can cause the net to underfit.
14. **Check the preprocessing of your pretrained model - with a pretrained model make sure your input data is similar in range[0, 1], [-1, 1] or [0, 255]?**
15. Check the preprocessing for train/validation/test set - CS231n points out a common pitfall: Any preprocessing should be computed ONLY on the training data, then applied to val/test

---

III. Implementation issues

16. Try solving a simpler version of the problem -divide and conquer prediction, i.e., class and box coordinates, just use one.
17. Look for correct loss “at chance” - calculat loss for chance level, i.e 10% baseline is -ln(0.1) = 2.3 Softmax loss is the negative log probability. Afterwards increase regularization strength which should increase the loss.
18. Check your custom loss function.
19. Verify loss input - parameter confusion.
20. Adjust loss weights -If your loss is composed of several smaller loss functions, make sure their magnitude relative to each is correct. This might involve testing different combinations of loss weights.
21. Monitor other metrics -like accuracy.
22. Test any custom layers, debugging them.
23. Check for “frozen” layers or variables - accidentally frozen?
24. Increase network size - more layers, more neurons.
25. Check for hidden dimension errors - confusion due to vectors ->(64, 64, 64)
26. Explore Gradient checking -does your backprop work for custon gradients? 1 2 3.

IV. Training issues

27. Solve for a really small dataset - can you generalize on 2 samples?
28. Check weights initialization - Xavier or He or forget about it for networks such as RNN.
29. Change your hyperparameters - grid search
30. Reduce regularization - too much may underfit, try for dropout, batch norm, weight, bias , L2.
31. Give it more training time as long as the loss is decreasing.
32. Switch from Train to Test mode - not clear.
33. Visualize the training - activations, weights, layer updates, biases. Tensorboard and Crayon. Tips on Deeplearning4j. Expect gaussian distribution for weights, biases start at 0 and end up almost gaussian. Keep an eye out for parameters that are diverging to +/- infinity. **Keep an eye out for biases that become very large.** This can sometimes occur in the output layer for classification if the distribution of classes is very imbalanced.
34. Try a different optimizer, Check this excellent post about gradient descent optimizers.
35. Exploding / Vanishing gradients - Gradient clipping may help. Tips on: Deeplearning4j: “A good standard deviation for the activations is on the order of **0.5 to 2.0. Significantly outside of this range may indicate vanishing or exploding activations.”**
36. Increase/Decrease Learning Rate, or use adaptive learning
37. Overcoming NaNs, big issue for RNN - decrease LR, how to deal with NaNs. evaluate layer by layer, why does it appear.

GRADIENT DESCENT

(**What are?**) batch, stochastic, and mini-batch gradient descent are and the benefits and limitations of each method.

1. Gradient descent is an **optimization algorithm** often used for finding the weights or coefficients of machine learning algorithms, such as artificial neural networks and logistic regression.
2. the model makes predictions on training data, then **use the error on the predictions to update the model to reduce the error.**
3. The goal of the algorithm is to find model parameters (e.g. coefficients or weights) that **minimize the error of the model on the training dataset**. It does this by making changes to the model that move it along a gradient or slope of errors down toward a minimum error value. This gives the algorithm its name of "gradient descent."

Stochastic:

- calculate error and updates the model **after every training sample**

Batch:

- calculates the error for each example in the training dataset, but only updates the model **after all training examples have been evaluated.**

Mini batch (most common):

- splits the training dataset into small batches, used to calculate model error and update model coefficients.
- Implementations may choose to sum the gradient over the mini-batch or take the average of the gradient (reduces variance of gradient) **(unclear?)**

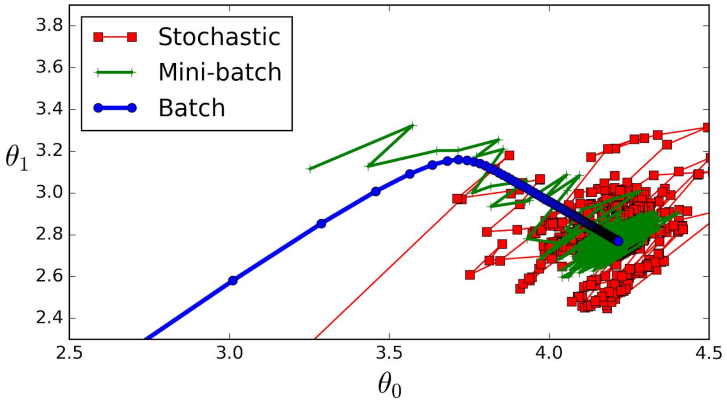
+ Tips on how to choose and train using mini batch in the link above

So what is a batch size in NN (another source) - and how to find the "right" number. In general terms a good mini bach between 1 and all samples is a good idea. Figure it out empirically.

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

How to balance and what is the tradeoff between batch size and the number of iterations.



GD with Momentum - explain

Python

How to use better OOP in python.

Deep Learning for NLP

- (did not fully read) Yoav Goldberg's course syllabus with lots of relevant topics on DL4NLP, including bidirectional RNNS and tree RNNs.
- (did not fully read) CS224d: Deep Learning for Natural Language Processing, with slides etc.

Deep Learning using Linear Support Vector Machines - 1-3% decrease in error by replacing the softmax layer with a linear support vector machine

SCIKIT LEARN

- Awesome code examples about using svm\knn\naive\log regression in sklearn in python, i.e., “fitting a model onto the data”

Jupytr Notebooks

[Importing a notebook as a module]

KERAS

A make sense introduction into keras, has several videos on the topic, going through many network types, creating custom activation functions, going through examples.
+ Two extra videos from the same author, examples and examples-2

Didn't read:

1. Keras cheatsheet
2. Seq2Seq RNN
3. Stateful LSTM - Example script showing how to use stateful RNNs to model long sequences efficiently.
4. CONV LSTM - this script demonstrate the use of a conv LSTM network, used to predict the next frame of an artificially generated move which contains moving squares.

How to force keras to use tensorflow and not teano (set the .bat file)

Callbacks - how to create an AUC ROC score callback with keras - with code example.

Batch size vs. Iterations in NN \ Keras.

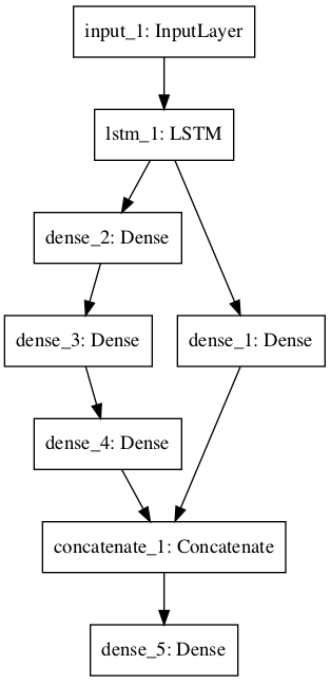
Keras metrics - classification regression and custom metrics
Keras Metrics 2 - accuracy, ROC, AUC, classification, regression r^2.

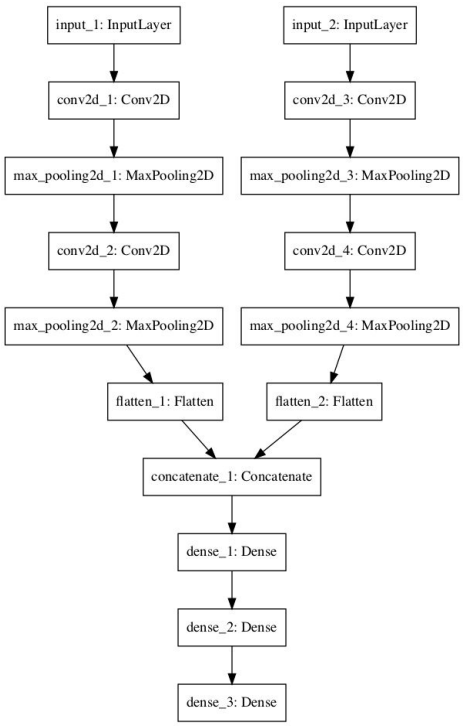
Introduction to regression models in Keras, using MSE, comparing baseline vs wide vs deep networks.

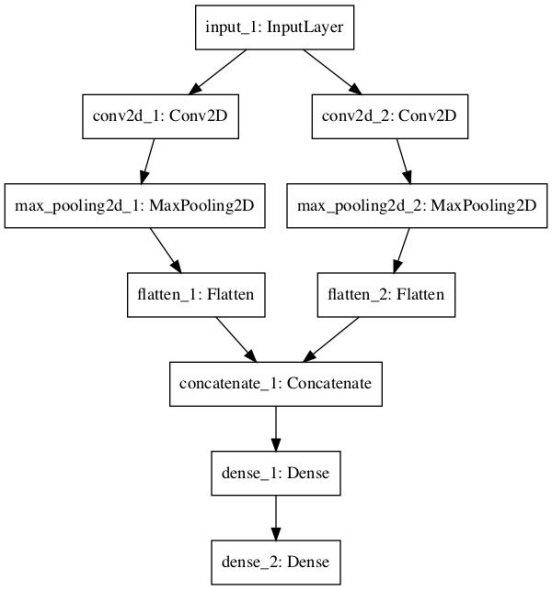
How does Keras calculate accuracy? Formula and explanation
Compares label with the rounded predicted float, i.e. bigger than 0.5 = 1, smaller than = 0
For categorical we take the argmax for the label and the prediction and compare their location.
In both cases, we average the results.

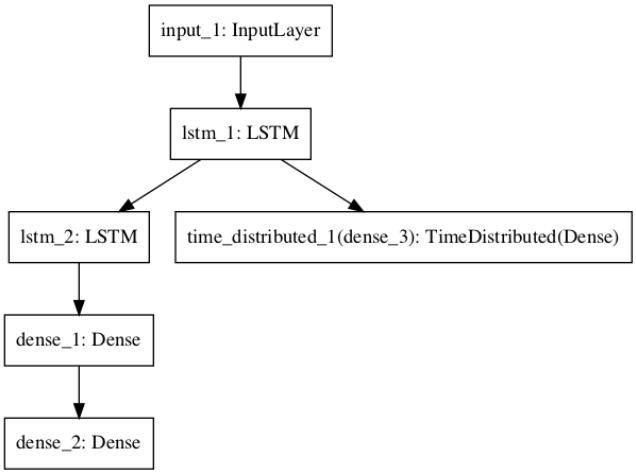
KERAS FUNCTIONAL API

What is and how to use? A flexible way to declare layers in parallel, i.e. parallel ways to deal with input, feature extraction, models and outputs as seen in the following images.









Keras: Predict vs Evaluate

here:

.predict() generates output predictions based on the input you pass it (for example, the predicted characters in the MNIST example)

.evaluate() computes the loss based on the input you pass it, along with any other metrics that you requested in the metrics param when you compiled your model (such as accuracy in the MNIST example)

Keras metrics

For classification methods - how does keras calculate accuracy, all functions.

LOSS

Very Basic advice: You should probably switch train/validation repartition to something like 80% training and 20% validation. In most cases it will improve the classifier performance overall (more training data = better performance)

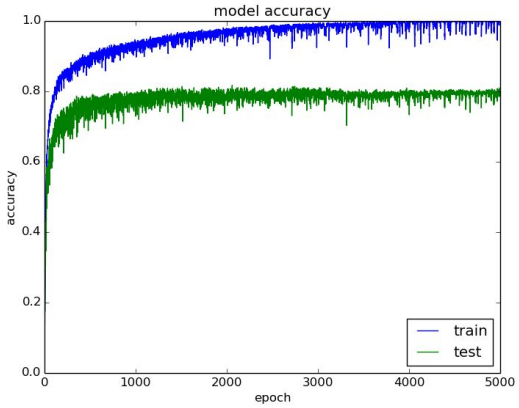
+If Training error and test error are too close (your system is unable to overfit on your training data), this means that your model is too simple. Solution: more layers or more neurons per layer.

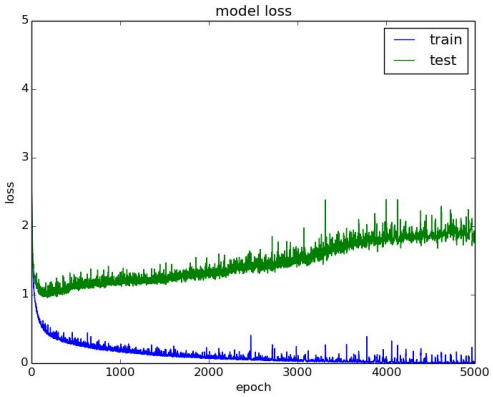
Early stopping: If you have never heard about "early-stopping" you should look it up, it's an important concept in the neural network domain : https://en.wikipedia.org/wiki/Early_stopping . To summarize, the idea behind early-stopping is to stop the training once the validation loss starts plateauing. Indeed, when this happens it almost always mean you are starting to overfitt your classifier. The training loss value in itself is not something you should trust, beacause it will continue to increase event when you are overfitting your classifier.

With cross entropy there can be an issue where the accuracy is the same for two cases, one where the loss is decreasing and the other when the loss is not changing much.

GOLD class	predicted prob	predicted class (Thres=0.5)	loss	predicted prob	predicted class (Thres=0.5)	loss
1	0.6	1	0.510825624	0.92	1	0.083381609
1	0.62	1	0.478035801	0.99	1	0.010050336
1	0.58	1	0.544727175	0.89	1	0.116533816
1	0.55	1	0.597837001	0.91	1	0.094310679
0	0.48	0	0.653926467	0.01	0	0.010050336
0	0.39	0	0.494296322	0.2	0	0.223143551
0	0.44	0	0.579818495	0.1	0	0.105360516
0	0.39	0	0.494296322	0.07	0	0.072570693
		Accuracy= 100%			Accuracy= 100%	

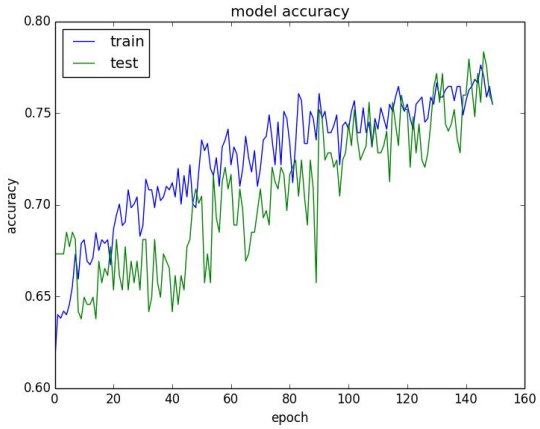
[How to read LOSS graphs (and accuracy on top)]

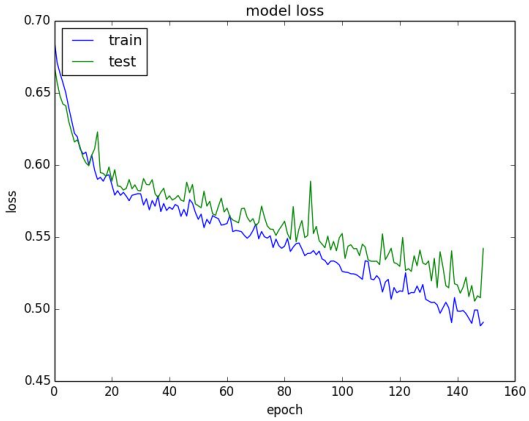




This indicates that the model is overfitting. It continues to get better and better at fitting the data that it sees (training data) while getting worse and worse at fitting the data that it does not see (validation data).

[This is a very good example of a train/test loss and an accuracy behavior.]





LOSS IN KERAS

Why is the training loss much higher than the testing loss? A Keras model has two modes: training and testing. Regularization mechanisms, such as Dropout and L1/L2 weight regularization, are turned off at testing time.
The training loss is the average of the losses over each batch of training data. Because your model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss.

LSTM IN KERAS

LSTM - what is?

- Paper - all LSTMs variants are pretty much the same performance.

Unread - sentiment classification of IMDB movies using Keras and LSTM

- **Very important** - how to interpret LSTM neurons in keras

LSTM for time-series - (jakob) single point prediction, sequence prediction and shifted-sequence prediction with code.

Stateful vs Stateless: crucial for understanding how to levrage LSTM networks:

1. A good description on what it is and how to use it.
2. ML mastery
3. ML mastery 2
4. Philippe remy on stateful vs stateless, intuition mostly with code, but not 100% clear

Machine Learning mastery:
A good tutorial on LSTM: important notes:
1. **Scale to -1,1,** because the internal activation in the lstm cell is ***tanh***.
2.stateful - True, needs to reset internal states, False =stateless. Great info & results HERE, with seeding, with training resets (and not) and predicting resets (and not) - **note:** empirically matching the shampoo input, network config, etc.
Another explanation/tutorial about stateful lstm, should be thorough.
3. what is return_sequence, return_states, and how to use each one and both at the same time. Return_sequence is needed for stacked LSTM layers.
4.stacked LSTM - each layer has represents a higher level of abstraction in TIME!

Keras Input shape - a good explanation about differences between input_shape, dim, and what is. Additionally about layer calculation of inputs and output based on input shape, and sequence model vs API model.

Benchmarking LSTM variants: - it looks like LSTM and GRU are competitive to mutation (i believe its only in pytorch) adding a bias to LSTM works **(a bias of 1 as recommended in the paper)**, but generally speaking there is no conclusive empirical evidence that says one type of network is better than the other for all tests, but the mutated networks tend to win over lstm\gru variants.
BIAS 1 in keras - **unit_forget_bias**: Boolean. If True, add 1 to the bias of the forget gate at initializationSetting it to true will also force `bias_initializer="zeros"`. This is recommended in Jozefowicz et al.

Validation_split arg - The validation split variable in Keras is a value between [0..1]. Keras proportionally split your training set by the value of the variable. The first set is used for training and the 2nd set for validation after each epoch.
This is a nice helper add-on by Keras, and most other Keras examples you have seen the training and test set was passed into the fit method, after you have manually made the split. The value of having a validation set is significant and is a vital step to understand how well your model is training. Ideally on a curve you want your training accuracy to be close to your validation curve, and the moment your validation curve falls below your training curve the alarm bells should go off and your model is probably busy over-fitting.
Keras is a wonderful framework for deep learning, and there are many different ways of doing things with plenty of helpers.

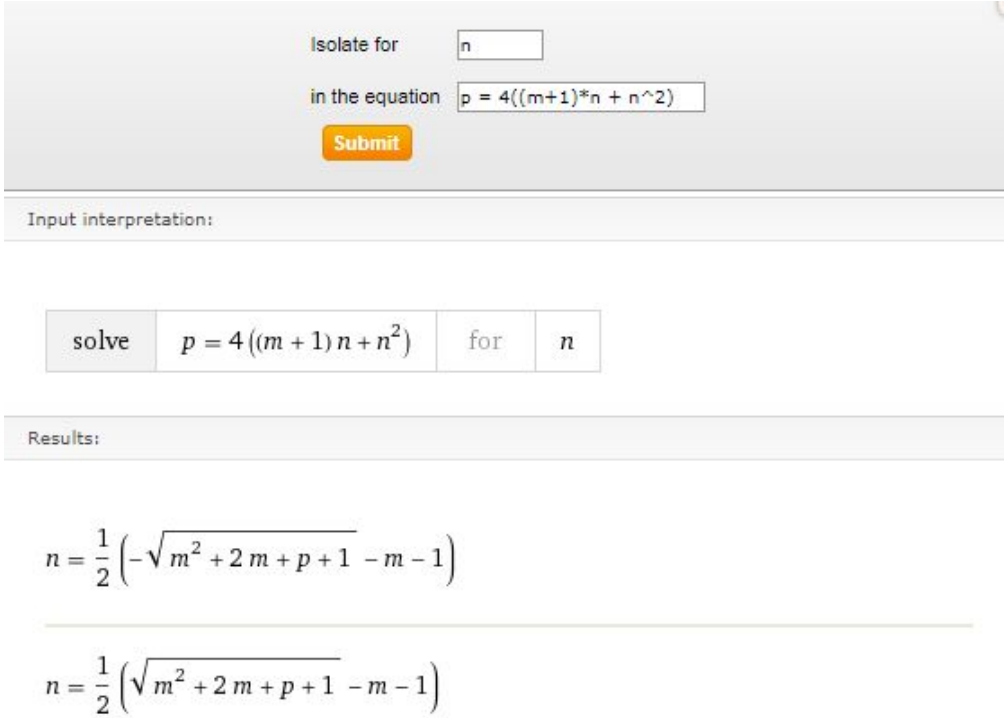
Return_sequence: unclear.

Sequence.pad_sequences - using maxlength it will either pad with zero if smaller than, or truncate it if bigger.

Using batch size for LSTM in Keras

Imbalanced classes? Use class_weights, another explanation here about class_weights and sample_weights.

number of units in LSTM
Calculate how many params are in an LSTM layer?



Understanding timedistributed in Keras, but with focus on lstm one to one, one to many and many to many - here the timedistributed is applying a dense layer to each output neuron from the lstm, which returned_sequence = true for that purpose.
This tutorial clearly shows how to manipulate input construction, lstm output neurons and the target layer for the purpose of those three problems (1:1, 1:m, m:m).

UNSUPERVISED LSTM

1. Paper, paper2, paper3
2. In keras

GRU

A tutorial about GRU

RECURRENT WEIGHTED AVERAGE (RNN-WA)

What is? (a type of cell that converges to higher accuracy faster than LSTM.
it implements attention into the recurrent neural network:
1. the keras implementation is available at https://github.com/keisuke-nakata/rwa
2. the whitepaper is at https://arxiv.org/pdf/1703.01253.pdf

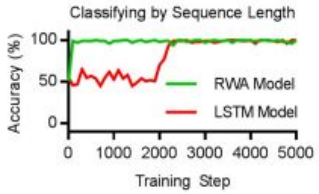
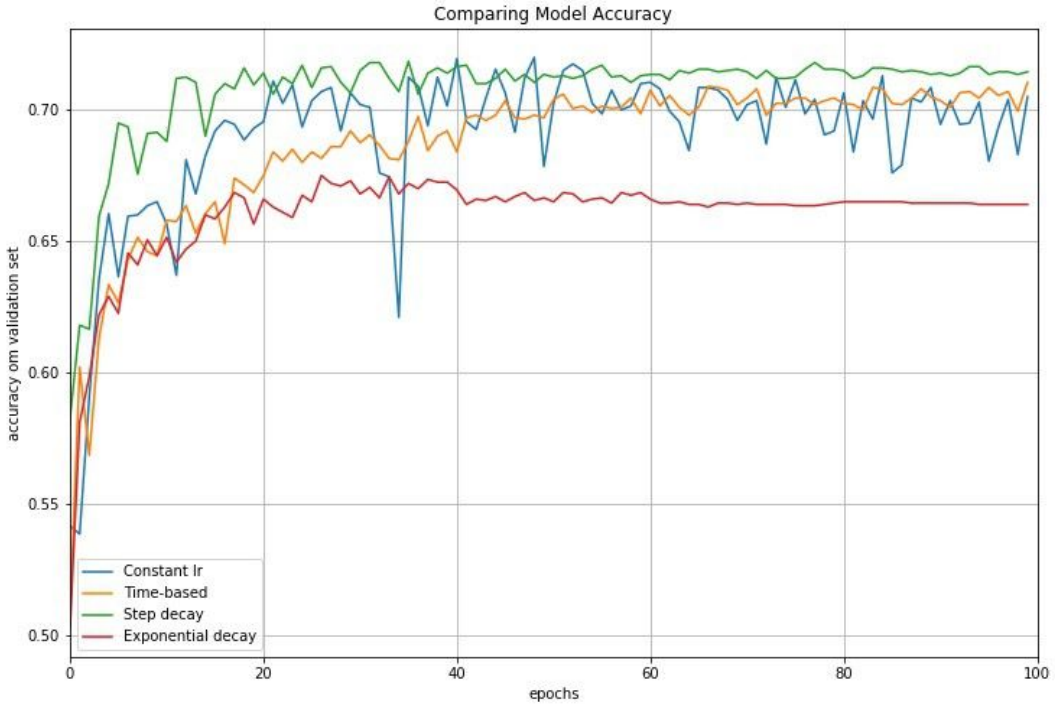


Figure 3: Plot comparing the performance of the RWA and LSTM models when classifying sequences by length. The models must determine if a sequence is longer than 500 symbols, which is half the maximum possible sequence length. Each sequence is populated with numbers drawn at random from a unit normal distribution. The traces show the accuracy of each model on the test data while the models are being fitted to the training data. The RWA model achieves a classification accuracy near 100% before the LSTM model.

LEARNING RATE REDUCTION

Intro to Learning Rate methods - what they are doing and what they are fixing in other algos.
Callbacks, especially ReduceLROnPlateau - this callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.
Cs123 (very good): explains about many things related to CNN, but also about LR and adaptive methods.

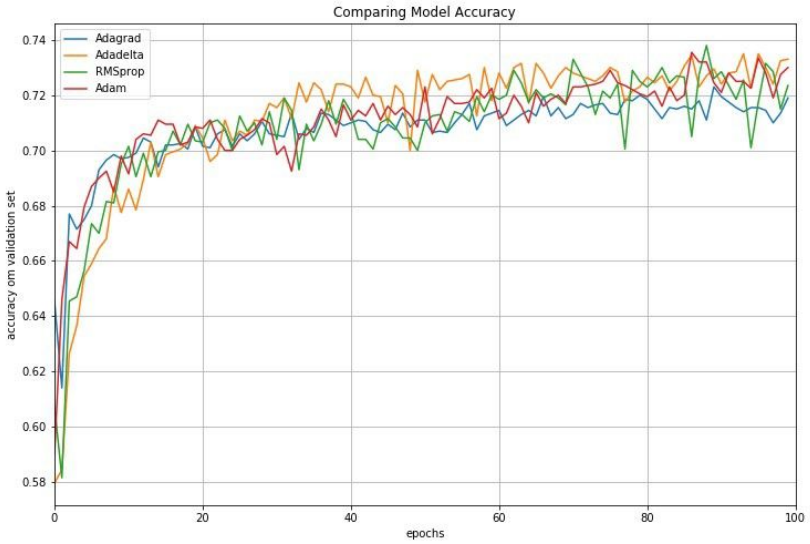
An excellent comparison of several learning rate schedule methods and adaptive methods:
(same here but not as good)



Adaptive gradient descent algorithms such as Adagrad, **Adadelta**, RMSprop, Adam, provide an alternative to classical **SGD**.
These per-parameter learning rate methods provide **heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually.**

1. **Adagrad** performs larger updates for more sparse parameters and smaller updates for less sparse parameter. It has good performance with sparse data and training large-scale neural network. However, its **monotonic learning rate usually proves too aggressive and stops learning too early** when training deep neural networks.
2. **Adadelta** is an extension of Adagrad that seeks to **reduce its aggressive, monotonically decreasing learning rate.**

3. **RMSprop** adjusts the Adagrad method in a very **simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate**.
4. **Adam** is an update to the RMSProp optimizer which is like **RMSprop with momentum**.



adaptive learning rate methods demonstrate better performance than learning rate schedules, and they require much less effort in hyperparamater settings

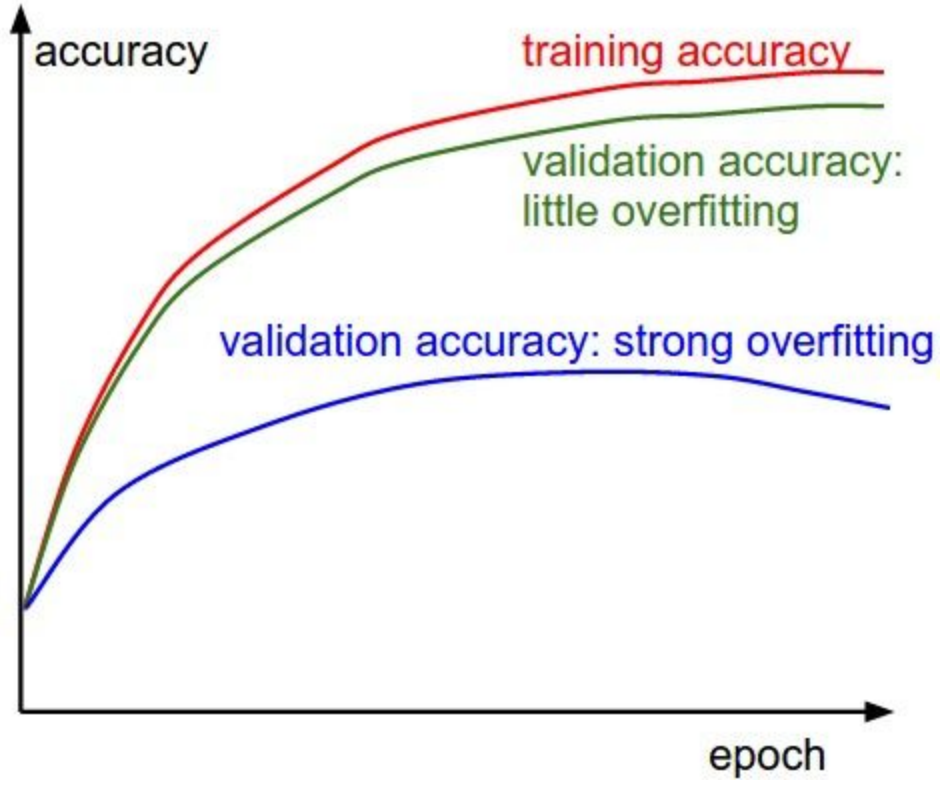
Recommended paper: practical recommendation for gradient based DNN

Another great comparison - pdf paper and webpage link -

- if your input data is sparse, then you likely achieve the best results using one of the adaptive learning-rate methods.
- An additional benefit is that you will not need to tune the learning rate but will likely achieve the best results with the default value.
- In summary, RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. It is identical to Adadelta, except that Adadelta uses the RMS of parameter updates in the numerator update rule. Adam, finally, adds bias-correction and momentum to RMSprop. Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. [10] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice

Train/Val accuracy in NEURAL NETS

The second important quantity to track while training a classifier is the validation/training accuracy. This plot can give you valuable insights into the amount of overfitting in your model:



- The gap between the training and validation accuracy indicates the amount of overfitting.
- Two possible cases are shown in the diagram on the left. The blue validation error curve shows very small validation accuracy compared to the training accuracy, indicating strong overfitting (note, it's possible for the validation accuracy to even start to go down after some point).
- NOTE: When you see this in practice you probably want to increase regularization:
    - **stronger L2 weight penalty**
    - **Dropout**
    - **collect more data**.
- The other possible case is when the validation accuracy tracks the training accuracy fairly well. This case indicates that your model capacity is not high enough: **make the model larger by increasing the number of parameters.**

OPTIMIZERS IN KERAS and in general

Documentation about optimizers

- SGD can be fine tuned
- For others Leave most parameters as they were

DROPOUT LAYERS IN KERAS AND GENERAL

A very influential paper about dropout and how beneficial it is - bottom line always use it.

OPEN QUESTIONs:

1. **does a dropout layer improve performance even if an lstm layer has dropout or recurrent dropout.**
2. **What is the diff between a separate layer and inside the lstm layer.**
3. **What is the diff in practice and intuitively between drop and recurrentdrop**

Dropout layers in keras, or dropout regularization:

- Dropout is a technique where randomly selected neurons are ignored RANDOMLY during training.
- contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.
- As a neural network learns, neuron weights settle into their context within the network.
- Weights of neurons are tuned for specific features providing some specialization. **Neighboring neurons become to rely on this specialization,** which if taken too far can result in a fragile model too specialized to the training data. (overfitting)
- **This reliant on context for a neuron during training is referred to complex co-adaptations.**
- After dropout, **other neurons will have to step in and handle the representation required to make predictions for the missing neurons,** which is believed to result in multiple independent internal representations being learned by the network.
- **Thus, the effect of dropout is that the network becomes less sensitive to the specific weights of neurons**.
- This in turn leads to a network with better generalization capability and less likely to overfit the training data.

Another great answer about drop out -

- as a consequence of the 50% dropout, the neural network will learn different, redundant representations; **the network can't rely on the particular neurons and the combination** (or interaction) of these to be present.
- Another nice side effect is that training will be faster.
- Rules:
    - Dropout is only applied during training,
    - Need to rescale the remaining neuron activations. E.g., if you set 50% of the activations in a given layer to zero, you need to scale up the remaining ones by a factor of 2.
    - if the training has finished, you'd use the complete network for testing (or in other words, you set the dropout probability to 0).

Implementation of drop out in keras is “inverse dropout” - n the Keras implementation, the output values are corrected during training (by dividing, in addition to randomly dropping out the values) instead of during testing (by multiplying). This is called "inverted dropout". Inverted dropout is functionally equivalent to original dropout (as per your link to Srivastava's paper), with a nice feature that the network does not use dropout layers at all during test and prediction. This is explained a little in this Keras issue.

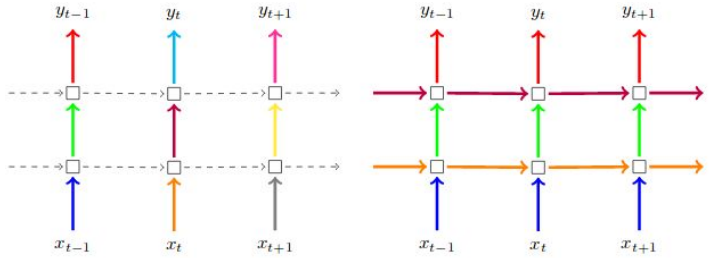
Dropout notes and rules of thumb aka “best practice” -

- dropout value of 20%-50% of neurons with 20% providing a good starting point. (A probability too low has minimal effect and a value too high results in underlearning by the network.)
- Use a large network for better performance, i.e., when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.
- Use dropout on VISIBLE AND HIDDEN. Application of dropout at each layer of the network has shown good results.
- **Unclear ?** Use a large learning rate with decay and a large momentum. Increase your learning rate by a factor of 10 to 100 and use a high momentum value of 0.9 or 0.99.
- **Unclear ?** Constrain the size of network weights. A large learning rate can result in very large network weights. Imposing a constraint on the size of network weights such as max-norm regularization with a size of 4 or 5 has been shown to improve results.

Difference between LSTM ‘dropout’ and ‘recurrent_dropout’ - vertical vs horizontal.

I suggest taking a look at (the first part of) this paper. Regular dropout is applied on the inputs and/or the outputs, meaning the vertical arrows from `x_t` and to `h_t`. In you add it as an argument to your layer, it will mask the inputs; you can add a Dropout layer after your recurrent layer to mask the outputs as well. Recurrent dropout masks (or "drops") the connections between the recurrent units; that would be the horizontal arrows in your picture.

This picture is taken from the paper above. On the left, regular dropout on inputs and outputs. On the right, regular dropout PLUS recurrent dropout:



BIDIRECTIONAL LSTM

(what is?) Wiki - The basic idea of BRNNs is to connect two hidden layers of opposite directions to the same output. By this structure, the output layer can get information from past and future states.

BRNN are especially useful when the context of the input is needed. For example, in handwriting recognition, the performance can be enhanced by knowledge of the letters located before and after the current letter.

Another explanation- It involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second.
.. It allows you to specify the merge mode, that is how the forward and backward outputs should be combined before being passed on to the next layer. The options are:

- **'sum'**: The outputs are added together.
- **'mul'**: The outputs are multiplied together.
- **'concat'**: The outputs are concatenated together (the default), providing double the number of outputs to the next layer.
- **'ave'**: The average of the outputs is taken.

The default mode is to **concatenate**, and this is the method often used in studies of bidirectional LSTMs.

Another simplified example

Lets say we try to predict the next word in a sentence, on a high level what a unidirectional LSTM will see is

> The boys went to ....

And will try to predict the next word only by this context, with bidirectional LSTM you will be able to see information further down the road for example

Forward LSTM:

> The boys went to ...

Backward LSTM:

> ... and then they got out of the pool

You can see that using the information from the future it could be easier for the network to understand what the next word is.

Batch size

(a good read) about batch sizes in keras, specifically LSTM, read this first!
A sequence prediction problem makes a good case for a varied batch size as you may want to have a batch size equal to the training dataset size (batch learning) during training and a batch size of 1 when making predictions for one-step outputs.

power of 2: have some advantages with regards to vectorized operations in certain packages, so if it's close it might be faster to keep your batch_size in a power of 2.

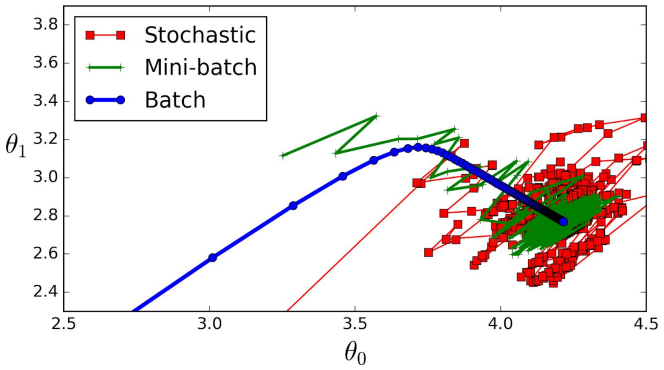
(pushing batches of samples to memory in order to train) -
Batch size defines number of samples that going to be propagated through the network.
For instance, let's say you have 1050 training samples and you want to set up batch_size equal to 100. Algorithm takes first 100 samples (from 1st to 100th) from the training dataset and trains network. Next it takes second 100 samples (from 101st to 200th) and train network again. We can keep doing this procedure until we will propagate through the networks all samples. The problem usually happens with the last set of samples. In our example we've used 1050 which is not divisible by 100 without remainder. The simplest solution is just to get final 50 samples and train the network.

Advantages:

- It requires less memory. Since you train network using less number of samples the overall training procedure requires less memory. It's especially important in case if you are not able to fit dataset in memory.
- Typically networks trains faster with mini-batches. That's because we update weights after each propagation. In our example we've propagated 11 batches (10 of them had 100 samples and 1 had 50 samples) and after each of them we've updated network's parameters. If we used all samples during propagation we would make only 1 update for the network's parameter.

Disadvantages:

- **The smaller the batch the less accurate estimate of the gradient.** In the figure below you can see that mini-batch (green color) gradient's direction **fluctuates** compare to the full batch (blue color).



Small batch size has an effect on validation accuracy.

I am running VGG16 on Cats vs Dogs (Lesson 1) on my laptop, with a small NVIDIA Quadro K1100M GPU, and due to its less memory I have been experimenting with different (smaller) batch sizes. Here is what I have got:

batch size	validation accuracy
2	0.88
4	0.97
8	0.98

i.e. it seems that validation accuracy benefits from bigger batch size. Could you elaborate a little on this?

IMPORTANT: batch size in '.prediction' is needed for some models, only for technical reasons as seen here, in keras.

1. (unread) about mini batches and performance.
2. (unread) tradeoff between bath size and number of iterations

Another observation, probably empirical - **to answer your questions on Batch Size and Epochs:**
In general: Larger batch sizes result in faster progress in training, but don't always converge as fast. Smaller batch sizes train slower, but *can* converge faster. It's definitely problem dependent.

In general, the models improve with more epochs of training, to a point. They'll start to plateau in accuracy as they converge. Try something like 50 and plot number of epochs (x axis) vs. accuracy (y axis). You'll see where it levels out.

INITIALIZERS

XAVIER GLOROT:

Why's Xavier initialization important?

In short, it helps signals reach deep into the network.

- If the weights in a network start too small, then the signal shrinks as it passes through each layer until it's too tiny to be useful.
- If the weights in a network start too large, then the signal grows as it passes through each layer until it's too massive to be useful.

Xavier initialization makes sure the weights are 'just right', keeping the signal in a reasonable range of values through many layers.

To go any further than this, you're going to need a small amount of statistics - specifically you need to know about random distributions and their variance.

When to use glorot uniform-over-normal initialization?

However, i am still not seeing anything empirical that says that glorot surpesses everything else under certain conditions (except the glorot paper), most importantly, does it really help in LSTM where the vanishing gradient is ~no longer an issue?

He-et-al Initialization

This method of initializing became famous through a paper submitted in 2015 by He et al, and is similar to Xavier initialization, with the factor multiplied by two. In this method, the weights are initialized keeping in mind the size of the previous layer which helps in attaining a global minimum of the cost function faster and more efficiently.

w=np.random.randn(layer_size[l],layer_size[l-1])*np.sqrt(2/layer_size[l-1])

OPTIMIZERS

There are several optimizers, each had his 15 minutes of fame, some optimizers are recommended for CNN, Time Series, etc..

There are also what I call ‘experimental’ optimizers, it seems like these pop every now and then, with or without a formal proof. It is recommended to follow the literature and see what are the ‘supposedly’ state of the art optimizers atm.

Backstitch - September 17 - supposedly an improvement over SGD for speech recognition using DNN. **Note: it wasnt tested with other datasets or other network types.**

(how does it work?) **take a negative step back, then a positive step forward**. I.e., When processing a minibatch, instead of taking a single SGD step, we first take a step with −α times the current learning rate, for α > 0 (e.g. α = 0.3), and then a step with 1 + α times the learning rate, with the same minibatch (and a recomputed gradient). So we are taking a small negative step, and then a larger positive step. This resulted in quite large improvements – around 10% relative improvement [37] – for our best speech recognition DNNs. **The recommended hyper parameters are in the paper.**

Drawbacks: takes twice to train, momentum not implemented or tested, dropout is mandatory for improvement, slow starter.

ACTIVATION FUNCTIONS

(a bunch of observations, seems like a personal list) -

- Output layer - linear for regression, softmax for classification
- Hidden layers - hyperbolic tangent for shallow networks (less than 3 hidden layers), and ReLU for deep networks

ReLU - The purpose of ReLU is to **introduce non-linearity**, since most of the real-world data we would want our network to learn would be nonlinear (e.g. convolution is a linear operation – element wise matrix multiplication and addition, so we account for nonlinearity by introducing a nonlinear function like ReLU, e.g here - search for ReLU).

- Relu is quite **resistant to vanishing gradient** & allows for deactivating neurons and for sparsity.
- Other nonlinear functions such as tanh or sigmoid can also be used instead of ReLU, but **ReLU has been found to perform better in most situations**.

1. Visual + description of activation functions
2. A very good explanation + figures about activations functions

TENSOR FLOW

Reduce_sum on GPU is not deterministic - workaround includes a new “transpose * one_vec” and inclusion of bias vector in weight matrix calculation in the back prop.

Performance measurements

Precision \ Recall \ ROC \ AUC - **Performance Measures:**

A balanced confusion matrix is better than one that is either one row of numbers and one of zeros, or a column of numbers and a column of zeros. Therefore an algorithm that outputs a lower classification accuracy but has a better confusion matrix wins.

# of Positive predictions divided by the total number of positive class values predicted.
Precision = True Positives / (True Positives + False Positives)
Low can be thought of many false positives.

# of positive predictions divided by the number of positive class values in the test data
Recall (sensitivity) = True Positives / (True Positives + False Negatives)
Low can be thought of many false negatives.

F1_Score = 2 * ((Precision * Recall) / (Precision + Recall))
F1 helps select a model based on a balance between precision and recall.

- **Accuracy** = (1 – Error) = (TP + TN)/(PP + NP) = Pr(C), the probability of a correct classification.
- **Sensitivity** (recall) = TP/(TP + FN) = TP/PP = the ability of the test to detect disease in a population of diseased individuals.
- **Specificity** = TN/(TN + FP) = TN / NP = the ability of the test to correctly rule out the disease in a disease-free population.

(What are ?) Sensitivity and specificity against ROC and AUC.

ROC curve and AUC in weka - explains how the curve should look like for the negative or positive predictions, against what is actually plotted.

RECALL, PRECISION AND F1

Best explanation ever :

- Recall:
  - one day, your girlfriend asks you: 'Sweetie, do you remember all birthday surprises from me?'
  - This simple question makes your life in danger. To extend your life, you need to **recall** all **10** surprising events from your memory.
  - So, `recall` is the ratio of a number of **events you can *correctly* recall** to a number of **all correct events**. If you can recall all **10** events correctly, then, your recall ratio is **1.0** (**100%**). If you can recall 7 events correctly, your recall ratio is **0.7** (**70%**).
- Precision:
  - For example, you answers **15** times, **10** events are correct and **5** events are wrong. This means you can recall all events but it's not so `precise`.
  - So, `precision` is the ratio of a number of **events you can *correctly* recall** to a number **all events you recall** *(mix of correct and wrong recalls)*. In other words, it is how precise of your recall.
  - From the previous example (10 real events, 15 answers: 10 correct answers, 5 wrong answers), you get **100%** recall but your precision is only **66.67%** (10 / 15).

Confusion matrix wise: bottom line is recall (**% correct out of positive cases**), right column is precision (% of POS predictions) & **% accuracy in diagonal**

	Predicted Negative	Predicted Positive
Negative Cases	TN: 9,760	FP: 140
Positive Cases	FN: 40	TP: 60

F1 score:

- conveys the balance between the precision and the recall
- 2*((precision*recall)/(precision+recall)

Yet another(pretty good) source
Another (bad) source for explaining, preɔision, recall, accuracy, true positive rate etc.

(**How to use precision and recall**?) answer by aurelien geron:

- In a binary classifier, the decision function is the function that produces a score for the positive class.
- In a logistic regression classifier, that decision function is simply a linear combination of the input features.
- If that score is greater than some threshold that you choose, then the classifier "predicts" the positive class, or else it predicts the negative class.
- **If you want your model to have high precision (at the cost of a low recall), then you must set the threshold pretty high. This way, the model will only predict the positive class when it is absolutely certain. For example, you may want this if the classifier is selecting videos that are safe for kids: it's better to err on the safe side.**
- **Conversely, if you want high recall (at the cost of a low precision) then you must use a low threshold. For example, if the classifier is used to detect intruders in a nuclear plant, then you probably want to detect all actual intruders, even if it means getting a lot of false alarms (called "false positives").**
- If you make a few assumptions about the distribution of the data (i.e., the positive and negative class are separated by a linear boundary plus Gaussian noise), then computing the logistic of the score gives you the probability that the instance belongs to the positive class. A score of 0 corresponds to the 50% probability. So by default, a LogisticClassifier predicts the positive class if it estimates the probability to be greater than 50%. **In general, this sounds like a reasonable default threshold, but really it all depends on what you want to do with the classifier.**
- If the assumptions I mentioned above were perfect, then if the Logistic Classifier outputs a probability of X% for an instance, it means there is exactly X% chance that it's positive. But in practice, the assumptions are imperfect, so I try to always make it clear that we are talking about an "estimated probability", not an actual probability.

(RMSE - what is?) - it is important to recall that RMSE has the same unit as the dependent variable (DV). It means that there is no absolute good or bad threshold, however you can define it based on your DV. For a datum which ranges from **0 to 1000, an RMSE of 0.7 is small,** but if the range goes from **0 to 1, it is not that small anymore.** However, although the smaller the RMSE, the better,

(R^2 vs RMSE) - R-squared is conveniently scaled between 0 and 1, whereas RMSE is not scaled to any particular values. This can be good or bad; obviously R-squared can be more easily interpreted, but with RMSE we explicitly know how much our predictions deviate, on average, from the actual values in the dataset. So in a way, RMSE tells you more.
I also found this video really helpful.

Kappa - ?

References:

1. A Survey on Deep Learning in Medical Image Analysis

YOUTUBE COURSES

- DEEPNET.TV YOUTUBE (excellent)
- Mitchel ML Lectures (too long)
- Quoc Les (google) wrote DNN tutorials and 3H video (not intuitive)
- KDnuggets: numpy, panda, scikit, tutorials.
- Deep learning online book (too wordy)
- Genetic Algorithms - grid search hyper params better than brute force.. obviously
-
- CNN tutorial
- Introduction to programming in scikit
- SVM in scikit python
- Sklearn scipy PCA tutorial
- RNN
- Matrix Multiplication - linear algebra

Deep learning Course:

Kadenze - deep learning tensor flow - Histograms for (Image distribution - mean distribution) / std dev, are looking quite good.

Machine Learning Course:

Week1: Introduction Lesson 4: Supervised, unsupervised.
Lesson 6: model regression, cost function
Lesson 71: optimization objective, large margin classification
PCA at coursera #1
PCA at coursera #2
PCA #3
SVM at coursera #1 - simplified

Predictive Analytics Course:

Syllabus
Week 2: Lesson 29: supervised learning
Lesson 36: From rules to trees
Lesson 43: overfitting, then validation, then accuracy
Lesson 46: bootstrap, bagging, boosting, random forests.
Lesson 52: NN
Lesson 55: Gradient Descent
Lesson 59: Logistic regression, SVM, Regularization, Lasso, Ridge regression
Lesson 64: gradient descent, stochastic, parallel, batch.

Unsupervised: Lesson X K-means, DBscan

Containers

PMML - an XML file that describes a ML model that is transferrable between applications.

- PMML uses XML to represent mining models.
- The structure of the models is described by an XML Schema.
- One or more mining models can be contained in a PMML document

Docker -

-
-

PLOTLY in python

How to use plotly in python

Recommender Algorithms

1. Collaborative filtering, SVD
2. Spotlight, item2vec, Neural nets for Recommender systems
3.

Reinforcement Learning

A review paper about RL in DL