January 2022

AI&DM EXAM REVIEW

By Gil Sasson

Taken from Prof. Avi Rosenfeld's lecture notes, Harvard's Introduction to Data Science lecture by Pavlos Protopapas, Kevin Rader and Chris Tanner, Stanford's lecture notes by Andrew Ng, Neural Networks and Deep Learning by Michael Nielsen, and Wikipedia

1) SQL

SQL keyword	Description
SELECT	select data from a database. Example: SELECT * FROM Customers WHERE Country='Mexico'; (*) means all columns.
FROM	specifies which table to select or delete data from
WHERE	filters a result set to include only records that fulfill a specified condition
ORDER BY	sorts the result set in ascending or descending order. Example: SELECT * FROM Customers ORDER BY CustomerName; To sort the records in descending order, use the DESC keyword. ASC for ascending order
INSERT INTO	used to insert new rows in a table. Example: INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway')
DELETE	used to delete existing records in a table. Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted! Syntax: DELETE FROM tablename WHERE condition

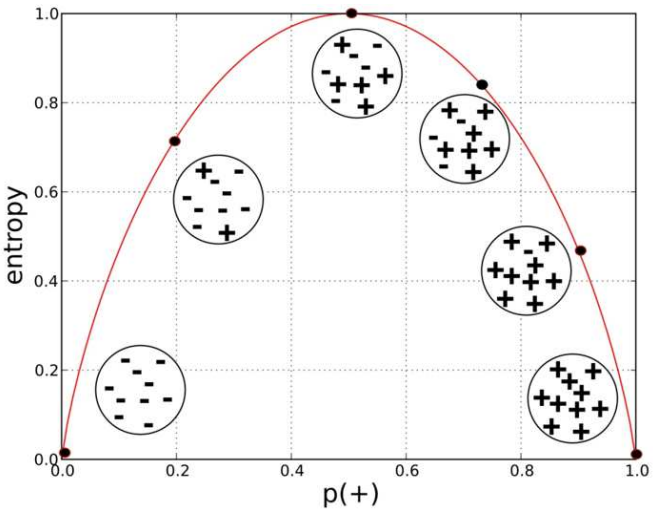
SQL Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
AND	Logical and
OR	Logical or
LIKE	Search for a pattern. Example: SELECT * FROM Customers WHERE CustomerName LIKE 'a%'
IN	allows you to specify multiple values in a WHERE clause

Note that 'a%' searches for any string that starts with 'a'.
In the same manner, '%a%' searches for any string that include 'a'.
'_' (underscore) is also useful in SQL, for example:
'_' Represents a single character for example, h_t may find hot, hat, and hit

2) Decision Tree (Supervised Learning) and Measures

Entropy:

Entropy is a measure of disorder or uncertainty and the goal of machine learning models and Data Scientists in general is to reduce uncertainty.



- Low entropy is better
- 0 entropy can be either all '-' (minuses) or all '+' (pluses)
- Entropy of 1 happens when the groups are exactly mixed
- Information Gain (IG) is how much we learned from an attribute and is defined

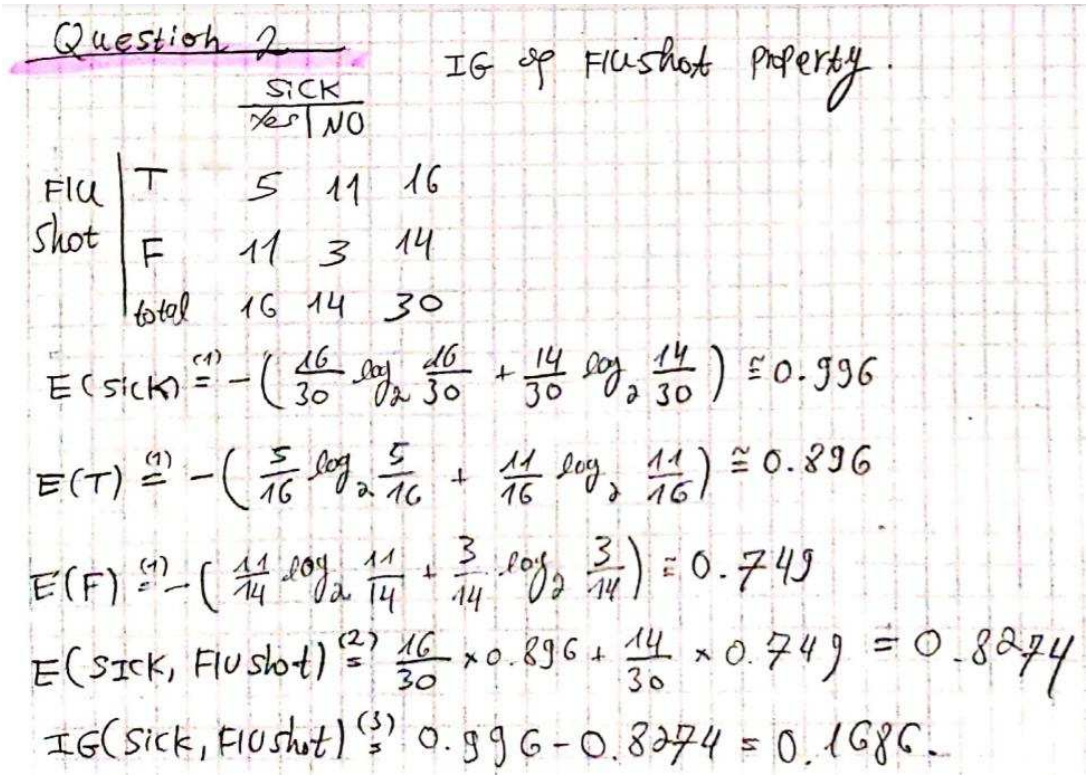
$$Entropy\,(T) = -\sum_{i=1}^{n} P_i \cdot log_2(P_i) \tag{1}$$

$$Entropy\,(T, X) = \sum_{c \in X} P\,(c) \cdot Entropy\,(c) \tag{2}$$

$$IG\,(T, X) = Entropy\,(T) - Entropy\,(T, X) \tag{3}$$

Where T is the target value and X is an attribute.

The following is an example of IG computation from homework 2



What does it means when some feature A has a higher IG than some feature B?

- Feature A reduces more disorder in our target variable than feature B

- Decision tree would use this result to make the first split on our data using feature A
- The decision tree algorithm would use this process at every split to decide what feature it is going to split on next
- With more than two features the first split is made on the most informative feature and then at every split the information gain for each additional feature needs to be recomputed because it would not be the same as the information gain from each feature by itself
- Decision tree would repeat this process as it grows deeper and deeper till either it reaches a pre-defined depth or no additional split can result in a higher information gain beyond a certain threshold which can also usually be specified as a hyper-parameter

Imbalanced Classification:

An imbalanced classification problem is an example of a classification problem where the distribution of examples across the known classes is biased or skewed and therefore it creates an overfitting problem.
For example if we have many categories with only one value (one measurement, for instance), then the Entropy will be 0 because

$$Entropy\,(category) \overset{(1)}{=} -\frac{1}{1} \cdot log_2\left(\frac{1}{1}\right) = 0$$

That creates an overfitting problem is not good for classification.

How to deal with Imbalanced Classification? One solution is Gain Ratio.

Gain Ratio:

a modification of the information gain that reduces its bias on high-branch attributes

- Large when data is evenly spread
- Small when all data belongs to one branch
- 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
- Importance of attribute decreases as intrinsic information gets large
- When values are evenly distributed between k groups then Information Gain = Gain Ratio
- When values are evenly distributed between 2 groups intrinsic information = 1
- When values are evenly distributed between 3 groups intrinsic information = 1.5
- When values are evenly distributed between 4 groups intrinsic information = 2

$$GainRatio = \frac{Gain\,(S, A)}{IntrinsicInfo\,(S, A)} \tag{4}$$

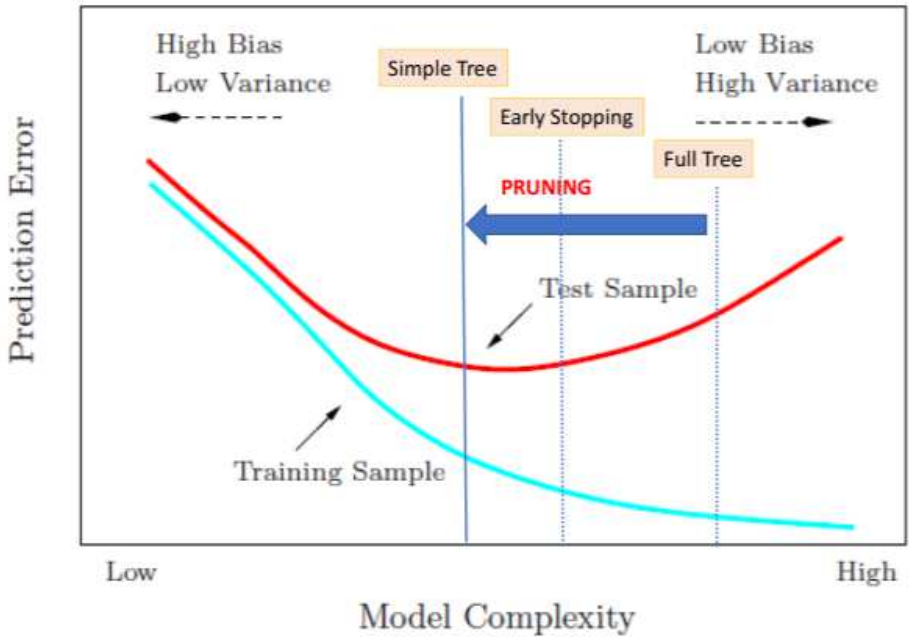
$$IntrinsicInfo = -\sum \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|} \tag{5}$$

Another criterion for a decision tree is Gini

$$Gini\,(P_1, P_2, ..., P_n) = 1 - \sum_i P_i^2 \tag{6}$$

Where P_i is the probability that an instance is of class i.
Like entropy, Gini index increases as n increases. Given binary classification, $Gini = 2P\,(1 - P)$
The algorithm is CART. Parabola with value of 0 at P=0,1 and with maximum value of $\frac{1}{2}$ at $P = \frac{1}{2}$.

How to avoid overfitting in a decision tree algorithm? Avoid overfitting by pruning (the removal or reduction of parts of a tree) or limiting the depth of the tree and using Cross-Validation and EarlyStopping. Large trees have high variance and are prone to overfitting. A simpler model with fewer parameters have a higher bias but lower variance and may help in avoiding overfitting.



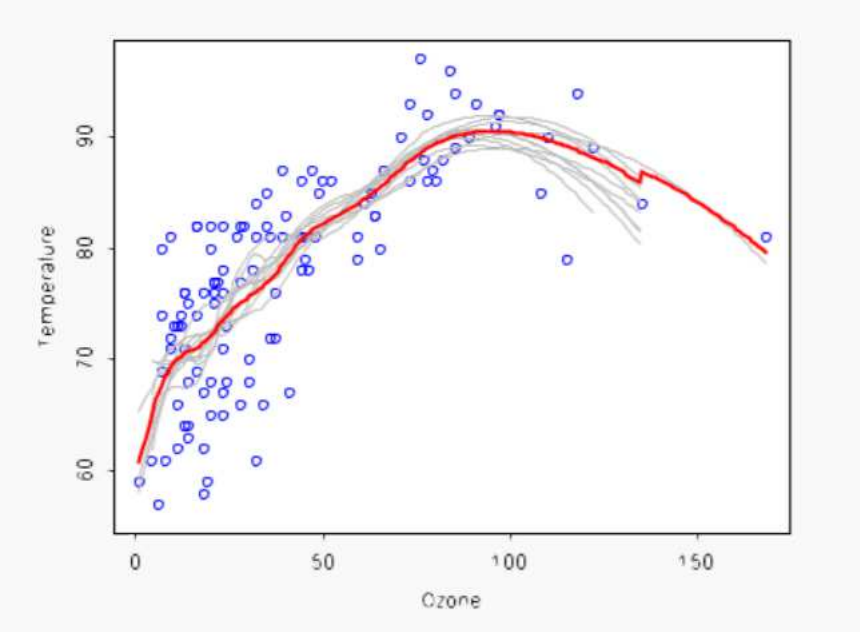
Bagging:
One way to adjust for the high variance of the output of an experiment is to perform the experiment multiple times and then average the results. We can generate multiple samples of training data, via bootstrapping. We train a full decision tree on each sample of data. For classification, we return the class that is outputted by the plurality of the models. For regression we return the average of the outputs for each tree. **Benefits of Bagging**:

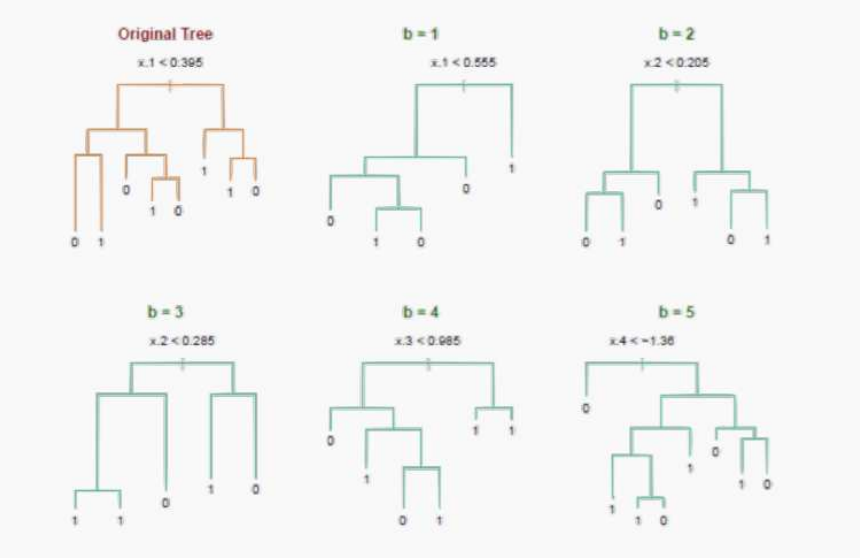
- High expressiveness - by using full trees each model is able to approximate complex functions

and decision boundaries

- Low variance - averaging the prediction of all the models
- Bagging improves prediction accuracy at the expense of interpretability
- Reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees

Here are some examples of Bagging:





Do you see any problems? Still some overfitting if the trees are too large. If trees are too shallow it can still underfits.

Bagging drawbacks One can no longer trace the 'logic' of an output through a series of decisions based on predictor values.

Random Forest:

Random Forest is a modified form of bagging that creates ensembles of independent decision trees. It trains on different parts of the data to avoid overfitting in deep trees. **How it works?**

- Train each tree on a separate bootstrap sample of the full training set (same as in bagging)
- For each tree, at each split, we randomly select a set of k predictors
- From amongst the k predictors, we select the optimal predictor and the optimal corresponding threshold for the split from the full set of predictors. **Hyper-parameters**:
  - The number of predictors to randomly select at each split
  - The total number of trees in the ensemble
  - The minimum leaf node size

  When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly because in each split, the chances of selected a relevant predictor will be low and hence most trees in the ensemble will be weak models.
  Some notes:
  - Increasing the number of trees in the ensemble generally does not increase the risk of overfitting
  - If the number of trees is too large, then the trees in the ensemble may become more correlated, increase the variance

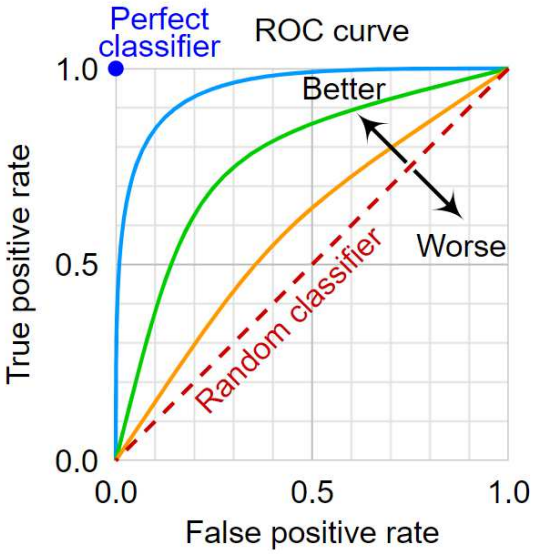
Forests are like the pulling together of decision tree algorithm efforts. Taking the teamwork of many trees thus improving the performance of a single random tree. Though not quite similar, forests give the effects of a K-fold cross validation.

Evaluating Categorical Learning - Recall, Precision, F-measure:

		Predicted by the model	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Measure	Formula
Precision	TP / (TP + FP) TN / (TN + FN)
Recall	TP / (TP + FN) TN / (TN + FP)
Accuracy	(TP + TN) / (TP + TN + FT + FN)
F-Measure (F1 score)	2*precision*recall/ precision+recall

ROC Curve:
The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the curve (AUC) is the probability when given one randomly selected positive instance and one randomly selected negative instance, the classifier will be able to tell which one is which.



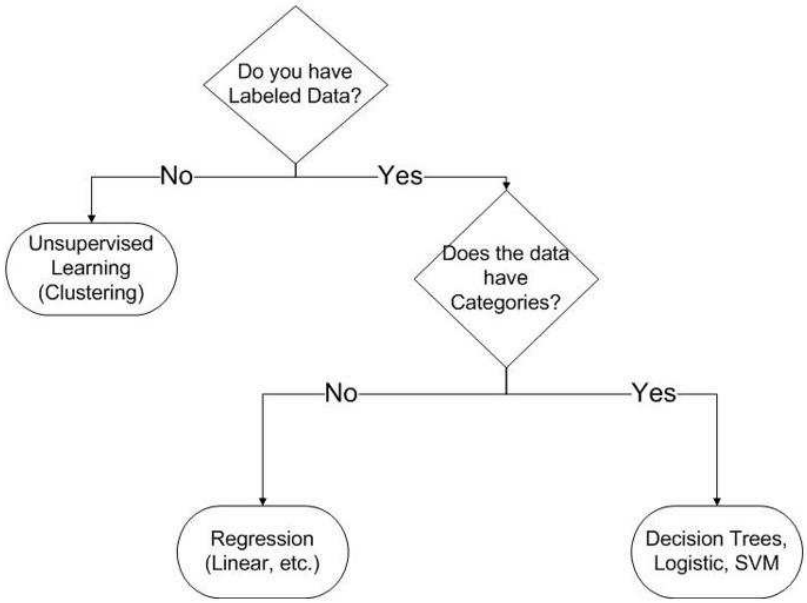
How to read large Confusion Matrices:

Predicted Values

Actual Values	Mantled Howler	Patas Monkey	Bald Uakari	Japanese Macque	Pygmy Marmoset	White Headed Capuchin	Silvery Marmoset	Common Squirrel Monkey	Black Headed Night Monkey	Nilgiri Langur
Mantled Howler	True Negative	False Positive								
Patas Monkey	False Negative	True Positive								
Bald Uakari										
Japanese Macque										
Pygmy Marmoset										
White Headed Capuchin										

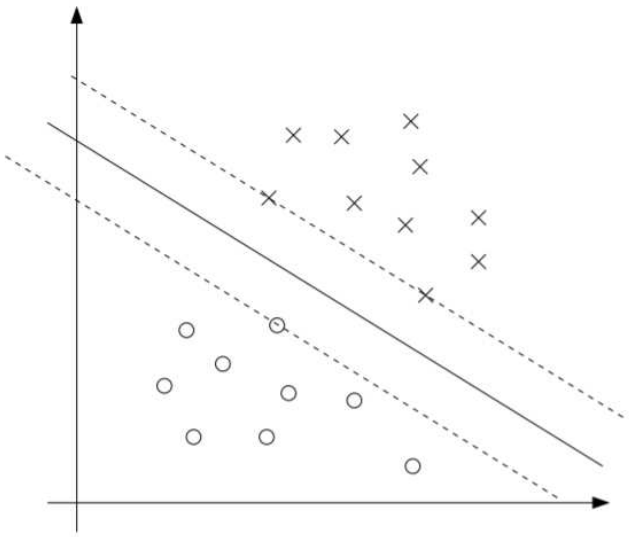
3) Bias and Variance

There are three types of machine-learning:



- Categorical Supervised Learning
  - Decision Trees, Logistic Regression, SVM, linear regression, Random Forest, kNN, Neural Networks
- Numerical Supervised Learning
  - SVM, linear regression, Random Forest, kNN, Neural Networks
- Unsupervised Learning
  - K-Means, Neural Networks

Support Vector Machine (SVM):
Support Vector Machine (SVM) learning algorithm. SVMs are among the best (and many believe are indeed the best) "off-the-shelf" supervised learning algorithms. The idea is to try to find a decision boundary that maximizes the (geometric) margin.



The three points with the smallest margins are exactly the ones closest to the decision boundary (one circle, two crosses) are called the support vectors. The number of support vectors can be much smaller than the size the training set.

Gradient Descent:
Estimate of the regression coefficients:

- Start from a random point

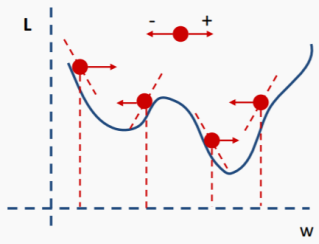
  1. Determine which direction to go to reduce the loss (left or right).
  2. Compute the slope of the function at this point and step to the right if slope is negative or step to the left if slope is positive.
  3. Go to #1.

- In Linear Regression (we will talk about later), there is no local minima because the loss function is convex

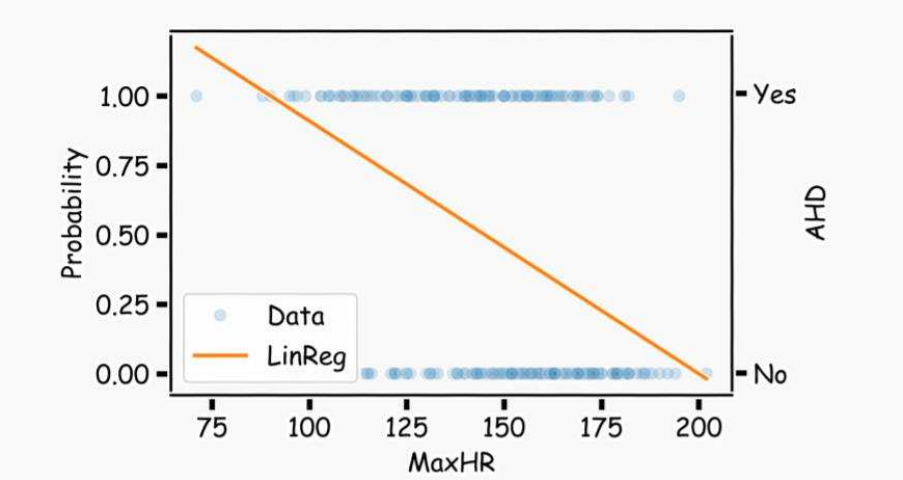
Summary of Gradient Descent

- Algorithm for optimization of first order to finding a minimum of a function.
- It is an iterative method.
- *L* is decreasing in the direction of the negative derivative.
- The learning rate is controlled by the magnitude of λ.

$$w^{(i+1)} = w^{(i)} - \lambda \frac{d\mathcal{L}}{dw}$$



Why not Linear Regression? Noncontinuous category to predict
Given a binary target, linear regression may fail horribly - let's show this in a simple graph:



Logistic Regression
Logistic Regression addresses the problem in the graph above. Think of a smoothed out version of a step function; we call it the Sigmoid or the Logistic function and is defined as

$$\sigma = F(x) = Proba\,(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{7}$$

β_0 is called the intercept and it shifts the curve left or right by

$$c = -\frac{\beta_0}{\beta_1}$$

β_1 controls how steep the S-shaped, Sigmoid curve is.
Example from Avi's lecture

X	X Coefficient(weight)
Intercept	-4.0777
Hours	1.5046

$$Proba\,(Passing\ Exam\ Blabla) = F\,(Hours) = \frac{1}{1 + e^{-(-4.0777 + 1.5046 \cdot Hours)}}$$

Let's compare the probability of passing an exam between a student who studies 2 hours to another who studies 4 hours.

$$F(2) = \frac{1}{1 + e^{-(-4.0777 + 1.5046 \cdot 2)}} = 0.255$$
$$F(4) = \frac{1}{1 + e^{-(-4.0777 + 1.5046 \cdot 4)}} = 0.874$$

4) KNN and Feature Analysis

K-Nearest Neighbor (K-NN), the "Lazy" Learner
The "big" idea: similar things are similar. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

- Regression (value before neighbors)
- Classification
  - both of the above are done in real time and without a model (hence the name Lazy)
- K, i.e. the number of neighbors is needed to be defined
- Classifying uses a majority voting mechanism

Bag of Words example: For the categories:

$$['John', 'likes', 'to', 'watch', 'movies', 'also', 'football', 'games', 'Mary', 'too']$$

and for the text:

"*John likes to watch movies. Mary likes movies too.*"

the following vector will be saved:

$$[1, 2, 1, 1, 2, 0, 0, 0, 1, 1]$$

Benefits of Lazy Learning:

- It works (sometimes)
- Decent when we don't know the number of categories in advance
- Very simple learning (some may even consider it as the simplest method)

Drawbacks of Lazy Learning:

- Requires a lot of memory for storage each instance
- Long running time
- We have to optimize K

What is considered "close" or "near"?
In Avi' lectures we studied three different methods to evaluate distances:

$$Euclidean\ Distance = d(i,j)_{Euc} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{ip} - x_{jp})^2} \quad (8)$$

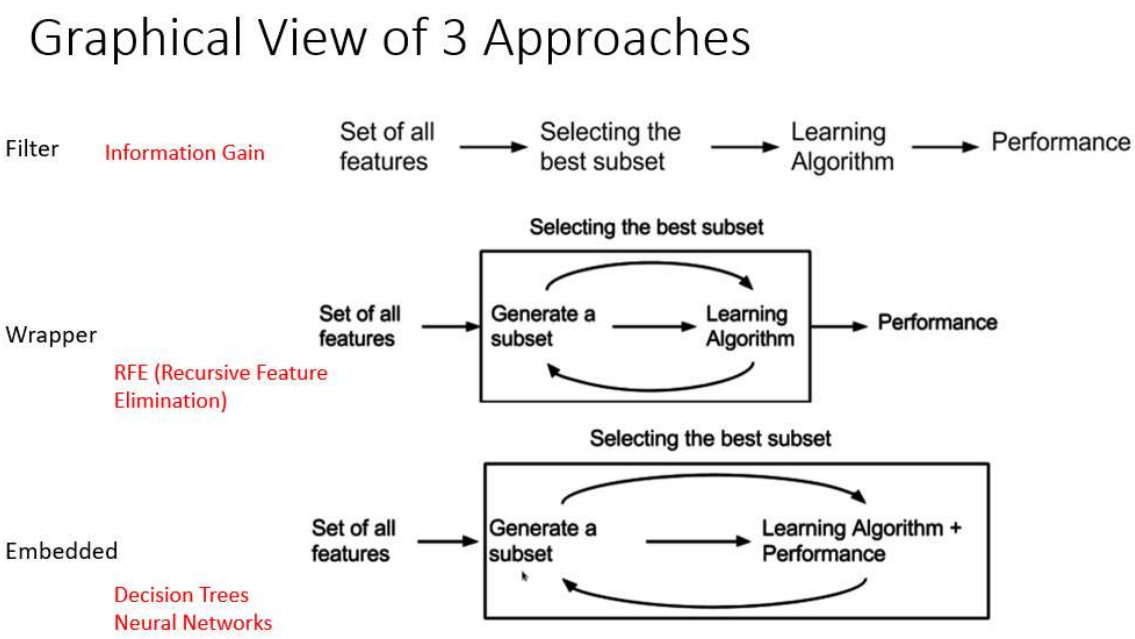
$$Manhattan\ Distance = d\left(\vec{X}, \vec{Y}\right)_{Man} = |x_1 - y_1| + |x_2 - y_2| + ... \quad (9)$$

$$Hamming\ Distance = the\ number\ of\ positions\ at\ which\ the\ corresponding\ symbols\ are\ different\ between\ two\ equal\ length\ strings\ of\ symbols \quad (10)$$

Example of Hamming distance: sinsin $\rightarrow sinsig = 1$
Implementation : $count\ +=\ int\ (s1[i]\ !=\ s2[1])$

Feature Selection :

- Can help with accuracy, less overfitting to avoid a "dead fish"
- Takes less time and money to model
- Makes model more understandable



Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.

Filter Univariate methods:
Consider one feature's contribution to the class at a time, e.g. Information Gain, Chi-square, Correlation. Advantage - It is computationally efficient and works in many fields such as Image Mining, Bioinformatics, Marketing Analysis, et cetera. Disadvantage - may select low quality feature subsets in other fields.

Correlatiion Analysis (Nominal Data)

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}} \tag{11}$$

Where **o** is the observed frequency of the events and **e** is the expected frequency of the events, that is

$$e_{ij} = \frac{count\,(A = a_i) \cdot count\,(B = b_j)}{n}$$

- The larger χ^2 is the more likely the variables are related
- The cells that contribute the most to the χ^2 value are those whose actual count is very different from the expected count
- Correlation does not imply causality e.g., # of hospitals and # of car-theft in a city are correlated. Both are causally linked to the third variable: population

Chi-Square calculation, an example from lecture:

	Play chess	Not play chess	Sum (row)
Like science fiction	250(90)	200(360)	450
Not like science fiction	50(210)	1000(840)	1050
Sum(col.)	300	1200	1500

Numbers in parenthesis are expected counts calculated based on the data distribution in the two categories

$$e\left(plays\ chess,\ likes\ science\ fiction\right) = \frac{300 \cdot 450}{1500} = 90$$

$$e\left(plays\ chess,\ doesn't\ like\ science\ fiction\right) = \frac{300 \cdot 1050}{1500} = 210$$

$$e\left(doesn't\ play\ chess,\ \ likes\ science\ fiction\right) = \frac{1200 \cdot 450}{1500} = 360$$

$$e\left(doesn't\ play\ chess,\ doesn't\ like\ science\ fiction\right) = \frac{1200 \cdot 1050}{1500} = 840$$

$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840} = 507.94$$

For $\chi^2 > 6.63$, the assumption of independence can be rejected with 99% confidence (and for $\chi^2 > 10.828$ with confidence of 99.9%). Hence, the hypothesis is rejected, i.e. likes science fiction and play chess are strongly correlated in the group.

5) Unsupervised Learning and intro to Neural Networks

K-means – the unsupervised cousin of Knn

Algorithm:

1. Define k as the number of Clusters to find.
2. Randomly select k values within the records as seeds (centroids).
3. Find the distance of every point from every seed (centroid).
4. Each point with the smallest distance will be joined to the cluster.
5. After all points have been assigned recalculate the center of each cluster (mean).
6. Repeat steps 3-5 until the distances don't decrease (i.e. when the center stops moving).

K-Means example from lecture:

Instance	X	Y
1	1.0	1.5
2	1.0	4.5
3	2.0	1.5
4	2.0	3.5
5	3.0	2.5
6	5.0	6.0

Let's assume that Instances 1 & 3 were randomly chosen as centroids.

$$C_1 = (1.0,\ 1.5)$$
$$C_2 = (2.0,\ 1.5)$$

Hence k=2 and we are in step 3 in the algorithm that's in the previous page. Let's calculate the distances using Euclidean Distances and apply the algorithm.

$$\begin{aligned} d\left(C_1, 1\right)) &\triangleq 0 \\ d\left(C_2, 3\right)) &\triangleq 0 \\ d\left(C_1, 2\right)) &\overset{(8)}{=} \sqrt{(1-1)^2 + (1.5-4.5)^2} \\ &= 3 \end{aligned}$$

and so on...

	1(1, 1.5)	2(1, 4.5)	3(2, 1.5)	4(2, 3.5)	5(3, 2.5)	6(5, 6)
C_1	**0.0000**	**3.0000**	1.0000	2.2360	2.2360	6.0207
C_2	1.0000	3.1622	**0.0000**	**2.0000**	**1.4142**	**5.4083**

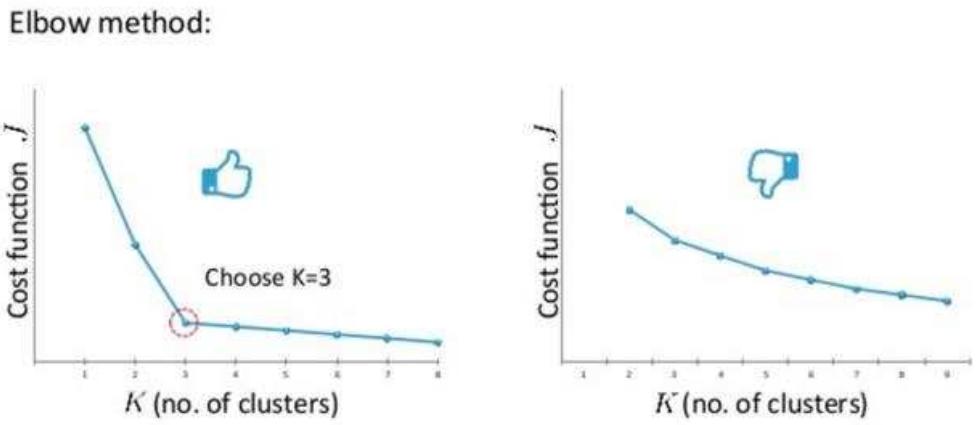
In accord to step 4 in the algorithm, the points with the smaller distances are joined the cluster and are bolded in the table above. Calculating the new centroids as the mean of the points in each group (step 5):

$$C_{1new} = \left(\frac{1+1}{2},\ \frac{1.5+4.5}{2}\right) = (1,\ 3)$$
$$C_{2new} = \left(\frac{2+2+3+5}{4},\ \frac{1.5+3.5+2.5+6}{4}\right) = (3,\ 3.375)$$

Sum of Squared Error, Evaluating K-Mean Clusters:
SSE is the sum of the squared differences between each observation and its group's mean. It can be used as a measure of variation within a cluster. If all cases within a cluster are identical the SSE would then be equal to 0. Given two clusters, we can choose the one with the smallest error.

One easy way to reduce SSE is to increase K, the number of clusters. A good clustering with smaller K can have a lower SSE than a poor clustering with higher K.

Elbow method for clustering The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. The intuition is that increasing the number of clusters will naturally improve the fit (explain more of the variation), since there are more parameters (more clusters) to use, but that at some point this is over-fitting, and the elbow reflects this. There will be a sharp elbow in the graph of explained variation versus clusters: increasing rapidly up to k (under-fitting region), and then increasing slowly after k (over-fitting region). In practice there may not be a sharp elbow, and as a heuristic method, such an "elbow" cannot always be unambiguously identified.

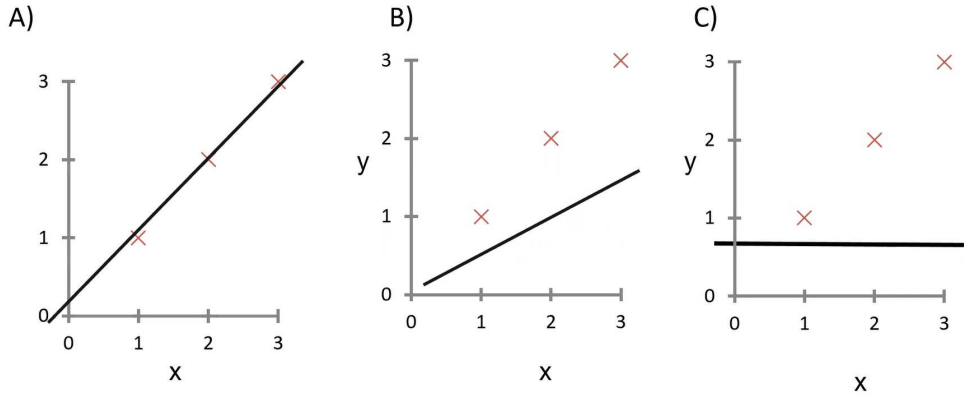


Evaluating numerical prediction using gradient descent

Linear regression - highest bias

Neural network - highest variance

Q. which function has the lowest cost? Ans. A (C has highest).



We want to minimize the cost. We'll start from a simple Linear regression model where y = mx+b. We want to find the best 'm' and 'b' given our set of data (points) that will decrease our errors (MSE,SSE,RMSE).

Example from lecture, "filename: in class targil" (we'll show one iteration):

Housing Size (X)	Price (Y)	min/max	min/max
1100	1999000	0	0
1400	2450000	0.22222	0.21989
1425	3190000	0.24074	0.58069
1550	2400000	0.33333	0.19551
1600	3120000	0.37037	0.54656
1700	2790000	0.44444	0.38567
1700	3100000	0.44444	0.53681
1875	3080000	0.57407	0.52706
2350	4050000	0.92593	1
2450	3240000	1	0.60507

$$\frac{value - min}{max - min}$$

$$\frac{\partial SSE}{\partial b} = -(Y - YP) \tag{12}$$

$$\frac{\partial SSE}{\partial m} = -(Y - YP)\,X \tag{13}$$

$$Where \quad YP = mX + b$$

iteration 1						∂SSE/∂b	∂SSE/∂m
m	b	X	Y	YP=mX+b	SSE	"-(Y-YP)"	"-(Y-YP)X"
0.75	0.45	0	0	0.45	0.101	0.45	0
		0.22	0.22	0.62	0.08	0.4	0.09
		0.24	0.58	0.63	0.001	0.05	0.01
		0.33	0.2	0.7	0.125	0.5	0.17
		0.37	0.55	0.73	0.016	0.18	0.07
		0.44	0.39	0.78	0.076	0.39	0.17
		0.44	0.54	0.78	0.029	0.24	0.11
		0.57	0.53	0.88	0.061	0.35	0.2
		0.93	1	1.15	0.011	0.15	0.14
α	α	1	0.61	1.2	0.174	0.59	0.59
0.01	0.01			Sums	0.674	3.3	1.55

$$b_{new} = b_{current} - \underbrace{\alpha}_{learning\ rate} \cdot \sum_{i=1}^{n} \frac{\partial SSE}{\partial b} \tag{14}$$

$$m_{new} = m_{current} - \underbrace{\alpha}_{learning\ rate} \cdot \sum_{i=1}^{n} \frac{\partial SSE}{\partial m} \tag{15}$$

$$b_{new} = 0.45 - 0.01 \cdot 3.30 \approx 0.42$$
$$m_{new} = 0.75 - 0.01 \cdot 1.55 \approx 0.73$$

Q. How many iterations will the process take to find the optimal values? Ans. If 100 iterations are pre-defined, we will need exactly 100 iterations to find the optimal value. Once we define the change in the cost function to be less than some given value ϵ, It may still take 100 iterations to converge but might be less. The bigger ϵ is the less iterations we have but we will have less accuracy because by definition ϵ is the value of error that satisfies us. Bigger learning rate doesn't mean better or less iterations you have to run the code and see for yourself!

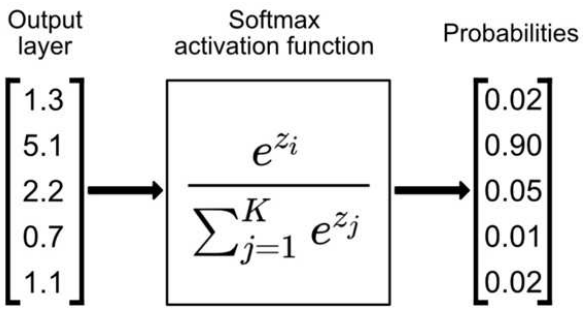
Softmax
Definition:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{16}$$

Where i = 1,...,K and z=(z_1,...,z_K) $\in$ R^K
This function's similar to Logistic Regression. Softmax with 2 options is exactly Logistic Regression.

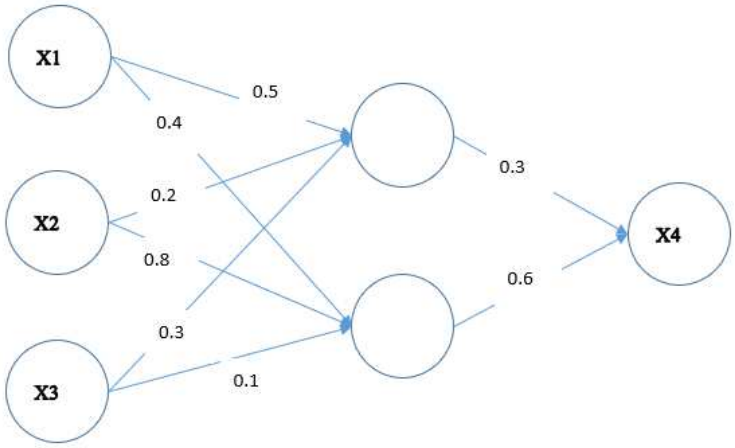
Example from lecture, "filename: softmax calc"

Output Layer	E^z	E^z/sum
1.3	3.669297	0.02019
5.1	164.0219	0.902538
2.2	9.025013	0.049661
0.7	2.013753	0.011081
1.1	3.004166	0.016531
sum	181.7341	



6) Neural Networks

Example for how to read NN Given inputs X1, X2, X3, the weights depicted in the picture below and the activation functions in the tabular, what is the output X4?



Function	X1	X2	X3	X4=?
Step	-1	1	0	**1**
ReLU	1	-3	2	0.15

$$ReLU(x) = \begin{cases} 0, & for\ x < 0 \\ x, & for\ x \geq 0 \end{cases} \tag{17}$$

$$Step(x) = \begin{cases} 0, & for\ x < 0 \\ 1, & for\ x \geq 0 \end{cases} \tag{18}$$

Let's name the two unknown neurons in the central layer, the upper neuron and lower neuron h_1 and h_2 respectively. Then, given a Step activation function:

$$\begin{aligned} h_1 &= Step\,(X1 \cdot 0.5 + X_2 \cdot 0.2 + X3 \cdot 0.3) \\ &= Step\,(-1 \cdot 0.5 + 1 \cdot 0.2 + 0) \\ &= Step\,(-0.3) = 0 \\ h_2 &= Step\,(X1 \cdot 0.4 + X_2 \cdot 0.8 + X3 \cdot 0.1) \\ &= Step\,(-1 \cdot 0.4 + 1 \cdot 0.8 + 0) \\ &= Step\,(0.4) = 1 \\ X4 &= Step\,(h_1 \cdot 0.3 + h_2 \cdot 0.6) = \mathbf{1} \end{aligned}$$

This process is called forward propagation.

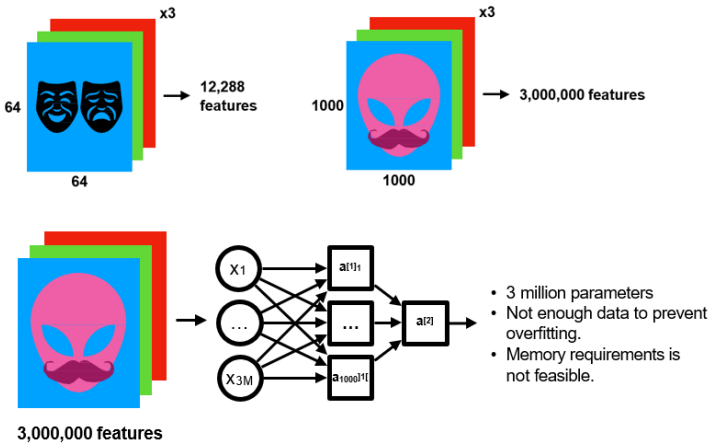
How the networks are trained? Gradient Descent!
Training process is as follows:

- Initialize the weights and biases randomly
- Iterate over the data
  - Compute the predicted output using forward propagation
  - Compute the loss using a loss function
  - Update the weights:

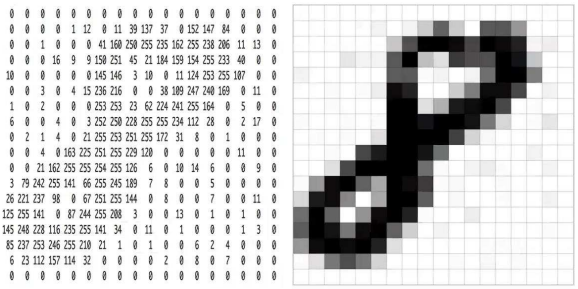
$$W(new) = W(old) - \alpha \cdot \Delta W(backward)$$

- Repeat until the error is minimal

Image Processing:
Stage one: Each pixel is a feature. Assuming black/white picture, the value is binary. But even in black/white we have gray scales. Bigger problem is, the world isn't black/white. How do we deal with it? RGB (Red-Green-Blue).



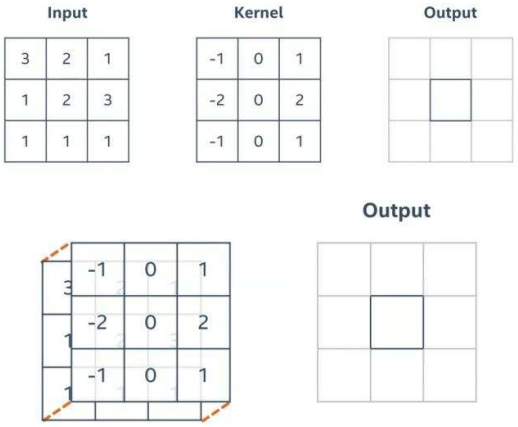
Example: MNIST Data (Modified National Institute of Standards and Technology database). It is a large database of handwritten digits that's commonly used for training various image processing systems. 18x18 256 Color grayscale of numbers: $18{\cdot}18 \cdot 256 = 82944$ *feature values* (*pixels*).



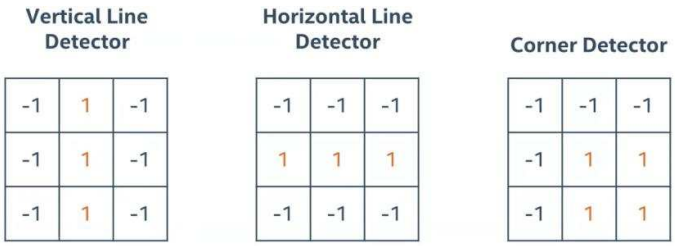
Kernels - think of them as "local feature detectors"
Used for blurring, sharpening, edge detection, embossing. Kernel is a grid of weights "overlaid" on image, centered on one pixel. Each weight is multiplied with with underneath it. Output over the centered pixel:

$$\sum_{p=1}^{P} W_p \cdot pixel_p \tag{19}$$

Example from lecture:



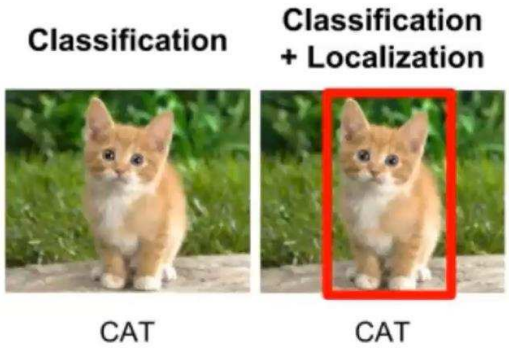
$$Output = 3 \cdot (-1) + 0 + 1 \cdot 1 + 1 \cdot (-2) + 0 + 3 \cdot 2 + (-1) \cdot 1 + 0 + 1 \cdot 1$$
$$Output = 2$$



In the real world, we don't need to beat our heads against the wall to compute each kernel, the computer does this using CNN and determines which kernels are most useful. It uses the same set of kernels across an entire image and reduces the number of parameters and "variance" (from bias-variance point of view). The network shifts over the picture (convolutes) in order to find useful parts. Finally we flatten out the data and work with a one dimensional array.

If we want to identify objects in real time then we must first recognize where the object is! (before determining if the object is indeed the object we desire to identify). This is called bounding-boxes and is another type of filter. There are several methods such as:

- Yolo - You only look once
- R-CNN (CNN+Bounding box)



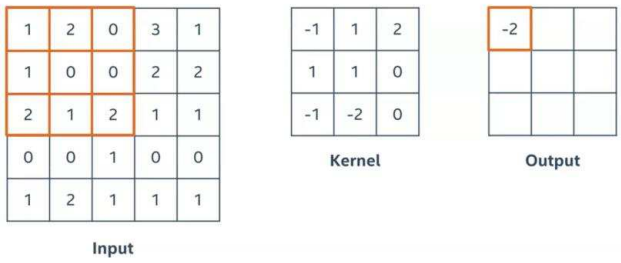
CNN's settings: Kernel size, grid size, padding (how can we find a cat in the corner if we don't scan the corner?), stride (simplest is one pixel - the problem is that stride of one means we scan too much and this is not desirable obviously), depth (how many color channels we use), pooling (mathematical manipulations on the output): max-pool, min-pool, avg-pool.

Grid Size:

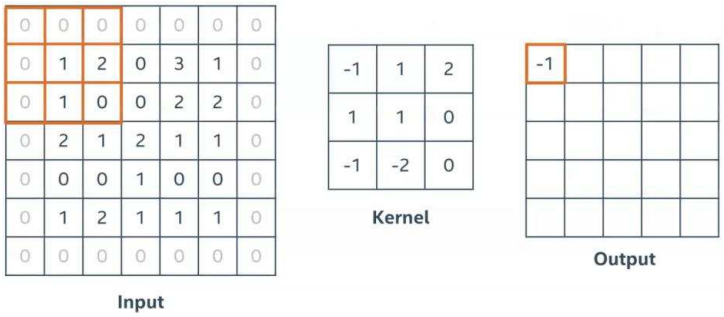
Setting the height and weight of the the grid size - it is the number of pixels the kernel "sees" at once. Its typically given an odd number so there is a clear center location. It does not have to be square but it always almost is.

Padding:

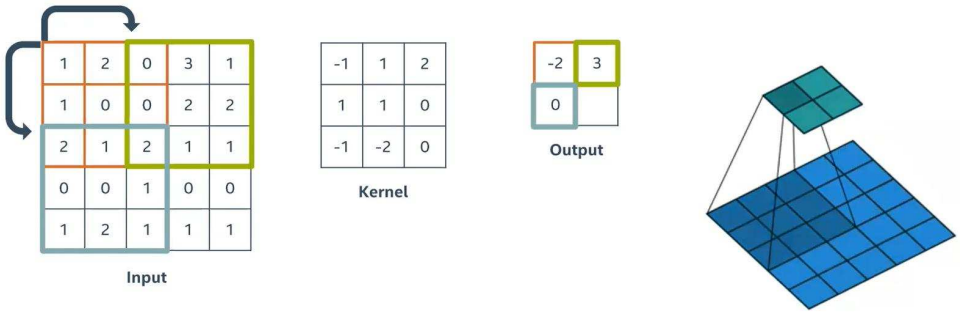
Using kernels directly, there will be an "edge effect" meaning pixels near the edge will not be used as "center pixels" since there aren't enough surrounding pixels. Padding therefore adds extra pixels around the frame. In that manner, every pixel of the original image will be a center pixel as the kernel moves across the image. Added pixels are typically of value zero (zero-padding).
Example without padding:



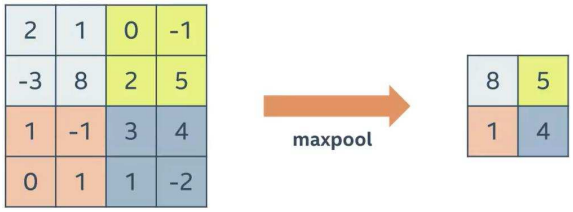
Example with padding:



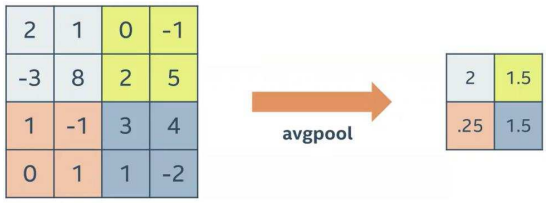
Stride:
Stride is how to scan the input to produce the output, basically how many steps to move the orange box that is depicted in the image above while scanning the input. Increasing the Stride reduces the size of the output. In the image below, setting of stride = 2 is visualized.



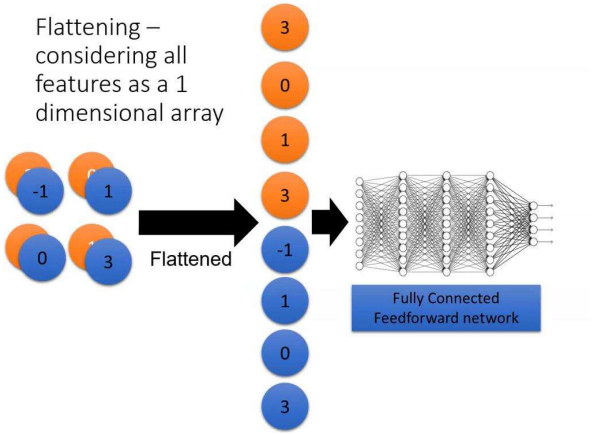
Depth:
In images, we often have multiple numbers associated with each pixel location. These numbers are refereed to as "channels". RGB image - 3 channels. CMYK image (cyan, magenta, yellow, key i.e black) - 4 channels. The number of channels is referred to as the depth. The kernel itself will have a depth the same size as the number of input channels.
Example: a 5x5 kernel on an RGB image, there will be 5x5x3 = 75 weights. A 24-bit RGB image has 2^{24} different colors.
Pooling:
The idea is to reduce the image size by mapping a patch of pixels to a single value. It shrinks the dimension of the image and is used to stress the "important" areas. Doesn't have parameters/weights. Example of 2x2 max-pool - for each distinct patch, represent it by the maximum:



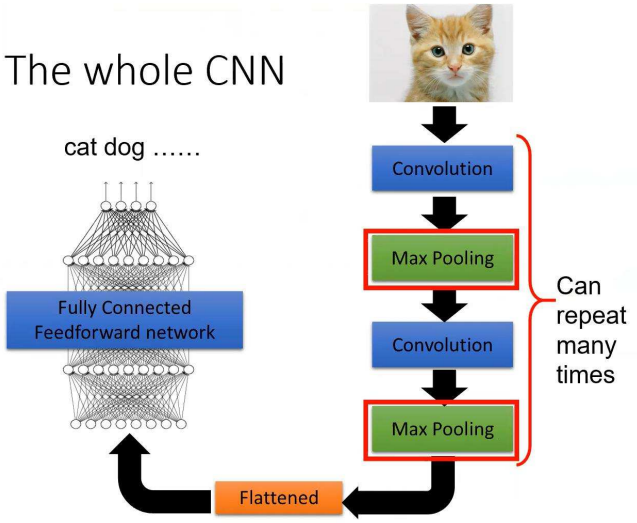
Example of 2x2 avg-pool - for each distinct patch, represent it by the average:



Flattening:

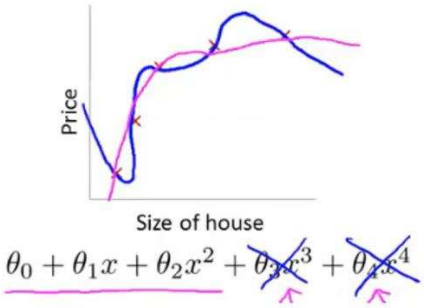


CNN process:

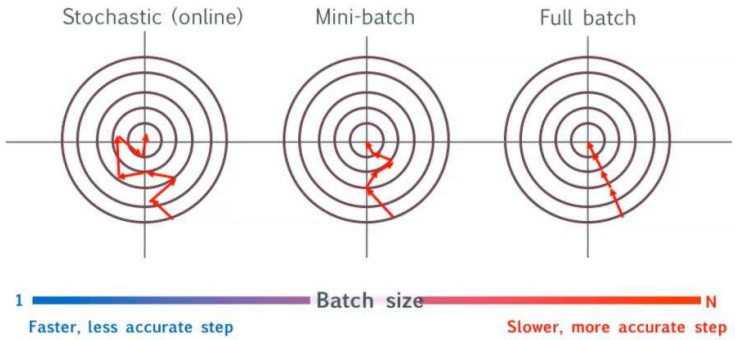


How to deal with overfitting? Augmentation, Transfer learning, Dropout, EarlyStopping. Regularization, Optimizers (e.g. momentum), SGD, RMSprop, Adam

Dropout
The term dropout refers to randomly "dropping out", or omitting, units (both hidden and visible) during the training process of a neural network. Both the thinning of weights and dropping out units trigger the same type of regularization.
Augmentation "Attacking the input" - distortion of the input in different ways. We want to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. Data augmentation is done only on training set as it helps the model become more generalize and robust.
Regularization This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. General idea: just like a spline function, we want a less accurate function that reflects reality. Regularization names - Ridge Regression, Elastic Regression, Elastic Regression. Visualization:



Stochastic Gradient Descent (SGD) Update gradiennts after each record. If you have N records you can make N updates per 1 epoch. Noisier steps but you can have more of them. In theory, these noisy steps should "balance out". Too much noise might be bad for example if the data is too noisy and has many differences. Normally what is done is what we call "Mini-batch Gradient Descent" where we take a mini-batch and find the average gradients of these records. We have $\frac{N}{batch\_size}$ number of updates per epoch. It is less noisy then SGD and allows more updates per epoch but at the expense of yet another hyperparameter that is called batch size.



Stochastic Gradient Descent - Momentum:
Momentum is method which helps accelerate gradients vectors in the right directions, thus leading to faster converging. It deals with the problem of Stochastic Gradient Descent, mostly by allowing the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space. Mathematically, any value below 1 will converge where typical values for momentum are 0.8 - 0.99.

Root Mean Square propagation - RMSprop
Uses only the sign of the gradient. It automatically adjusts the learning rate and considers previous gradients. It divides the gradient by $\sqrt{MeanSquare(w,t)}$.

Adaptive Moment Optimization - Adam
Has both elements of Momentum and RMSprop. Considered (often) as state of the art.

7) Examples with code

Let's try and explain the following code

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

$$keras.Input\,(shape = input\_shape)\,,$$

Where

$$input\_shape = (\ \underbrace{28, 28}_{28x28\ pixels}\ ,\ \overbrace{1}^{num.\ of\ color\ channels}\ )$$

$$layers.Conv2D(\ \underbrace{32}_{32\ different\ filters}\ , kernel\_size = \underbrace{(3, 3)}_{dimensions\ of\ filters}\ , activation = "relu"),$$

Q. Given an image of 28x28 pixels, applying a kernel of size (3,3) without padding and a stride of 1, what will be the size of the output? Ans. 26x26.

$$layers.MaxPooling2D(pool\_size = (2, 2)),$$

Applying max_pooling of pool_size (2,2) on a 26x26 image yields reduces the output by half that is we get 13x13. That is because each patch of 2x2 yields one value.

$$layers.ConvD(64, kernel\_size = (3, 3), activation = "relu"),$$

64 filters, "copies", or 64 distortions of the image. In this step we are scanning a 13x13 "image" without padding. Therefore we are left with a size of 11x11 (x64 copies).

$$layers.MaxPooling2D(pool\_size = (2, 2)),$$

Notice that image's dimensiom is uneven (11x11). Reducing the image size by half leaves us with two options - 5 or 6. The algorithm chooses 5, hence we are losing one pixel in this step. Q. How many weights (or parameters in keras' "language") we have to determine in the first layer? Ans.

320. If we have 32 distortions of our image, each of size 3x3 then:

$$3 \cdot 3 \cdot 32 \neq 320$$

We have to add one for our bias that is:

$$(3 \cdot 3 + 1) \cdot 32 = 320$$

Q. How many parameters or weights do we have in the third layer? Ans. 18496. We have to take the number of distortions in our previous layer into account so that:

$$(32 \cdot 3 \cdot 3 + 1) \cdot 64 = 18496$$

Note that the pooling layers (max-pooling, avg-pooling) are mere mathematical processes and do not require any weights to do such mathematical manipulations therefore each pooling layer will have 0 parameters (same for Dropout).

$$layers.Flatten(),$$

After our second MaxPooling layer, we have 64 distortions of 5x5 images. Flatten is used to create a one dimensional vector out of that. Therefore our output shape in the flatten layer is simply:

$$5 \cdot 5 \cdot 64 = 1600$$

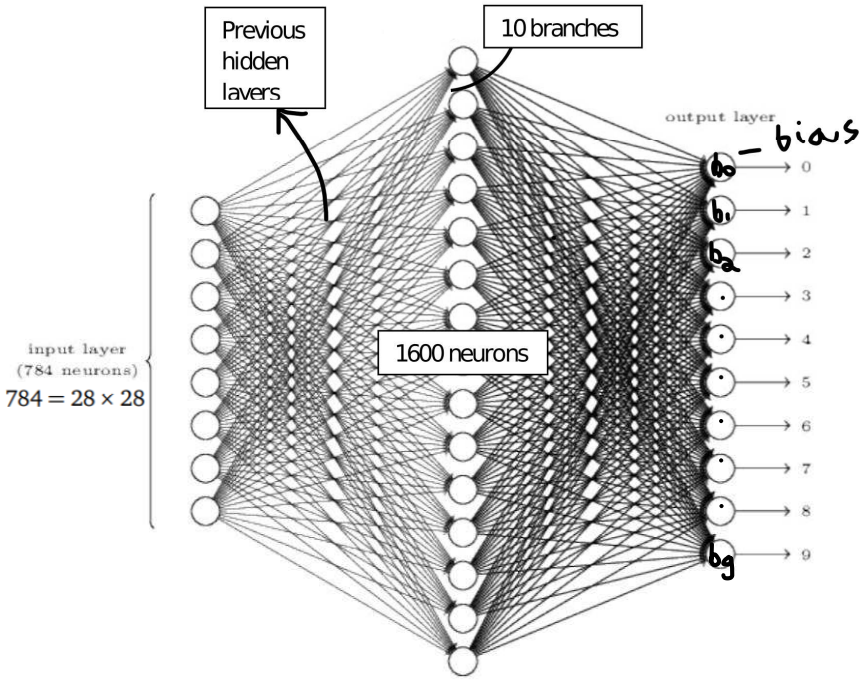
Since we want to classify numbers from 0-9 then in our final layer we'd like an output vector of size 10. For example, the number '3' classifies as [0,0,0,1,0,0,0,0,0,0]. We end up with 16010 parameters. $1600{\cdot}10\ for\ the\ branches + 10\ for\ the\ biases,\ one\ for\ each\ output.$

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010

Total params: 34,826
Trainable params: 34,826

Visual explanation for the 16010 parameters:

;



SGD with Momentum:

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

Gradient descent (with momentum) optimizer.

Update rule for parameter `w` with gradient `g` when `momentum` is 0:

```
w = w - learning_rate * g
```

Update rule when `momentum` is larger than 0:

```
velocity = momentum * velocity - learning_rate * g
w = w + velocity
```

Note what happens when momentum = 0

8) Possible questions in the upcoming exam

8.1) Theoretic questions

- Which model is the best to predict numbers?
  a. Linear Regression
  b. Random Foest
  c. Logistic Regression
  d. CNN Neural Network
  e. Blabla

8.2) Code-related questions

- How many weights are in the first MaxPooling layer, in the following code?

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

a. 0
b. $(32 \cdot 2 \cdot 2 + 1) \cdot 32$
c. $(32 \cdot 3 \cdot 3 + 1) \cdot 32$
d. 320
e. 777