ML & DM Notes

General Information

Topic	Marks	Due Date
Assignment 1 (group of 2)	15%	Week 7
Assignment 2 (group)	25%	Week 11
Exam (paper based)	60%	Exam Period

Exam Information

Exam allows us to have a sheet of paper, front and back full of notes. We can chill and copy from it. Exam will have no coding and is completely theoretical.

Total Exam Marks: 100
Marks required to pass: 40
Syllabus: Week 1 to Week 12
Types of Questions: MCQ, Short answer, Calculations.

WEEK 1

Introduction

Society produces a huge amount of data and stores all of this. The data is present everywhere and collected from almost every sector. Due to automated data collection tools and sensors, mature database technology, cheaper and more powerful computers, collecting and storing this huge amount of data is feasible. There are different kinds of data which are collected. A few examples are: purchase data from when someone purchases anything from someone, or some store, bank/credit card usage data, web data, telephone call details, scientific data which is basically data collected for scientific purposes, etc.

Data

The current trend which is being followed is to collect whatever data, whenever and wherever possible. It is expected that it will be useful either for the purpose of being collected or another purpose yet to be discovered. But the data which is captured is called raw data as it has a lot of waste data or noise. This data needs to be processed to gain useful information from it. Machine Learning and Data Mining is concerned with finding patterns in data which are meaningful, useful and actionable. This process is either automatic or semi-automatic. Following is the difference between Machine Learning and Data Mining:

Machine Learning	Data Mining
It is a core part of Artificial Intelligence	Most of the algorithms used for Data Mining is developed in Machine Learning
It does not necessarily deal with large and multidimensional data	It deals with large and multidimensional data
	It can be seen as applied Machine Learning

Data Mining

Data mining is a compilation of multiple factors such as:

- Databases
- Artificial intelligence

- Algorithms
- Machine Learning
- Information Retrieval
- Statistics

Machine Learning and Data Mining

There are two main types of tasks for Machine Learning and Data Mining. These are:

- Supervised Learning: Classification (The variable to be predicted is categorical) and Regression (The variable to be predicted is numerical)
- Unsupervised Learning: Clustering

There are other types of tasks for Machine Learning and Data Mining as well, these are:

- Association rule mining
- Reinforcement learning
- Outlier detection

Supervised Learning

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labelled datasets to train algorithms that to classify data or predict outcomes accurately. There are 2 types of supervised learning:

- Classification
  - It is used when the variable to be predicted is categorical.
  - Examples:
    - Fraud detection in credit card transactions.
    - Finding a set of customers who are likely to buy a product.
    - Predicting the class of star/galaxy from collected data.
- Regression
  - It is used when the variable to be predicted is numerical.
  - Examples:
    - Predict the electricity demand.
    - Predict the exchange rate of currencies.
    - Predict the retirement savings.
    - Predict house prices.
    - Predict stock market index.
    - Predict wind velocity based on temperature, humidity, and pressure.

The following formula is for the supervised discretization based on entropy:

$$Entropy\ (S) = -\sum_{i} P_i * \log_2 P_i$$

$$Total\ Entropy = \sum_{i}^{n} w_i\ entropy(S_i)$$

Where:

Pi → Proportion of examples from class i

Wi → Proportion of values in interval i

n → number of intervals

Unsupervised Learning

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyse and cluster unlabelled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. A few examples of clustering are:

- Segmenting customers into groups with distinct characteristics for targeted marketing.
- Analysing customer behaviour and segregating them to detect who are most likely to defect.
- Find genes with similar structure and functionality for gene clustering.
- Document clustering
- Clustering for understanding eating habits and dietary patterns of a particular cohorts.

Association Rule Mining

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is also called as Market-Basket Analysis. Another form of association rule mining is the sequential version wherein the frequent sequences in data are found. A few examples of association rule mining are:

- Rules are used for sales promotion, shelf management and inventory management (Business and Marketing)
- Find combination of alarms that occur together frequently in the same time period (Telecommunication Alarm Diagnosis)
- Unusual combinations of insurance claims can be a sign of a fraud (Insurance)
- Unusual combinations of insurance claims can be a sign of a fraud or Medical histories can give indications of complications based on combinations of treatments (Medical Informatics)

Outlier Detection

Outlier Detection means finding out the data objects whose properties and behaviour are different from the rest of the objects in the cluster or the data sets. Outlier Detection is the process of finding the outliers from the normal objects. It is essential to perform the Outlier Detection during the data pre-processing. Outliers are examples that are significantly different than the others. In statistics an outlier is typically defined as an example that differs more than 3 standard deviations from the mean of all examples. Detecting outliers is important, mainly for two reasons:

- Outliers are noise and should be removed before data analysis.
- Outliers are the goal of our DM analysis – to detect unusual behaviour.

The Data Mining Process

The data mining process is divided into multiple steps. These steps are:

- Business understanding

  In this step, the business objectives and requirements are investigated and determined if Data Mining can be applied to meet them. This leads one to determine what kind of data can be collected to build a deployable model.
- Data understanding

  An initial dataset is gathered and checked if it is suitable for further processing. If the data quality is poor, more data is collected based on a more stringent criteria and insights from the data is obtained. These insights are then reviewed along with the objectives and determined if Data Mining can be applied.
- Data preparation

In this step the data is processed so that machine learning algorithms can be applied. It involves cleaning of the data in which the incomplete (missing) values, noisy (outliers and errors) values, and inconsistent values are treated. Missing values are filled in, noisy values are smoothened, outliers are identified and removed, and inconsistencies are resolved. In transformation, the data is converted into a common format and normalized. It also undergoes dimensionality reduction and feature selection.

- Modelling

  In this step machine learning models such as prediction models are created.

- Evaluation

  A very important step wherein the model is evaluated based on its accuracy, F1 measure, etc. The patterns which are obtained are checked to be meaningful and useful or not or is it just reflecting spurious regularities. If the performance is poor, the project is redone from step one and if it is good, the model is deployed.

- Deployment

  It typically requires integration into a larger software system by software engineers. It may be necessary to reimplement the model in a different programming language so that it can be integrated well into the new software system.

Data

Data is a collection of facts also called as instances, records, observations, objects, etc. These facts are described with attributes. There are 2 types of attributes, namely:

- Nominal (categorical): Their values belong to a pre-specified finite set of possibilities.
- Numeric (continuous): Their values are numbers.

There are different types of data based on how they are collected. Some of these types are data matrix, sequential, transactional data, graphs, spatio-temporal data, etc. Data is not perfect and has a lot of noise due to various factors such as distortion of values, addition of spurious examples (facts), missing values or even due to inconsistent and duplicate data.

Noise is basically the human error when data is entered or limitations of measuring instruments or even the flaws in the data collection process. Nothing is perfect. Noise can be of multiple types, such as:

- Distortion of values
- Addition of spurious examples
- Inconsistent and duplicate data
- Missing values

There are multiple ways to deal with noise. Some of the ways are given below:

- Ignore all examples with missing values. Do only if there is a small percentage of missing values.
- Estimate the missing values by using the remaining values. For nominal values replace the missing values with the most common value among the examples of the same class, whereas for numerical values, replace with the average value of the nearest neighbours.

Data Pre-processing

Data pre-processing can refer to manipulation or dropping of data before it is used to ensure or enhance performance and is an important step in the data mining process. There are several steps to data pre-processing. These are:

- Data aggregation

  Combining 2 or more attributes into one leads to reduction in data which means less memory will be utilised and less computation time will be required. The scale is changed hence a high-level view is provided. The data becomes more stable as the aggregated data is less variable than non-aggregated data. The main disadvantage of this step is the potential loss of interesting details.
- Dimensionality reduction
- Feature extraction

  Feature extraction is the process of creation of features from raw data. This is a very important task as it can give meaning to the raw data. It requires domain expertise and may require mapping data to a new space to extract features.
- Feature subset selection

  The process of removing irrelevant and redundant features and selecting a small set of features that are necessary and sufficient for good classification is called as feature subset selection. It is very important for successful classification, and it typically improves accuracy. Using less features also means faster building of classifiers and hence these algorithms will be more often compact and easier to interpret classification rules. There are multiple methods for feature subset selection, a few are:
  - Brute force

    The process of trying all possible combinations of features as input to a machine learning algorithm and then selecting the best one.
  - Embedded

    The process of automatically selecting features. This can be done by only a few machine learning algorithms such as decision trees.
  - Filter

    Select features before the machine learning algorithm is run as the feature selection is independent of the machine learning algorithm. It is based on statistical measures and correlation.
  - Wrapper

    Select the best subset for a given machine learning algorithm as it uses the machine learning algorithm as a black box to evaluate different subsets and select the best.
- Feature weighting

  This is a method that can be used instead of feature reduction or can be used in conjunction to it. In this method the more important features are assigned a higher weigh and the less important ones, less weights. This can be done either manually (will require domain expertise) or automatically (some classification algorithms). The main idea is that the features with higher weights play an important role in the construction of the Machine Learning model.
- Converting attributes from one type to another

In this method numeric attributes are converted into nominal (this process is called discretization) or numeric and nominal attributes are converted into binary attributes (this process is called binarization). This method is needed in some ML algorithms which work only with numeric, nominal, or binary attributes.

Binarization is the process of converting categorical and numeric attributes into binary. There is no best method for this, but the simplest technique is to convert a categorical attribute into integer and then binary and a numeric attribute to categorical then integer then binary.

Discretization is the process of converting numeric attributes into nominal. It is further divided into 2 types: Supervised where the class information is used and Unsupervised where the class information is not used. It is a method where decisions need to be taken.

- Normalization

  Data normalization is a technique used in data mining to transform the values of a dataset into a common scale. This is important because many machine learning algorithms are sensitive to the scale of the input features and can produce better results when the data is normalized. This technique is used to avoid dominance of attributes with large values over attributes with small values. These are required for distance-based Machine Learning Algorithms but some other algorithms also work better with normalized data. Normalization is also called as the min-max scaling method.

Similarity Measures

Many Machine Learning algorithms require to measure the similarity between 2 examples. There are mainly 2 types of measures:

- Distance

  It can be simply explained as the ordinary distance between two points. Distance measures for numeric attributes. There are multiple types of distances:

  - Euclidean (L2 Norm)

    It can be simply explained as the ordinary distance between two points. This is the most frequently used measure. The formula is given as:

$$Euclidean\ D(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

  - Manhattan (L1 Norm)

    It is calculated as the sum of the absolute differences between the two vectors. The formula is given as:

$$Manhattan\ D(A,B) = |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_n - b_n|$$

  - Minkowski

    It is a distance measured between two points in N-dimensional space. It is basically a generalization of the Euclidean distance and the Manhattan distance. The formula is given as:

$$Minkowski\ D(A,B) = (\ |a_1 - b_1|^q + |a_2 - b_2|^q + \cdots + |a_n - b_n|^q)^{1/q}$$

- Weighted

  The distance-weighted mean is a measure of central tendency, a special case of weighted mean, where weighting coefficient for each data point is computed as the inverse sum of distances between this data point and the other data points. The formula is given as:

$$Weighted\ D(A,B) = \sqrt{w_1|a_1 - b_1|^2 + w_2|a_2 - b_2|^2 + \cdots + w_n|a_n - b_n|^2}$$

- Hamming

  The Manhattan distance for binary vectors

- Correlation

  Correlation is a bivariate analysis that measures the strength of association between two variables and the direction of the relationship. In terms of the strength of the relationship, the correlation coefficient's value varies between +1 and -1. It measures the linear relationship between numeric attributes. The formula for correlation is as follows:

$$Correlation\ r = \frac{n\sum xy - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

$$corr(x,y) = \frac{covar(x,y)}{std(x) * std(y)}$$

$$mean(x) = \frac{\sum_{k=1}^{n} x_k}{n} \qquad std(x) = \sqrt{\frac{\sum_{k=1}^{n} (x_k - mean(x))^2}{n-1}}$$

$$covar(x,y) = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - mean(x))(y_k - mean(y))$$

Where:

+1 = perfect positive correlation
-1 = perfect negative correlation
0 = no correlation

The formulae for different types of coefficients are as follows:

$$Simple\ Matching\ Coefficient = \frac{f_{11} + f_{00}}{f_{00} + f_{01} + f_{11} + f_{10}}$$

$$Jaccard\ Coefficient = \frac{f_{11}}{f_{01} + f_{11} + f_{10}}$$

$$Cosine\ similarity\ (d_1, d_2) = \frac{d_1 * d_2}{||d_1|| * ||d_2||}$$

Calculating Similarity Coefficients

A = [1 0 0 0 0 0 0 0 0 0]
B = [0 0 0 0 0 0 1 0 0 1]

D (A, B) = 3

F00: 7, F01: 2, F10: 1, F11: 0

$$Simple\ Matching\ Coefficient\ = \frac{0+7}{2+1+0+7} = 0.7$$

$$Jaccard\ Coefficient = \frac{0}{2+1+0} = 0$$

D1 = 3 2 0 5 0 0 0 2 0 0
D2 = 1 0 0 0 0 0 0 1 0 2

$$Cosines\ similarity\ (d_1, d_2) = \frac{d_1 * d_2}{||d_1|| * ||d_2||}$$

$$d_1 * d_2 = (3*1) + (2*0) + \cdots + (0*2) = 5$$

$$||d_1|| = \sqrt{(3^2) + (2^2) + \cdots + (0^2)} = \sqrt{42} = 6.481$$

$$||d_2|| = \sqrt{(1^2) + (0^2) + \cdots + (2^2)} = \sqrt{6} = 2.245$$

$$Cos\ (d_1, d_2) = 0.3150$$

An example of correlation is as follows:

X = (-3, 6, 0, 3, -6)
Y = (1, -2, 0, -1, 2)
N = 5

X	Y	XY	X^2	Y^2
-3	1	-3	9	1
6	-2	-12	36	4
0	0	0	0	0
3	-1	-3	9	1
-6	2	-12	36	4
0	**0**	**-30**	**90**	**10**

$$Correlation\ r = \frac{n\sum xy - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

$$= \frac{5(-30) - (0)(0)}{\sqrt{[(5*90) - (0)^2][(5*10) - (0)^2]}} = -1$$

WEEK 2

Nearest Neighbour Algorithm

The k-nearest neighbours' algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. To make a prediction for a new unlabelled example, the classifier will:

- Find the nearest training example to it using a distance measure.
- The class label of the nearest neighbour will be the predicted label for the new example.

Normalisation

Normalisation is the process of attribute transformation of the existing attributes to a new range, for example: 0, 1. This technique is used to avoid dominance of attributes with large values over attributes with small values when calculating distance. This is a must require step for calculating ML algorithms such as Nearest neighbours. Normalisation is performed for each of the attributes present in the equation. It is also called as the min-max scaling method. The formula for normalisation is given as:

$$Normalization: x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Where:

x → original value
x' → new value
min(x) & max(x) → minimum and maximum values of x

Following is the formula for Standardization:

$$Standardization: x' = \frac{x - \mu(x)}{\sigma(x)}$$

Where:

x → original value
x' → new value
μ(x) → mean value of the attribute
σ(x) → standard deviation of the attribute

Normalization using manual calculations.

Example: Predict if a customer is likely to buy a product based on 2 attributes: age and income

- *age* (in years) and annual *income* (in dollars) have different scales

A=[20, 40 000]
B=[40, 60 000]

$$Normalization: x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Using the above formula, we get the following:

Before normalization:	After normalization:
A=[20, 40 000]	A=[0.20, 0.4]
B=[40, 60 000]	B=[0.40, 0.6]
C=[25, 30 000]	C=[0.25, 0.3]

Now we calculate the distance using normalized values and can get good results.

Computational Complexity

It is the computational complexity, a measure of the amount of computing resources (time and space) that a particular algorithm consumes when it runs. Training an ML algorithm is fast as no model is built and it's just storing the training examples. Classification algorithms usually predict for a new example. This is done by comparing each unseen example with each training example. The memory requirements for each training example would be O(mn). This

is highly impractical due to the time and memory constraints for large datasets. However, there are more algorithms which can find the nearest neighbours quite efficiently.

K Nearest Neighbour

It is like nearest neighbour but is very sensitive to the value of k. Commercial packages usually use k=10. A general rule of thumb to follow is that the value of k should always be less than the square root of the number of training examples. Using more nearest neighbours increases the robustness to noisy examples. K-Nearest neighbours can not only be used for Classification but also for regression algorithms as the prediction will be the average value of the class values of the K nearest neighbours.

K-Nearest neighbour algorithm is usually very accurate and has been used in statistics since 1950s. It is very slow for big datasets as it requires efficient data structures such as the KD-trees for huge datasets. It is a distance based technique which requires normalization and produces arbitrarily (randomly) shaped decision boundary defined by a subset of the Voronoi edges. It is not effective for high dimensional (multivariate) data and to use the technique one might need to use dimensionality reduction or feature selection to proceed further with the technique. Lastly, the K nearest neighbour algorithm is sensitive to the value of K.

Predicting Nearest Neighbour Algorithm using Euclidean Distance

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

For a1 = 2, a2 = 4, a3 = 2

$$[yes]D(new, e1) = \sqrt{(1-2)^2 + (3-4)^2 + (1-2)^2} = \sqrt{3}$$
$$[yes]D(new, e2) = \sqrt{(3-2)^2 + (5-4)^2 + (2-2)^2} = \sqrt{2}$$
$$[no]\ D(new, e3) = \sqrt{(3-2)^2 + (2-4)^2 + (2-2)^2} = \sqrt{5}$$
$$[no]\ D(new, e4) = \sqrt{(5-2)^2 + (2-4)^2 + (3-2)^2} = \sqrt{14}$$

As the closest (smallest) answer is D (new, e2) with the class as “yes”, the prediction is that the class of a1=2, a2=4, a3=2 is also yes.

K-Nearest Neighbour example with k as 3

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

$$[yes]D(new, e1) = \sqrt{(1-2)^2 + (3-4)^2 + (1-2)^2} = \sqrt{3}$$
$$[yes]D(new, e2) = \sqrt{(3-2)^2 + (5-4)^2 + (2-2)^2} = \sqrt{2}$$
$$[no]\ D(new, e3) = \sqrt{(3-2)^2 + (2-4)^2 + (2-2)^2} = \sqrt{5}$$
$$[no]\ D(new, e4) = \sqrt{(5-2)^2 + (2-4)^2 + (3-2)^2} = \sqrt{14}$$

As the closest (smallest) 3 nearest neighbours are e1 (yes), e2 (yes) and e3 (no), and considering that (yes) is in the majority, the K-Nearest neighbour predicts class as (yes).

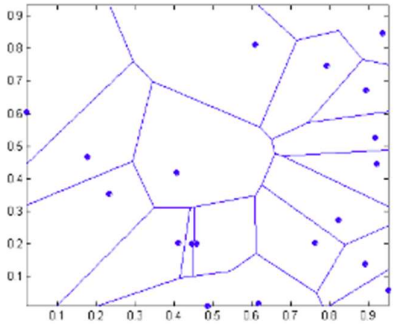
Weighted Nearest Neighbour

The general idea for this algorithm is that the closer neighbours should count more than distant neighbours. This algorithm is distance weighted nearest neighbour algorithm. The general flow of the algorithm is to first find the k-nearest neighbours and then weigh their contribution based on their distance to the new example. The formula for the relation between weight and distance is given by:

$$Weight\ (w) = \frac{1}{d^2}$$

Decision boundary of 1 nearest neighbour

Nearest neighbour classifiers produce decision boundaries with an arbitrary shape, meaning that the boundaries are set completely randomly. The 1 nearest neighbour boundary is formed by the edges of the Voronoi diagram that separates the points of 2 classes. The Voronoi region is the region associated with each training example; meaning that the region consists of all data points for which the class point is the closest example. An example of the Voronoi diagram is given below:



Rule Based Algorithm [1R]

1R represents 1 rule algorithm which means that the algorithm generates 1 rule to test the value of a single attribute. The rule can be as simple as an if/else statement. A rule is generated for each attribute and the one with the smallest error rate (or with the highest accuracy) on training data is the best rule to be used. The basic algorithm for a 1R is:

- Generate a rule for each attribute.
- Evaluate each rule on the training data and calculate the number of errors.
- Choose the one with the smallest number of errors.

1R is a simple and computationally efficient algorithm. It produces rules that are easy to understand in contrast to the black box models such as the neural networks. This algorithm sometimes produces surprisingly good results. It will be useful to try simple algorithms first and compare their performance with more sophisticated algorithms. Numerical datasets require discretization and 1R has an inbuilt procedure to do this.

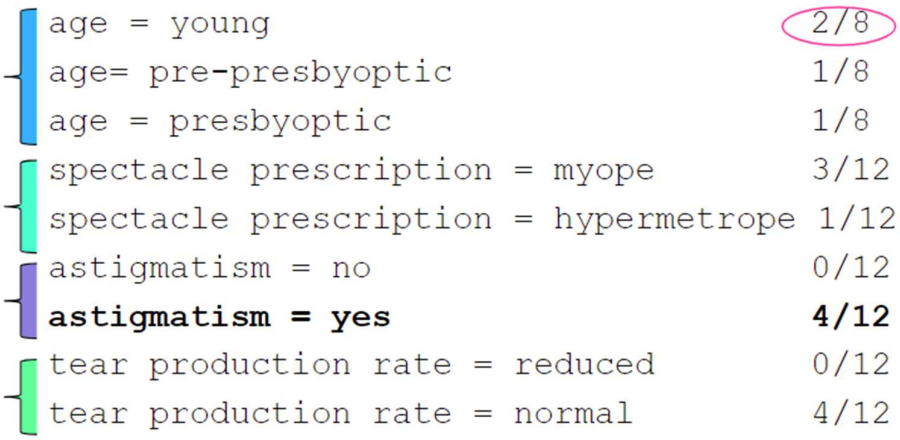
Rule Based Algorithm [PRISM]

PRISM is an example of rule-based covering algorithm. This type of algorithm considers each class in turn and constructs a set of "if-then" rules that cover all examples from this class and do not cover any examples from another class. This is called a covering algorithm because at each stage, a rule is identified that covers some examples. The "if" part of the rules contains conditions that test the value of an attribute which is initially empty but to which tests are added incrementally in order to create a rule with maximum accuracy. The accuracy on training data for such algorithms is always 100%. A perfect rule covers only examples from the same class.

The main idea of the PRISM algorithm is to generate a rule by adding tests that maximise the accuracy of the rule. For each class, the rule is started off by an empty rule and gradually rules are added incrementally wherein each condition tests the value of 1 attribute. By adding a new test, the rule coverage is reduced, and the rule becomes more specific. The rules stop being generated if the p/t=1 (p: positive examples from the class under consideration, t: total number of examples, t-p: number of errors made by the rule) meaning that the rule covers only examples from the same class, making it a perfect rule with an accuracy of 100%.

There is one problem with rule-based algorithm PRISM, and that is that not all test examples might be covered and hence will not receive classification. Hence a default rule is needed to assign them to the class with the most training examples. As this algorithm is dealing with numeric attributes, discretization is needed.

Rule based Algorithm [PRISM] example.



Step i. Calculate the p/t accuracy and select the highest one.
Step ii. In case there is a tie, choose one between them randomly.
Step iii. Refine and recalculate the p/t scores, select the highest accuracy.

- Possible tests:

age = young	2/4
age= pre-presbyoptic	1/4
age = presbyoptic	1/4
spectacle prescription = myope	3/6
spectacle prescription = hypermetrope	1/6
tear production rate = reduced	0/6
tear production rate = normal	**4/6**

Step iv. Continue refinement until p/t = 1 is obtained.

- Possible tests

age = young	2/2
age= pre-presbyoptic	1/2
age = presbyoptic	1/2
spectacle prescription = myope	**3/3**
spectacle prescription = hypermetrope	1/3

Example:

i. Astigmatism and tear production rate (normal) = 1/3
   a. Rule: if astigmatism = yes then recommendation = hard.

ii. Tear production rate (normal) = 2/3
   a. Rule: If astigmatism = yes & tear production rate = normal then recommendation = hard.

iii. Spectacle prescription (myope) = 1
   a. Rule: if astigmatism = yes & tear production rate = normal & spectacle prescription = myope then recommendation = hard.

Considering that the rule is perfect, no more rules are added as all examples covered are from class hard. Considering there are other classes excluding hard, new rules are required. We start the process all over again to get the rule for the remnant class. Thus the rules which cover all classes are:

- if astigmatism = yes & tear production rate = normal & spectacle prescription = myope then recommendation = hard.
- if age = young and astigmatism = yes and tear production = normal then recommendation = hard.

Same process for “normal” and “soft”

Covering vs Divide and Conquer algorithms.

- PRISM is an example of the covering approach for constructing classification rules.
- Decision trees is an example of the divide and conquer approach.
- Both the methods select the best test to add to the current rule but the criteria is different.
  - PRISM maximises the accuracy p/t.
  - DT maximises the separation between classes.

Classification Task #1

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Task: Build a model (classifier) that can be used to predict the class of new (unseen) examples.

WEEK 3

Introduction

Linear regression is a prediction method used for regression tasks. Regressions tasks are those for which predicted variables which are numeric data types. Logistic regression on the other hand is an extension of linear regression for classification tasks. Classification tasks are those wherein the predicted variable is nominal in nature. Both regressions are very popular in the field of statistics.

Linear Regression

Given a dataset of 2 continuous variables, one would be the feature X (independent variable) and the other would be the predicted variable Y (Target variable or dependent variable). The main goal of linear regression is to approximate the relationship between these 2 variables with a straight line for the given dataset. There are 2 different types of applications of linear regression, these are:

- Prediction: Given a new value of independent variable, we use the regression line to predict the value of the dependent variable.
- Descriptive analytics: We assess the strength of the relationship between the two variables.

The equation of the Linear regression Line is given as:

$$y = b_0 + b_1 x$$

Where:

Y = dependent variable
X = independent variable
b0 and b1 = regression coefficients

The closest line to the data is the one which is usually the best fit line. Mathematically the prediction error (residual) is given by the formula:

$$\varepsilon = y_i - \hat{y}_i$$

The main goal is to select a line that minimises the SSE (Sum of Squared Errors) which is given by the formula:

$$SSE = \sum_i (y_i - \hat{y}_i)^2$$

To select the line with the minimal SSE, we can use the method of least squares which uses the following formulae:

$$b_1 = \frac{\sum x_i y_i - \frac{(\sum x_i)(\sum y_i)}{n}}{\sum x_i^2 - \frac{(\sum x_i)^2}{n}}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

Where:

$\bar{x}$ = mean value of x
$\bar{y}$ = mean value of y
n = number of training examples

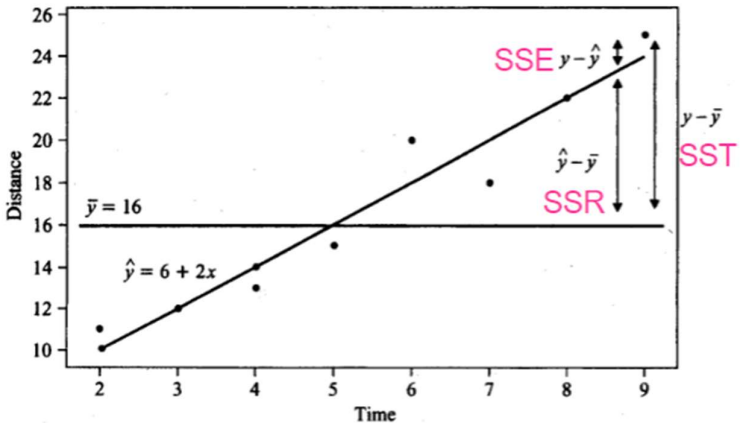
The least squares method finds the best fit of the data but doesn't tell us how good the fit is. R^2 measures the goodness of fit of the regression line found by the least squares method. The value ranges from 0 to 1 and the higher the number, the better the fit. The formula is:

$$R^2 = \frac{SSR}{SST}$$

$$Sum\ of\ Squared\ Prediction\ Errors\ (SSE) = \sum_i^n (y_i - \hat{y}_i)^2$$
$$= actual\ value - predicted\ value$$

$$Sum\ of\ Squared\ Total\ Errors\ (SST) = \sum_i^n (y_i - \bar{y}_i)^2 = actual\ value - mean\ value$$
$$SST = (n-1)\ var\ (y)$$

$$Sum\ of\ Squared\ Regression\ Errors\ (SSR) = \sum_i^n (\hat{y}_i - \bar{y}_i)^2$$



r stands for the correlation coefficient and measures the linear relationship between 2 vectors x and y. The formula for r is given as:

$$r = corr(x,y) = \frac{covar(x,y)}{std(x)std(y)} = \frac{covar(x,y)}{\sqrt{var(x)var(y)}}$$

$$R^2 = \frac{SSR}{SST}$$

$$r = \pm\sqrt{R^2}$$

Performance Measures for evaluating model.

There are 3 performance measures which can be used to evaluate how good the model is and how well it works on the new data. These are:

i. Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

ii. Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

iii. Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Note that if the LR model is fitted on one dataset but tested on another dataset, then it is possible that the R^2 value is negative. Negative value means a poor fit. If the model is trained and tested on the same dataset, R2 is always between 0 and 1.

Multiple Regression

A simple regression uses only 1 feature, multiple regression uses more than 1 feature. In a simple regression, the line becomes a plane in 2 dimensional space and a hyperplane in more than 2 dimensional plane. R^2 is similarly defined as the multiple coefficients of determination.

Logistic Regression

Logistic regression is used for classification tasks. The two classes are usually 0 and 1 (false and true) but there are more extensions if there are more than 2 classes. The regression model fits the data into a logistic (sigmoidal curve) instead of fitting it into a straight line. The logistic regression assumes that the relationship between the feature and the class variable is non-linear. A simple bivariate logistic regression curve's equation is given by:

$$p = \frac{e^{b_0+b_1x}}{1+e^{b_0+b_1x}}$$

The logistic regression model gives a value between 0 and 1 that is interpreted as a probability for the class membership. P is the probability for class 1 and (1-P) is the probability for class 0. The logistic regression uses maximum likelihood method to find parameters b_0 and b_1 which is basically the curve that best fits the data. A simplified equation of logistic regression model is given by:

$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

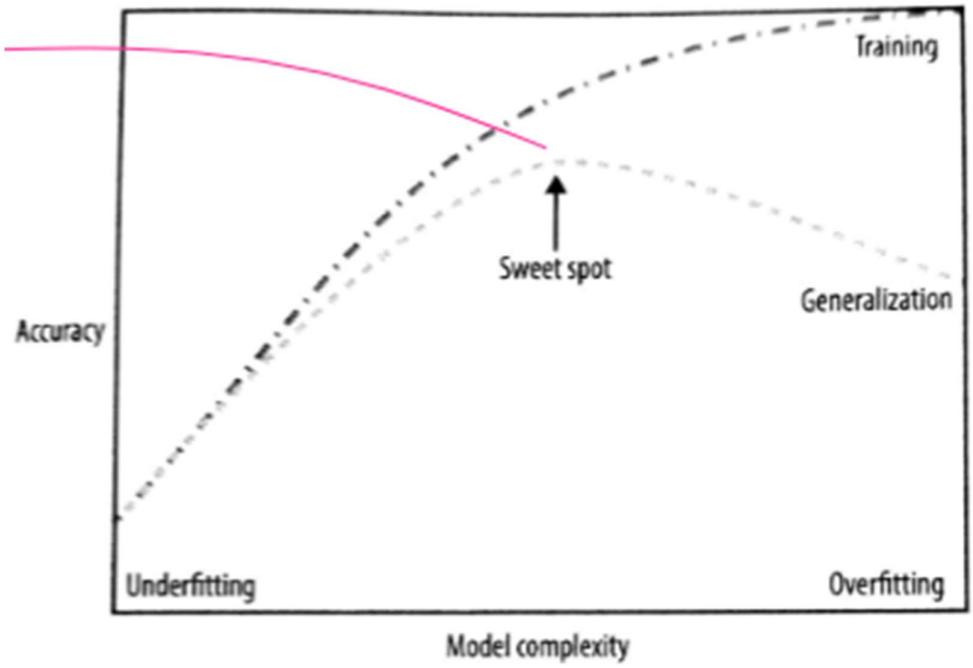
Overfitting

Overfitting is an undesirable machine learning behaviour that occurs when the machine learning model gives accurate predictions for training data but not for new data. This is usually because the classifier has memorized the training examples but has not learned to generalize new examples. It occurs when we fit a model too closely to the particularities of the training set, meaning that the resulting model is very specific and works well on the training data but not on the testing data. There are various reasons why overfitting might happen, a few reasons are:

- Issues with the data
  - Noise in the training data.
  - Training data does not contain enough representative examples.
  - Training data is very different from the testing data.
- How the algorithm operates
  - Some algorithms are more susceptible to overfitting than others.
  - Different algorithms have different strategies to deal with overfitting, for example:
    - Decision trees: prune the tree.
    - Neural networks: Early stopping in the training.

Underfitting

The model is too simple and doesn't capture all the important aspects of the data. In this problem, the model performs badly on both the training, as well as the testing data set. The reason why underfitting might occur is because the rule might be too generic.



Regularization

Regularization means explicitly restricting a model to avoid overfitting. It is used in certain regression models such as the Ridge and Lasso regression models and in some neural networks.

Ridge Regression

It is a regularized version of Linear Regression and is also known as the Tikhonov Regularization. It uses the same equation as the Linear Regression Model to make predictions however the regression coefficients are chosen in such a way that not only does it fit the training data well but it also makes sure that the magnitude of coefficients is as small as possible. Small values of coefficients means that each feature will have little effect on the outcome and the regression line will have a small slope. This makes it in such a way that the linear regression model is less complex and more restricted thus making it less likely to overfit. It uses the so called L2 regularization method. It minimizes the following cost function:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \alpha\sum_{i=1}^{m} w_i^2$$

Where:

n = number of training examples.

m = number of regression coefficients.

The main goal of this algorithm is to have high accuracy on the training data and be a low complexity model. The parameter α controls the trade off between the performance on the training set and the model complexity.

By increasing the α, the coefficients become smaller (close to 0). This typically decreases the performance of the training set but may improve the performance on the testing set. Decreasing the α means less restricted coefficients, for very low α values, the Ridge Regression will behave similarly to the Linear regression models.

Lasso Regression

It is another type of regularized version of the standard Linear Regression Model. LASSO stands for Least Absolute Shrinkage and Selection Operator Regression. As the Ridge regression, this regression model adds a regularization term to the cost function, but it uses the L1 norm of the regression coefficient vector unlike the Ridge regression which uses L2 norm. The following cost function is obtained:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \alpha\sum_{i=1}^{m} |w_i|$$

Where:

n = number of training examples.

m = number of regression coefficients.

The main goal of this regression model is to get high accuracy on training data while have a low complexity model. The consequence of using the L1 norm is that some regression coefficients can become exactly 0. That means that certain features would be completely ignored by the model, which makes it kind of like an automatic feature selection model. As the features would be less, the model itself will be less complex.

The α controls the trade off between the performance on the training set and model complexity in this case as well. By increasing the α, the coefficients become smaller some exactly 0. This typically decreases the performance of the training set but may improve the

performance on the testing set. Decreasing the α means less restricted coefficients, for very low α values, the Lasso Regression will behave similarly to the Linear regression models.

WEEK 4

Probabilistic Classifiers

Probabilistic Classification methods compute the class membership probability. Meaning that it gives the probability of a given sample belonging to a certain class. Naïve Bayes is a prominent example of this group of classifiers. This theorem is based on Bayes Theorem which describes the probability of an event based on prior knowledge of conditions that might be related to the event. The Bayes theorem is given by:

$$P(H \mid E) = \frac{P(E \mid H)\, P(H)}{P(E)}$$

Where:

H = Hypothesis
E = Evidence
P = Probability

Naïve Bayes Algorithm

This algorithm uses the Bayes theorem to solve classification tasks. It takes into consideration 2 assumptions, these are:

- The attributes are conditionally independent of each other for each class value.
- All attributes are equally important.

Considering that these assumptions are almost never correct, the algorithm is called Naïve Bayes theorem. These assumptions, however, leads to a simple and easy to implement algorithm which works well in practice. Given below are the steps to use Bayes Theorem:

i. Calculate P(H | E) for each H (class). ~ P(yes | E) and P(no | E).
ii. Compare them and assign E to the class with the highest probability.
iii. For P(H | E) we need to calculate P€, P(H) and P(E | H) from the given training data.

Calculating Probabilities from Training Data

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Consider just 1 row:

outlook	temp.	humidity	windy	play
sunny	cool	high	true	?

E1 = outlook = sunny
E2 = temp = cool
E3 = humidity = high
E4 = windy = true

Considering E1, E2, E3, and E4 are independent in the given class, their combined probabilities are:
P(E | yes) = P(E1 | yes) P(E2 | yes) P(E3 | yes) P(E4 | yes)
P(E | no) = P(E1 | no) P(E2 | no) P(E3 | no) P(E4 | no)

This gives us:

$$P(yes \mid E) = \frac{P(E_1 \mid yes)P(E_2 \mid yes)P(E_3 \mid yes)P(E_4 \mid yes)P(yes)}{P(E)}$$

&

$$P(no \mid E) = \frac{P(E_1 \mid no)P(E_2 \mid no)P(E_3 \mid no)P(E_4 \mid no)P(no)}{P(E)}$$

Based on the training data, we estimate the E1, E2, E3, and E4 probability values.

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

P(E1 | yes) = P(outlook = sunny | yes) = 2/9
P(E2 | yes) = P(temp = cool | yes) = 3/9
P(E3 | yes) = P(humidity = high | yes) = 3/9
P(E4 | yes) = P(windy = true | yes) = 3/9
P(yes) = 9/14

$$P(yes \mid E) = \frac{P(E_1 \mid yes)P(E_2 \mid yes)P(E_3 \mid yes)P(E_4 \mid yes)P(yes)}{P(E)}$$

$$P(yes \mid E) = \frac{\frac{2}{9} * \frac{3}{9} * \frac{3}{9} * \frac{3}{9} * \frac{9}{14}}{P(E)} = \frac{0.0053}{P(E)}$$

P(E1 | no) = P(outlook = sunny | no) = 3/5
P(E2 | no) = P(temp = cool | no) = 1/5
P(E3 | no) = P(humidity = high | no) = 4/5
P(E4 | no) = P(windy = true | no) = 3/5
P(no) = 5/14

$$P(no \mid E) = \frac{P(E_1 \mid no)P(E_2 \mid no)P(E_3 \mid no)P(E_4 \mid no)P(no)}{P(E)}$$

$$P(no \mid E) = \frac{\frac{3}{5} * \frac{1}{5} * \frac{4}{5} * \frac{3}{5} * \frac{5}{14}}{P(E)} = \frac{0.0206}{P(E)}$$

Since P(no | E) > P(yes | E)
Naïve Bayes predicts Play = no for the new day.

The Zero Frequency Problem
Considering the probability is given as:

$$P(yes \mid E) = \frac{P(E_1 \mid yes)P(E_2 \mid yes)P(E_3 \mid yes)P(E_4 \mid yes)P(yes)}{P(E)}$$

If P(E1 | yes) = 0 then the entire probability becomes 0. This means that the prediction will ignore the values of all the other variables. For this, we add 1 to the numerator and m to the denominator. This is called Laplace correction or smoothing and it ensures that the probability will never be 0. This is a generalization of the Laplace correction called m-estimate.

Missing Values

Naïve Bayes can easily deal with missing values during classification. When calculating the probability, the missing value attribute itself is not taken into consideration. During training, missing value in the counts are not included and the probabilities are calculated based on actual number of training examples without missing values for each attribute.

Applying Naïve Bayes theorem to numeric attributes

An assumption is taken into consideration wherein we assume that the numeric attributes follow a normal or gaussian distribution and use a probability distribution function. The probability density function for a normal distribution is given as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

Where:

σ = Standard deviation
μ = Mean

Once this has been calculated, just substitute the probability in the bayes theorem formula.

Evaluating Machine Learning Algorithms

The machine learning algorithm is evaluated by its performance and there are multiple evaluation procedures and performance measures which can let us know about the performance of the data and evaluate the ML algorithm. A few evaluation procedures are:

- **Holdout method**

  In this method we split the data into 2 sets, the training set and the testing set. This is usually done in the ratio of 2/3 and 1/3 with the larger part being training dataset. We build the model on the training data and then evaluate the model on the testing data. By evaluation, we mean to calculate accuracy and other such performance measures.

Sometimes a third dataset is needed which is called the validation dataset. In this case the dataset is split into three datasets, the training, validation and testing datasets. This is used in cases where some classification models such as decision trees, random forests, etc. operate in 2 stages. Stage 1 where they build the classifier, and Stage 2 where they tune its hyperparameters. Thus, to tune the hyperparameters, we need the validation dataset.

- **Holdout method with parameter tuning**

  Hyperparameters are parameters that can be tuned to optimize the performance of a ML algorithm. A few examples of hyperparameters are the k from k nearest neighbour algorithm, or the number of hidden layers or the number of hidden epochs in neural networks. However, in ML, Hyperparameter tuning is known as Parameter tuning.

- **Holdout method with stratification**

  Since the split of data is random in nature when creating the training and testing data sets, stratification is required to make sure all classes are present in both, the training as well as the testing dataset. Stratification can be used together with the holdout method to improve it and in the process ensures that each class is represented with approximately equal proportions in both the datasets.

- **Repeated Holdout method**

  The holdout method is made more reliable by repeating the random split into training and testing datasets several times and calculating the average accuracy. Further optimization can be done by making sure there are no overlaps in the test sets.

- **Cross validation**

  This method avoids the overlaps in the test sets. The 10-fold cross validation technique is typically used. The steps for this method are as follows:

  - Split the data into 10 sets named set 1, set 2, set 3… set 10 approximately of the same size.
  - A classifier is built 10 times, each time testing is on 1 set and the training is on the remaining sets. Meaning:

    Run 1: Test = Set 1, Train = Set 2-9
    Run 2: Test = Set 2, Train = Set 1 & Set 3-9
    …
    Run 10: Test = Set 10, Train = Set 1-9
  - Cross validation accuracy is calculated which is basically the average of the accuracies generated from each run.

- **Cross validation with stratification**

  In this method of cross validation, we typically use 10 sets, and each set of data is stratified. This is a standard method of evaluation used in ML.

- **Leave-one-out cross validation.**

  It is a special form of n-fold cross validation where the number of folds is set as the same number of training examples. This gives us many advantages as it makes the best use of the data as the greatest possible amount of data is used for training and it becomes a deterministic procedure meaning no random sampling is involved and the same result will be obtained every time. The only disadvantage of this technique is the high computation cost, especially so for the large datasets.

- **Cross validation for parameter tuning.**

  Cross validation can also be used for searching through different parameter combinations and selecting the best one. The procedure which we use for this technique is the grid search with cross validation for parameter tuning. The steps for this procedure is as follows:

- Create the parameter grid (parameter combinations)
- Split the data into training and testing datasets.
- For each parameter combination train a KNN classifier on the training dataset using 10-fold cross validation.
- Compute the cross-validation accuracy.
- The best cross validation accuracy combination of parameters are chosen and the KNN model is rebuilt using the whole training dataset and the best parameters.
- Evaluate the model on the test data and report the result.

A few performance measures are as follows:

- Accuracy

  It is the proportion of correctly classified examples which means that the predicted class is the same as the actual class. We can calculate the accuracy on the training and testing data sets. Accuracy on the training data set is usually overly optimistic and not a good indicator of generalisation performance whereas the accuracy on the testing dataset is used to evaluate generalization performance.

Confusion Matrix

It has 4 different outcomes based on the 2 different classes which are yes and no. The 4 different outcomes are:

examples	# assigned to class yes	# assigned to class no
# from class yes	true positives (tp)	false negatives (fn)
# from class no	false positives (fp)	true negatives (tn)

The accuracy in terms of the above are given as:

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

The confusion matrix is not a performance measure but allows us to calculate performance measures. The following are obtained from the confusion matrix:

- Precision

  Precision is a metric that tells us about the quality of positive predictions. It is given by the formula:

$$P = \frac{tp}{tp + fp}$$

- Recall

  Recall is a metric that tells us about how well the model identifies true positives. It is given by the formula:

$$R = \frac{tp}{tp + fn}$$

- F1 Score classification.

  F1 score is a weighted average of precision and recall. F1 score is usually more useful than accuracy, especially if you have an uneven class distribution. It is given by the formula:

$$F1 = \frac{2PR}{P+R}$$

WEEK 5

Decision Trees

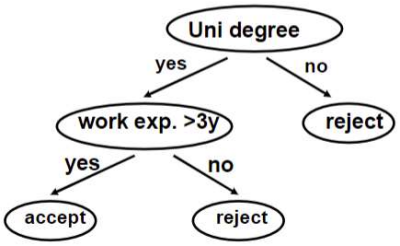
It is the most popular and well researched Machine learning and data mining algorithm. It is developed in parallel in machine learning by Ross Quinlan with ID3 algorithm. In statistics it is developed in parallel with the CART algorithm. Its most popular version is C4.5. It contains nodes and branches. Each non-leaf node corresponds to a test for the values of an attribute, branch corresponds to an attribute value and leaf node assigns a class.

A decision tree is used to predict a class for a new example. It starts from the root and test the values of the attributes until you reach the leaf node and then returns the class of the leaf node. It follows a top-down learning method using recursive divide and conquer process. The basic algorithm is as follows:

- Select the best attribute for the root node and create branch for each possible attribute value.
- Split the example into subsets, one for each branch extending from the node.
- Repeat recursively for each branch using only the examples that reach the branch.
- Stop if all examples have the same class, make a leaf node for this class.

An example for a decision tree will be as follows:

name	Uni degree	work exp. >3y	sex	class
David	yes	no	M	reject
Anna	no	yes	F	reject
Simon	yes	yes	M	accept
Kate	yes	yes	F	accept



An attribute needs to be selected on the following criteria:

- A leaf node with only 1 class (yes or no) will not have to be split further and the recursive process will terminate.
- The earliest will be selected.
- If we must choose the purity of each node, we can choose the attribute that produces the purest partitions.
- The measure of purity used is called information gain which is based on another measure called entropy.

Entropy

Entropy measures the homogeneity of a set with respect to the class. The smaller the entropy, the greater the purity of the data set. Entropy is also used in information theory, signal compression and physics. The formula is given by:

$$Entropy\ (H)\ of\ dataset\ (S) = I(S) = -\sum_i P_i * log_2 P_i$$

Where P_i is the proportion of examples that belong to class i.

Example:

Weather data: 9 yes and 5 no.

$$H(S) = -P_{yes} * log_2 P_{yes} - P_{no} * log_2 P_{no} = I(S) = I\left(\frac{9}{14}, \frac{5}{14}\right) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.940\ bits$$

While calculating entropy $log_2 0 = 0$.

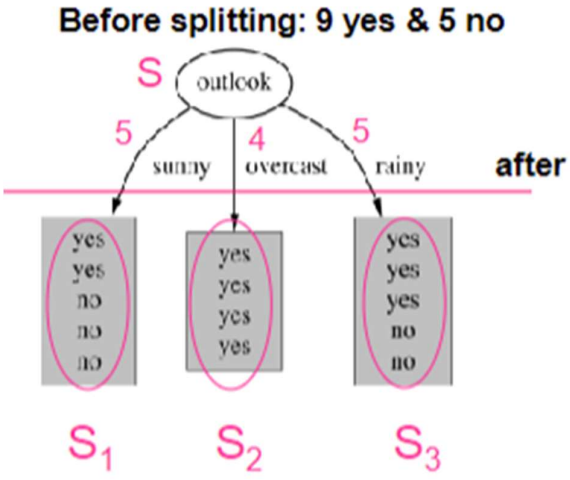
For binary tasks, range of entropy stays between 0 and 1 where H(S) = 0 means all elements belong to the same class (max purity, min entropy value) and H(S) = 1 means equal number of yes and no (min purity, max entropy value).

Information Gain

Information gain measures the reduction in entropy caused by using an attribute to partition the set of training examples. The best attribute is the one with the highest information gain (biggest reduction in entropy). Information gain is the difference between 2 values of entropies. It is given by the formula:

$$Gain = T1 - T2$$

Where T1 is the entropy od set of examples S associated with the parent node before the split and T2 is the remaining entropy in S after the split by the attribute. The larger the difference in entropy values, the more is the information gain. The best attribute is the one with the highest information gain. For example:



$$T1 = H(S) = I\left(\frac{9}{14}, \frac{5}{14}\right) = 0.940\ bits$$

$$H(S_1) = 0.971\ bits, H(S_2) = 0 bits, H(S_3) = 0.971\ bits$$

$$T2 = H(S|outlook) = \frac{5}{14} H(S_1) + \frac{4}{14} H(S_2) + \frac{5}{14} H(S_3) = \frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971) = 0.693\ bits$$

$$Gain = T1 - T2 = 0.940 - 0.693 = 0.247\ bits$$

Similarly, when calculated for the others, the information gain is as follows:

Temperature	0.029 bits
Humidity	0.152 bits
Windy	0.048 bits
Outlook	0.247 bits

And based on the above information gain values, the highest gain is from Outlook and hence that attribute is selected.

Calculating Information Gain

shape	color	class
circle	blue	+
circle	blue	+
square	blue	-
triangle	blue	-
square	red	+
square	blue	-
square	red	+
circle	red	+

x	y	-(x/y)* $\log_2(x/y)$	x	y	-(x/y)* $\log_2(x/y$	x	y	-(x/y)* $\log_2(x/y$	x	y	-(x/y)* $\log_2(x/y$
1	2	0.50	4	5	0.26	6	7	0.19	5	9	0.47
1	3	0.53	1	6	0.43	1	8	0.38	7	9	0.28
2	3	0.39	5	6	0.22	3	8	0.53	8	9	0.15
1	4	0.5	1	7	0.40	5	8	0.42	1	10	0.33
3	4	0.31	2	7	0.52	7	8	0.17	3	10	0.52
1	5	0.46	3	7	0.52	1	9	0.35	7	10	0.36
2	5	0.53	4	7	0.46	2	9	0.48	9	10	0.14
3	5	0.44	5	7	0.35	4	9	0.52			

1. What is the entropy of this collection of training examples?

$$H(S) = -\frac{5}{8}log_2\frac{5}{8} - \frac{3}{8}log_2\frac{3}{8} = 0.42 + 0.53 = 0.95\ bits$$

2. What is the information gain of the attribute shape?

$$T1 = H(S) = 0.95\ bits$$
$$H(S_{circle}) = -\frac{3}{3}log_2\frac{3}{3} - \frac{0}{3}log_2\frac{0}{3} = 0\ bits$$
$$H(S_{square}) = -\frac{2}{4}log_2\frac{2}{4} - \frac{2}{4}log_2\frac{2}{4} = 1\ bit$$
$$H(S_{triangle}) = -\frac{1}{1}log_2\frac{1}{1} - \frac{0}{1}log_2\frac{0}{1} = 0\ bits$$
$$T2 = H(S|Shape) = \frac{3}{8}(0) + \frac{4}{8}(1) + \frac{1}{8}(0) = 0.5\ bits$$
$$Gain = 0.95 - 0.5 = 0.45\ bits$$

Pruning Decision Trees

If we grow the decision tree to perfectly classify the training set, the tree may become too specific and overfit the data. Overfitting is the phenomenon of having high accuracy on the training data but low accuracy on the new data. This usually happens because the tree memorizes the data instead of extracting patterns. Overfitting occurs when the training data is too small and there is a lot of noise in the training data. Hence to avoid overfitting, we use the technique of tree pruning.

There are mainly 2 strategies of tree pruning:

- Pre-pruning
  It stops the tree growing in the initial stages before it reaches the point where it perfectly classifies the training data.
- Post-pruning
  In this strategy the tree is allowed to fully grow and perfectly cover the training data and then we prune it. Usually, this method is used in practice.

There are different post-pruning methods such as sub-tree replacement, sub-tree raising, converting the tree rules, and then pruning them, etc. In the decision tree algorithm, the data set is split into 3 parts: the training, testing and validation dataset. The training dataset is used to build the tree, validation set is used to evaluate the impact of pruning and decide when to stop and test data is used to evaluate how good the tree is.

Pruning by sub-tree replacement

It is a bottom-up approach where we start from the bottom of the tree to the root. Each non-leaf node is a candidate for pruning. The main idea is as follows:

- For each node, remove the sub-tree rooted at it.
- Replace it with a leaf with the majority class of examples.
- Compare the new tree with the old tree by calculating the accuracy on the validation set for both.
- If accuracy of the new tree is better or same as the accuracy of the old tree, keep the new tree.

Numerical attributes

Numerical attributes in the decision tree needs to be discretised (converted into nominal values). This is usually done by using the method of binary splits. Unlike nominal splits, numerical values have many possible splits. The main procedure of discretisation is:

- Sort the examples according to the values of the numerical attribute.
- Split points wherever the class changes halfway.
- Evaluate the information gain or other measure for every possible split point and choose the best one.
- Information gain for the best split point is the information gain for the attribute.

Alternatives to Information Gain

Gain ratio is a modification of information gain that reduces its bias towards highly branching attributes. It considers the number of branches when choosing an attribute and penalises highly branching attributes.

Ensemble Methods

An ensemble combines the predictions of multiple classifiers. The classifiers that are combined are called base classifiers and are created using the training data. There are various ways to make a prediction for new examples and ensembles tend to work better than the individual base classifiers are they combine. An example of an ensemble model would be one with the 3 base classifiers as k-nearest neighbour, logistic regression, and decision tree.

An ensemble model works better if the error of the base classifier is less than 0.5. This means that the predictions of the base classifier which is binary in nature are better than random guesses. For an ensemble to perform better than a single classifier:

- The base classifiers should have an error of less than 0.5.
- The base classifiers are independent of each other.

It is not possible to ensure total independence among the base classifiers and there have been cases of good results when there is a slight correlation between the base classifiers as well. The best way to create an ensemble model is that it consists of base classifiers that are reasonably correct and diverse (independent). A method for creating this ensemble by:

- Manipulate the training data – create multiple training sets by resampling the original data.
- Manipulate the attributes – use a subset of input features such as random forest and random subspace.
- Manipulate the class labels – error correction of output coding.
- Manipulate the learning algorithm – building a set of classifiers with different parameters.

Ensemble Methods – Bagging

Bagging is also called as bootstrap aggregation. Given a dataset 'D' with 'n' examples. Bootstrap examples 'D-'contains 'n' examples as well chosen randomly from 'D' where some examples will appear more than once or some not at all. The probability to choose an example is given by:

$$\left(1-\frac{1}{n}\right)^{n}$$

Method of implementing Bagging:

- Create M bootstrap samples.
- Use each sample to build a classifier.
- Get the predictions of each classifier and combine them with majority vote.

Bagging typically performs better than the single classifier and is never substantially worse. It is especially effective for unstable classifiers. Unstable classifiers are those for which small changes in the training set will result in large changes in predictions. If bagging is applied to regression tasks, the individual predictions are averaged than taking the majority vote.

Ensemble Methods – Boosting

Boosting is the most widely used ensemble method. The main idea is to make classifiers complement each other. This is done by having the next classifier created using examples that were difficult for previous classifiers. The most popular boosting algorithms are:

- AdaBoost

  This algorithm uses a weighted training set meaning that each training example has an associated weight to it which is greater or equal to 0. The higher the weight, the more difficult the example was to classify by the previous classifier and hence have a higher chance to be selected for the training set of the next classifier. The general flow of the algorithm is as given below:
  - Initially all examples in the training set have equal weights.
  - Using this training set the first classifier is generated.
  - It classifies some examples correctly and some incorrectly.
  - The method increases the weights of the incorrect examples.
  - Examples are selected for the training set of the next classifier.
  - Higher weighted examples are more likely to be selected.
  - These selected examples become the training set for the next classifier.
  - This process repeats until a certain number of classifiers are created.
  - Final ensemble is created by combining all the classifiers using a weighted vote based on how each of them performed on the training set.

  If the base learners are weak learners, then AdaBoost will return an ensemble that classifies the training data perfectly for a large enough number of iterations. Weak learner is a classifier whose classification performance is slightly better than random

guessing. Thus, AdaBoost boosts a set of weak learners into a strong learner. Theoretically, Boosting also works well if base classifiers are not too complex, their errors do not become too large too quickly as more iterations are done, etc.

- Gradient Boosting

  It uses decision trees as base learners, especially weak trees. As AdaBoost, it also works by sequentially adding base learners to the ensemble, each one focusing on the examples difficult to classify by the previous base learner. While AdaBoost updates the weights of the examples at each iteration, Gradient boosting adds a new model that minimizes the error in the previous model.

Both bagging and boosting use voting for classification and averaging for prediction to combine outputs of the individual learners. Both combine classifiers of the same type as well. Considering these 2 similarities, it does have its own set of differences as well. These are:

Bagging	Boosting
Base classifiers are created separately.	Base classifiers are created iteratively.
Combines equal weights to all base learners.	Combines different weights based on performance on training data.

Random Forest

In random forests each base classifier uses only a subset of the features. The basic flow of a random forest is as follows:

- Create feature subset by random selection from the original feature set.
- Build a classifier for each version of the training data.
- Combine predictions with majority vote.

The performance of random forests depends on the accuracy of the individual trees and the correlation between the trees. Ideally the decision trees are accurate and less correlated to each other. Bagging and random feature selection are used to generate diversity and reduce the correlation between the trees. As the number of features increases, the strength of the tree increases but so does the correlation between them. Random forests typically outperform a single decision tree and are robust to overfitting. It is as fast as only a subset of the features are considered.

WEEK 6

Maximum Margin Hyperplane

If a training data has been given, it can be separated into 2 based on a linear boundary. This linear boundary can either be a line or a hyperplane that separates the data. There are multiple possible decision boundaries that can be used.

Support Vectors are the examples (data points) that lie closest to the decision boundary. Margin is the separation between the boundary and the closest data point. The boundary line lies in the middle of the margin. It is possible to have more than one support vectors and it is a characteristic feature for the support vector to touch the margin of the decision boundary. The support vectors are a subset of the input vectors; hence they are also vectors of the dataset. The training examples are nothing but vectors where each dimension corresponds to 1 feature of the dataset.

Based on the established theory, we know that there are multiple boundary lines that can be created in a dataset. The main question arises, which one to choose? The choice is based

on the hyperplane having the biggest margin. This hyperplane with the biggest margin is called the maximum margin hyperplane and is the hyperplane with the highest possible distance to the training examples. Support Vector Machine selected the maximum margin hyperplane.

Decision boundaries with large margins are better than those with small margins as they are typically more accurate on new examples. This is because when the margin is small, then any slight change in the hyperplane or the training dataset at the boundary is more likely to affect the classification as there is a very narrow room allowed for data perturbations. This also is one of the reasons why a smaller margin is prone to overfitting. On the other hand, having a larger margin means more leeway for data perturbations and hence allowing them to have a better generalization performance.

Linear Decision Boundary

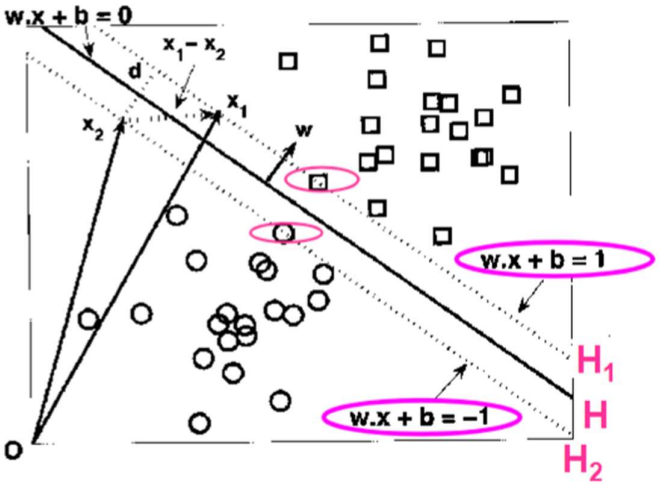
Suppose there is a binary classification problem with N training examples (input vectors). X is the input vector and Y is the class value. Then a decision boundary of a linear classifier is given as:

$$w * x + b = 0$$

Where: 'w' and 'b' are parameters and 'x' is the input vector. Learning a linear model is equivalent to finding these parameters. If we know the decision boundary, we can easily classify a new example by calculating f=wx+b and determining the sign:

- If 'x' is above the decision boundary: $w * x + b > 0$
- If 'x' is below the decision boundary: $w * x + b < 0$

SVM: A problem statement



$H_1: w * x + b = 1$, $H_2: w * x + b = -1$ are the margins and the points laying on these two are the support vectors. 'd' is the margin of the line and hence it can be shown that:

$$d = \frac{2}{||w||}$$

Hence through the expression we can understand that if we want to increase the 'd' we have to decrease the ||w|| value. The optimization problem can be solved using Quadratic Programming and Lagrange Multiplier Method. The problem is transformed into an equivalent form using the Lagrange multipliers λ. Then the values of λ are found using Quadratic programming and this results in the solution for the optimal decision boundary as:

$$w = \sum_{i=1}^{N} \lambda_i y_i x_i$$

Where:

- 'w' is the optimal decision boundary (maximum margin hyperplane) which is a linear combination of support vectors.
- $'\lambda'$ is the Lagrange multiplier, most of which are 0 due to the linear combination of a small number of training examples.
- $'x_i' \& non-zero\ '\lambda_i'$ training examples are support vectors and closest to the decision boundary 'w'.

Hence when classifying a new example:

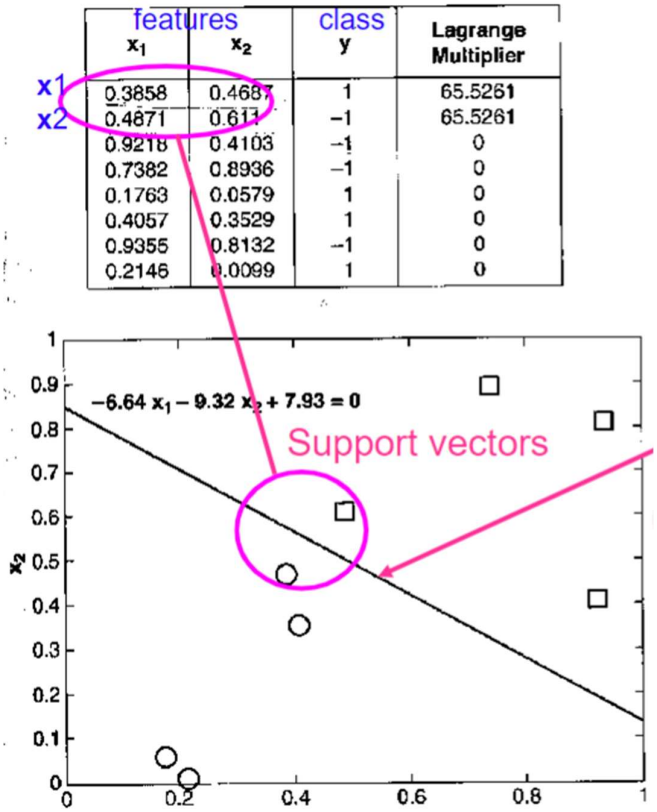
$$f = w * z + b$$

$$f = \sum_{i=1}^{N} \lambda_i y_i x_i * z + b$$

Where:

- $\sum_{i=1}^{N} \lambda_i y_i x_i * z$ is the dot product of the new vector and the support vector.
- Sign of 'f' depends on whether f<0 or f>0.

Example calculation



Considering that only 2 classes have Lagrange values not equal to 0, the w values would be:

$$w_1 = \sum_{i=1}^{2} \lambda_i y_i x_{i1} = 65.5261(1 * 0.3858 - 1 * 0.4871) = -6.64$$

$$w_2 = \sum_{i=1}^{2} \lambda_i y_i x_{i2} = 65.5261(1 * 0.4687 - 1 * 0.611) = -9.32$$

$$b = 7.93$$

Seeing the support vectors in the table: the square above the decision boundary is +1 and the circle below the decision boundary is -1.

SVM Soft margins

The methods discussed so far are those where no miss-classifications are allowed. We can allow some by modifying our method. The modified method will construct a linear boundary even if the data is not linearly separable. For such cases there is an additional parameter 'C' which is a hyper-parameter that allows for a trade-off between maximising the margin and minimizing the training error. The larger the value of 'C' the more emphasis has been placed on minimizing training error than maximising the margin.

Non-Linear Support Vector Machines

Most of the problems are linearly non-separable. To combat this challenge, SVM has extended to find a non-linear boundary as well. For this the data is transformed from its original feature space to a new space where a linear boundary can be used to separate the data. It is more likely to find a linear decision boundary in a transformed data if the transformation is non-linear to a higher dimensional space. The linear decision boundary found in the new feature space is mapped back to the original feature space resulting in a non-linear decision boundary in the original space. The mapping function is represented by the symbol: ϕ.

When learning non-linear SVM even if we know the value of ϕ, the new space is assumed to be at a higher dimension and computing dot products of vectors in high dimensional space is computationally expensive. We do this computation using the kernel trick.

Kernel trick

It is a method for computing the dot product of a pair of vectors in the new space without first computing the transformation of each vector from the original to the new space. First, we compute the dot product of the original features and use it in a 'kernel function' to determine the dot product of the transformed features. The kernel function specifies the relationship between the dot products in the original and transformed space. For example:

We need to transform a 2-dimensional original space to 3-dimensional new space.

$$\phi: (x_i, y_i) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

The spaces are given as:

$$u, v \rightarrow Vectors\ of\ original\ space$$
$$\phi(u), \phi(v) \rightarrow Transformed\ vectors\ in\ new\ space$$

The dot product of $\phi(u) * \phi(v)$ will be given as:

$$\phi(u) * \phi(v) = (u_1^2, \sqrt{2}u_1u_2, u_2^2)(v_1^2, \sqrt{2}v_1v_2, v_2^2)$$
$$= u_1^2v_1^2 + 2u_1u_2v_1v_2 + u_2^2v_2^2$$
$$= (u_1v_1)^2 + (u_2v_2)^2 + 2u_1u_2v_1v_2$$
$$= (u_1v_1 + u_2v_2)^2$$
$$= (u * v)^2$$

This means that the dot product of the new space can be expressed via the dot product of the original space. Hence the kernel function of our example would be:

$$K(u, v) = (u * v)^2$$

Kernel functions allow the dot product in the new space to be computed without first computing for each input vector. As they specify the relationship between the dot products in the original and new feature space, we can compute a function of dot product in the original space and use it to evaluate the dot product in the new spaces. Hence, we learn in higher dimensional space by computing kernel functions in lower dimensional space. Functions K for which the above relationship is true, need to satisfy the Mercer's Theorem which restricts the class of functions we can use. Some kernels which satisfy the Mercer's theorem are:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^p - \textit{polynomial kernel}$$

$$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} - RBF$$

$$K(\mathbf{x},\mathbf{y}) = \tanh(k\mathbf{x}\cdot\mathbf{y}-\theta) - \textit{tanget hyperbolic}$$

$$(\textit{satisfies Mercer's Th. only for some } k \textit{ and } \theta)$$

Kernelization can be applied to all algorithms that can be reformulated to work only with dot products. Once this is done, the dot product is replaced with a kernel function.

SVM training and classification using kernel functions.

- Training:

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j=1}^{N}\lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$

$$\textit{subject to } \lambda_i \geq 0, \sum_{i=1}^{N}\lambda_i y_i = 0$$

- Optimal hyperplane in the new space: $\mathbf{w} = \sum_{i=1}^{N}\lambda_i y_i \Phi(\mathbf{x}_i)$

- Classifying new example z:

$$f = \mathbf{w}\cdot\mathbf{z} + b = \sum_{i=1}^{N}\lambda_i y_i K(\mathbf{x}_i,\mathbf{z}) + b$$

Dimensionality Reduction

Some Machine learning problems have 1000s of features involved in the data and this makes the training very slow. Dimensionality reduction removes the redundant and highly correlated features and reduces the noise in the data. There are various problems to high dimensional data such as:

- Slower training of the model.
- Unreliable classification of the examples.
- Higher chances of overfitting of the data.
- Building interpretable models are not possible.
- Humans can't visualize high dimension data and can only interpret low dimensional data.
- Not all features are important and may lead to increasing the noise in the data.

Principal Component Analysis

PCA is the most popular dimensionality reduction method. It is often called a feature projection method. The main idea of this method is to find a new set of dimensions and project data into it. The dimensionality of the new space is smaller than the original space and the new axes captures the essence of the data. The resulting dataset (projection) can be used as an input to train a ML algorithm. In short, the PCA method deals with the construction of new features which are smaller than the number of original features.

Given N examples with dimensionality 'm', we need to find 'm' new axes $z_1, z_2, \dots z_m$ orthogonal to each other such that $var(z_1) > var(z_2) > \cdots > var(z_m)$ where $z_1, \dots, z_m$ are called principal components. These principal components are vectors that define a new coordinate system. The way this method works is that it selects the 'k' largest principal components $z_1, z_2, \dots, z_k$ and project data points on them (k<m). Based on the dimensionality of the data, we choose lower dimension. There are multiple principal components and there are methods to choose them, these are:

- Setting a minimum percentage of variance that should be preserved.
- Using the elbow method wherein the same number of dimensions are plotted as the function of the variance. A graph is plotted where an elbow curve is created and those number of dimensions are selected.

Principal components are found using a standard matrix factorization method called Singular Value Decomposition (SVD). The main theorem of this method is that any 'n x m' matrix X (where n >= m) can be written as the product of 3 matrices given as:

$$X = U * \Lambda * V^T$$

Where:

- X = original data.
- U = transformed data.
- V = defines the new set of axes.

Here X can also be written as the following:

$$X = \lambda_1 u_1 v_1^T + \lambda_2 u_2 v_2^T + \cdots + \lambda_m u_m v_m^T$$

Where λ is arranged in the decreasing order.

Data reduction comes from taking only the first k components (thus k<m) forming the equation as:

$$X_{reduced} = \lambda_1 u_1 v_1^T + \lambda_2 u_2 v_2^T + \cdots + \lambda_k u_k v_k^T$$

SVD for compression

Consider an image compression (greyscale image). The uncompressed image is given by 'n*m' pixels and 'n*m' integer numbers need to be stored. A compressed image using the first k components is formed where we need to store the following:

- 'k' singular values from the Λ matrix.
- The first 'k' columns of the 'U' matrix.
- The first 'k' columns of the 'V' matrix.
- Total $k * (1 + n + m)$

This will lead to a compression ratio:

$$Compression\ ratio\ (r) = \frac{After\ compression}{Before\ compression} = \frac{k(1+n+m)}{n*m}$$

Facial Recognition

The main task over here is to determine if a new image (previously unseen) belongs to a known image or not from the database. Main applications of facial recognition lie in photo collection, social media, security, etc. A possible solution for facial recognition is to build a separate classifier for each person, however, there are innumerable people in the dataset and hence does not make sense to make those many classifiers. There are multiple possible methods, a few are:

- Face recognition using nearest neighbour.
- Face recognition using PCA and nearest neighbour.

WEEK 7

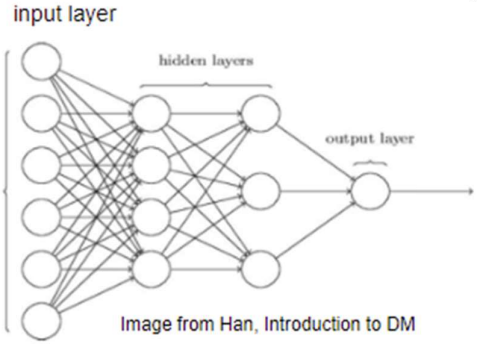
Deep Learning

Deep learning is an approach to machine learning that is inspired by how the brain operates, it refers to the modern artificial neural networks and emphasizes that these networks are deeper and have more layers than the previous networks. The depth enables them to learn more complex input-output mappings. Part of Artificial Intelligence focusses on creating large Neural Networks that can create data driven decisions. Deep Learning is suitable for complex data and large datasets.

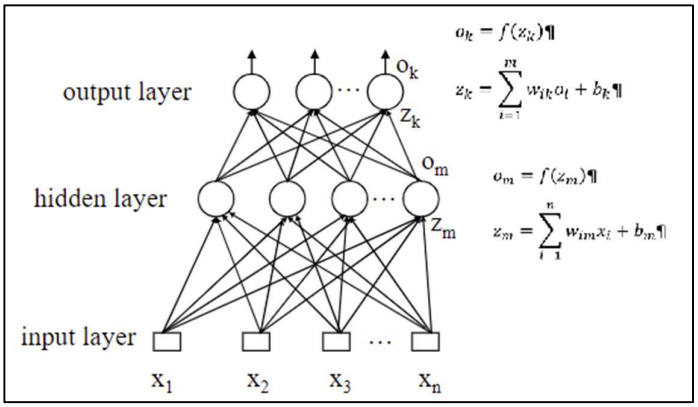
Deep Neural Networks can learn hierarchical feature representations. By making the Neural Networks deep by adding hidden layers, subjects the features to a sequence of transformations and allows to learn important features. Deep learning has been successful in various areas and is used by tech behemoths such as Facebook, Google, Baidu, Microsoft, Tesla, etc. Most deep learning methods use multilayer perceptrons as building blocks. The basics of Deep Learning are explained below.

Neural Network

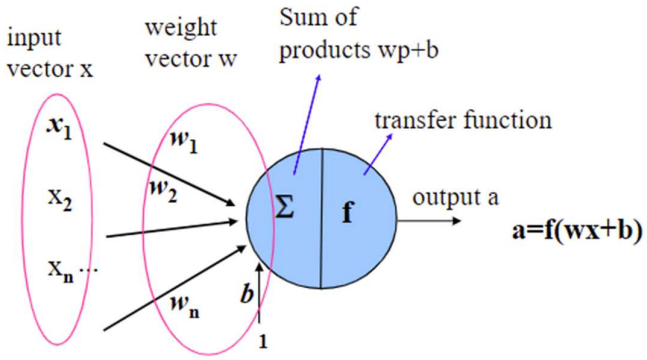
Neural networks consist of multiple neurons (units, nodes) that relate to each other with directed links (called edges) where each connection has an associated numerical weight. The neurons are organised in layers. These layers are Input, output, and hidden layers. The main idea of Neural networks is that Neural Networks need to be trained using optimal weights to perform a certain task. The neural network looks like the following:



The network architecture usually has the input layer, the output layer and 1 or more hidden layers. In Feed-Forward Neural Network, each neuron receives input only from the previous layer. A fully connected network is one wherein each neuron in the current layer is connected to all the neurons from the previous layer. An example of network architecture is given below:



Neuron



The blue circle is the neuron. It contains two parts, one the input and the other is the output. Input of the neuron is a weighted combination of vector inputs, and the output is a transfer function. W & b are the parameters of the neuron, and they are learned using the learning rule for the specific type of Neural Network.

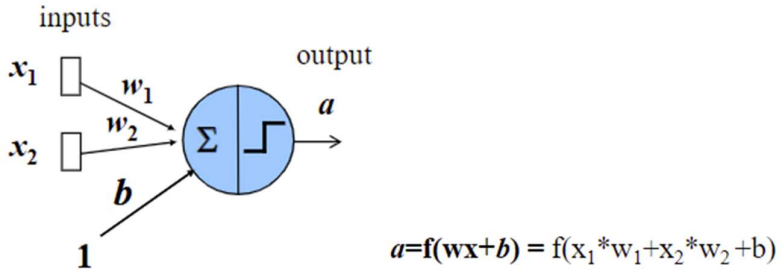
A single neuron calculates the weighted sum of the outputs of the previous layer, which is passed through the transfer function. The transfer function needs to be differentiable. The most widely used transfer function is the sigmoid function.

Perceptron

The simplest neural network is called a perceptron and it uses a step transfer function. It has a binary output 0 and 1 (or -1 and 1). The output is a weighted sum of its inputs and subject to the step transfer function. The step transfer function is given by:

$$a = f(n) = \begin{Bmatrix} 1 & if & n \geq 0 \\ 0 & if & n < 0 \end{Bmatrix}$$

An example of a perceptron is given below:



Hence an example would be:

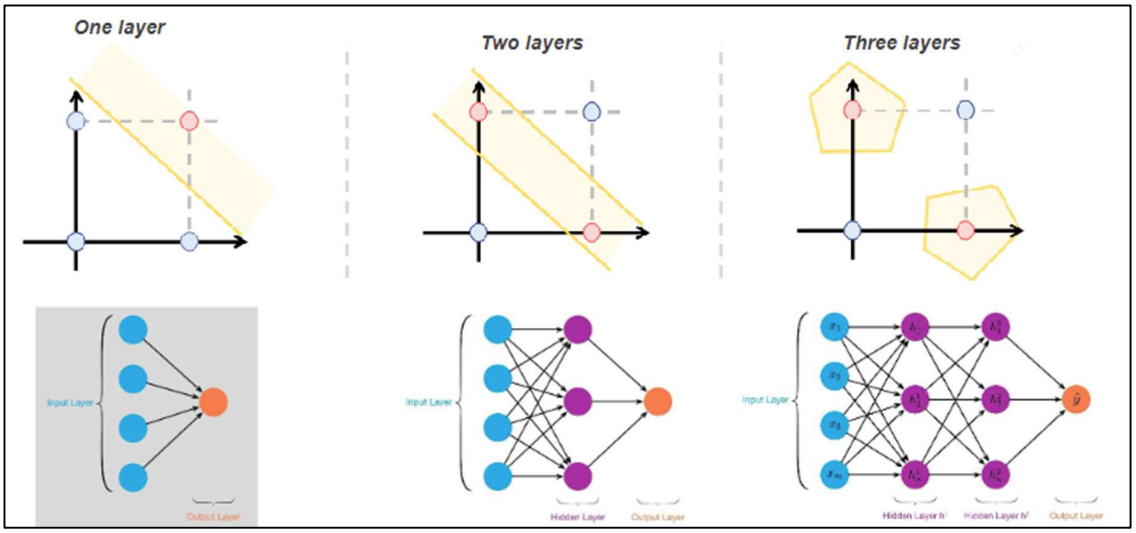
Example:
x_1=0.2 and x_2=0.3
w_1= 2, w_2=1, b = -1.5
a = step(0.2*2+0.3*1-1.5)=step(-0.8)=0

$$\mathbf{w}^{new} = \mathbf{w}^{old} + e\mathbf{x}^T$$
$$\mathbf{b}^{new} = \mathbf{b}^{old} + e$$

The perceptron learning algorithm is:

- Initialize the weights w and bias b to small random values and set epoch to 1.
- For each training example, calculate "a", compute the error and update the weights.
- At the end of each epoch, check if the stopping condition is satisfied.
    - The stopping condition is checked at the end of each epoch (One pass through the whole training set).
    - All training examples are passed again one-by-one and the perceptron's output is calculated and compared with the target output.
    - If all examples are correctly satisfied or maximum number of epochs are reached, the program stops.

The perceptrons will only work if the training examples are linearly separable. The perceptron learning rule is guaranteed to converge to a solution which is a set of weights which correctly classifies the training example in a finite number of steps. Most of the problems are not linearly separable and hence is a limitation of the perceptrons. We can add more layers and thus form more complex boundaries. An example is given below:



Back propagation Algorithm

The Back Propagation Algorithm is used to train a multi-layer perceptron neural network. The main idea of this algorithm is that for each training algorithm, an input is propagated through the network and an output is calculated. This is compared with the target output and error is calculated. The weights are then updated to reduce the error until the error for overall examples is less than the threshold value. The weights are updated backwards, from the output to the input neurons by propagating the weight change to minimize the error and hence is called a back propagation algorithm. The formula for which is:

$$w_{pq}^{new} = w_{pq}^{old} + \Delta w_{pq}$$

The algorithm for training Neural network using Backpropagation is:

- Initialize all weights to small random values.
- Repeat until the stopping condition is satisfied.
  - Forward pass
    - Present a training example and compute the network output.
  - Backward pass
    - Compute the Sigma values for the output neurons and update the weights to the output layer.
    - Starting with the output layer, repeat for each layer of the network:
      - Propagate the Sigma values to the previous layer.
      - Update the weights between the 2 layers.
- Check the stopping condition at the end of each epoch.

Gradient descent

We define the error function over all the training examples. We can minimize the standard errors b using the steepest gradient descent algorithm. This is also called as the standard optimization method. The neural weights are iteratively updated by moving downhill, in the direction that reduces errors the most. The step that is used to move downhill is called the learning rate. It is a hyperparameter of the algorithm. The gradient descent algorithm is not guaranteed to find the global minimum, it converges to the closest local minimum depending on the starting position.

Weight update formula

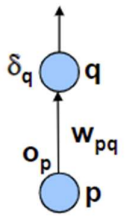
$w_{pq}(t)$ - weight from neuron p to neuron q at time *t*

$w_{pq}(t+1) = w_{pq}(t) + \Delta w_{pq}$

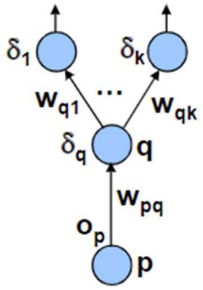
$\Delta w_{pq} = \eta \cdot \delta_q \cdot o_p$ - weight change

- The weight change is proportional to the output activation of neuron p and the error δ of neuron q
- δ is calculated in 2 different ways:

- q is an output neuron:
  $\delta_q = (t_q - o_q) f'(z_q)$



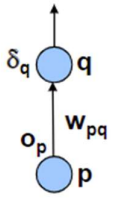
- q is a hidden neuron:
  $\delta_q = f'(z_q) \sum_i w_{qi} \delta_i$
- (*i* is over all neurons in the layer above q)



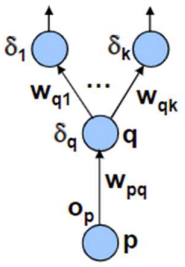
- $f'(z_q)$ is the first derivative of the activation function used in neuron q with respect to its input z_q

Weight update for sigmoid transfer function

- It can be shown that: $f'(x) = f(x)(1 - f(x))$
- => Weigh update formulas for sigmoid transfer function:
- q is an output neuron:
  $\delta_q = (t_q - o_q)o_q(1 - o_q)$



- q is a hidden neuron:
  $\delta_q = o_q(1 - o_q)\sum_i w_{qi}\delta_i$
- (*i* is over all neurons in the layer above q)

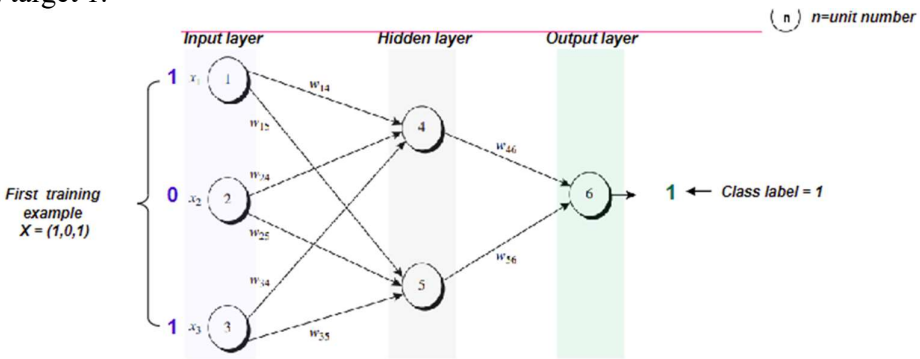


Training Neural Networks with Backpropagation algorithm

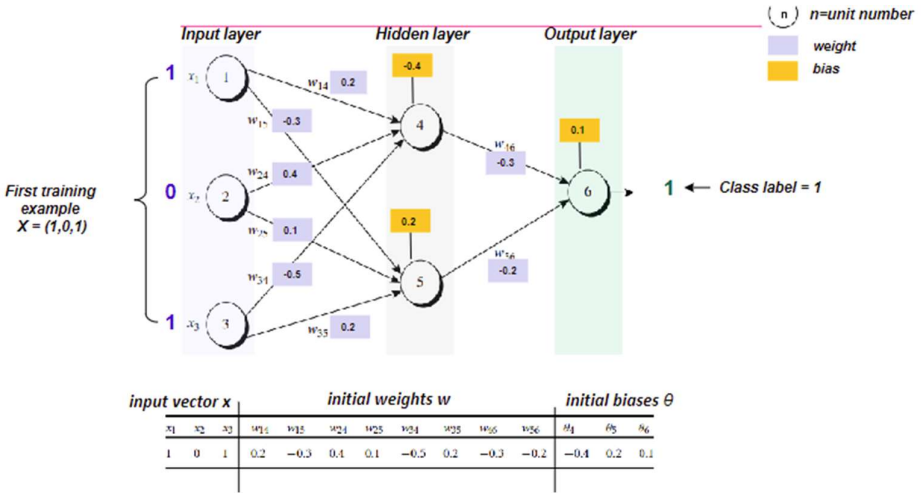
1. Initialize all weights to small random values.
2. Repeat until the stopping condition is satisfied.
   a. Forward pass
      i. Present a training example and compute the network output.
   b. Backward pass
      i. Compute the delta values for the output neurons and update the weights to the output layer.
      ii. Starting with the output layer, repeat for each layer in the network.
         1. Propagate the delta values back to the previous layer.
         2. Update the weights between the two layers.
3. Check the stopping condition at the end of each epoch.

Example of Backpropagation algorithm

Taking into consideration a neural network with 1 hidden layer and trained with backpropagation algorithm. Assuming that the learning rate is 0.9 and input example is x=(1,0,1) target 1.



Initialize the weights w and biases θ to small random values.



Sigmoid transfer function is used for hidden and output neurons. The calculations for the output of neurons for 4, 5, and 6 is calculated. The calculations are as follows:

Input to neuron 4: z_4=1*0.2+0*0.4+1*(-0.5)-0.4=-0.7, output of neuron 4: o_4=1/(1+$e^{0.7}$)=0.332
Input to neuron 5: z_5=1*(-0.3)+0*0.1+1*0.2+0.2=0.1, output of neuron 5: o_5=1/(1+$e^{-0.1}$)=0.525 *NN output*
Input to neuron 6: z_6=0.332*(-0.3)+0.525(-0.2) +0.1=-0.105, output of neuron 6: o_6=1/(1+$e^{0.105}$)= **0.474**

For the backward pass we compute the delta values for the output layer neurons and update the weights to the output neuron. The calculations are given below:

- δ_6= (t_6-o_6)*o_6*(1-o_6)=(1-0.474)*0.474*(1-0.474)=0.1311
- Δw_{46}=η* δ_6*o_4= 0.9*0.1311*0.332=0.039, w_{46}new=w_{46}old+ Δw_{46}=-0.3+0.039=-0.261
- Δw_{56}=η* δ_6*o_5= 0.9*0.1311*0.525=0.0619, w_{56}new=w_{56}old+ Δw_{56}=-0.2+0.0619=-0.138
- $\theta_6$new= $\theta_6$old + $\Delta\theta_6$ = $\theta_6$old +η* δ_6*1=0.1+0.9*0.1311*1=0.218

Following this, compute the delta values for hidden neurons and update the weights to the hidden layer. The calculations are given below:

- δ_4=o_4*(1- o_4)*w_{46}* δ_6=0.332*(1-0.332)*(-0.3)*0.1311=-0.0087
- Δw_{14}=η* δ_4*o_1= 0.9*(-0.0087) *1=-0.0079, w_{14}new=w_{14}old+ Δw_{14}=0.2-0.0079=0.1921
- Δw_{24}=η* δ_4*o_2= 0.9* *(-0.0087) *0=0, w_{24}new=w_{24}old+ Δw_{24}=0.4+0=0.4
- Δw_{34}=η* δ_4*o_3= 0.9* *(-0.0087) *1=-0.0079, w_{34}new=w_{34}old+ Δw_{34}=-0.5-0.0079=-0.5079
- $\theta_4$new= $\theta_4$old + $\Delta\theta_4$= $\theta_4$old +η* δ_4*1=-0.4+0.9*(-0.0087)*1=-0.4078

Stochastic Gradient Descent

The standard gradient descent algorithm sums the error of all the training examples and then updates the weights. The stochastic gradient descent updates the weights after each example. Stochastic gradient descent is usually faster. Mini-batch Stochastic gradient Descent is also called Batch Gradient Descent and in this, sum of the error of mini-batches of training examples are calculated and weights are updated.

Universality of Multi-Layer perceptrons

Multi-layer perceptrons trained with backpropagation algorithm are universal approximators. It is based on 2 theorems: Any continuous function can be approximated with arbitrary small error by a network with on hidden layer. And any function can be approximated to arbitrary small error by a network with 2 hidden layers. These are called the existence theorems. They don't say how to choose the network architecture and hyperparameters.

Improving the Neural network:

- Design issues in Architecture
    - Number of neurons in the input layer depends on the dimension. If numerical attributes are used, 1 neuron per attribute is needed. If categorical attributes with k values are taken, k neurons per attribute will be required. Number of neurons in the output layer depends on the task of the problem. We need k neurons for a k class problem, 1 for each class.

    - Deciding the number of hidden layers and neurons is difficult and needs to be done using trial and error. If there are too many hidden layers and neurons, it might lead to overfitting. On the other hand, it might lead to underfitting. The main idea to tackle this problem is to grow the hidden layer. Start with a small network and train until the error rate no longer improves then add neurons which are randomly initialized.
- Presentation of training example
    - The standard approach is to run each epoch through the same training example one by one, always in the same sequence. There are other approaches such as:
    - For each epoch, permutate the training examples.
    - Present examples with higher error more often and examples with lower error less often.
    - Present the examples in batches of N examples, summing up their individual errors and updating after each batch rather than one by one.
- Learning rate
    - The performance is very sensitive to learning rate. If it is too small there will be low convergence, if it is too big, oscillation or overshooting the minimum might take place. It is not possible to determine the best learning rate before training as it changes and depends on the error surface.
    - In the standard approach, we use a constant learning rate during training. There are multiple different approaches such as using a variable learning rate which is time dependent or use different decay schedules.
    - It is part of the optimization technique to help the backpropagation algorithm.
- Momentum
    - This is another type of optimization technique.
    - In this we make the current update dependent on the previous by introducing a hyperparameter called momentum and a momentum term in the weight update equation. This helps in reducing the oscillations and allows us to use a larger learning rate.
- Weight initialization
    - The performance of neural networks is very sensitive to the weights initialization. There are different strategies used for weight initialization, such as the standard approach of randomization, or the Xavier initialization where the weights are sampled from normal distribution cantered at 0 with standard deviation $\sigma = \sqrt{\frac{2}{N_{in}+N_{out}}}$

Deep Learning

Deep Learning are Neural Networks with many hidden layers. The main motivation for this was the better computational power. Complex high-level features can be constructed by combining lower-level features and greater number of hidden layers can give rise to deeper hierarchy of features. With higher computational power and availability of large and high-

quality datasets, deep learning can be successful. Another reason for the success of deep learning is the algorithmic advances.

Vanishing Gradient Problem

When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. We call this the vanishing gradient problem.

This is specifically problematic if there are many hidden layers. This will diminish the gradients and slow the convergence. Even if the activation in the output layer does not saturate, the repeated multiplications performed as we backpropagate the gradients from the output to the hidden layers may lead to diminishing gradients. The vanishing gradient problem is basically the slower learning speed of higher hidden layers due to the small weight changes. The solution to this is to use Piece-wise linear functions such as Rectified Linear Unit (ReLu) or Leaky Rectified Linear Unit (LReLu). These functions can be used as the have no upper bound and no saturation of the output. The gradient of ReLu is 1 for any input greater than 0 and there won't be any saturation of hidden nodes in this case.

Dropout

This is a method to prevent overfitting. The main idea is to avoid learning spurious features at the hidden nodes through intuition. Relevant features are more resilient to the removal of neurons, and they perform well for different combinations of neurons. While spurious features depend on certain neurons. Dropout forces the Neural Network to be less dependent on certain neurons to collect more evidence from the other neurons and to be more robust to noise. During training, at each iteration of the backpropagation, random neurons are selected in each layer and their values are set to 0. This results in a thinned sub-network of a smaller size. During training, the weights and biases are updated using back propagation algorithm and then new weight values are added to the original network. During testing, no neurons are dropped out and the network is scaled down based on the dropout rate.

SoftMax

The neural network outputs are processed to turn them into probabilities. The main motivation for this is to interpret the outputs as probabilities that sum up to 1. The SoftMax transformation is: $p_i = \frac{e^{o_i}}{\sum_j e^{o_i}}$.

Alternative Loss Functions

Using cross entropy loss instead of MSE. This can be done for classification tasks.

Categorical cross entropy loss for the classification of example *i*:

$$CCE_i = -\sum_{j=1}^{C} y_{ij}.log\ \hat{y}_{ij}$$

Cross entropy loss function - sum of losses of individual examples:

$$CCE = \sum_{i=1}^{N} CCE_i$$

WEEK 8

Convolutional Neural Networks

It was first introduced in 1989 and gained popularity in 2012. It is a special type of multilayer Neural Network which trained with the backpropagation algorithm. It was designed to recognize visual patterns directly from pixel images with minimal pre-processing. It can recognize patterns with high variability and be used in image and speech recognition as well. CNNs are designed for images. Images are represented as matrix of pixel values. It is encoded as black (0) and white (255) images in 1 matrix. The colour images can also be processed but for 3 channels: red, green, and blue making up to 3 matrices.

Traditional Neural Networks do not consider the spatial structure of the image. CNNs do this by using new types of layers: convolutional and pooling. Traditional Neural Networks are sensitive to the position of an object in the image whereas CNN are more robust to shifts/translation. Robustness to translation is important for image and sound data as translation is one of the main sources of distortion. Moving even a single pixel will lead to input values being completely different and hence renders the already learned weights useless.

Structure of CNN

The main structure of a CNN involves the following:

- Input layer for the image values. (INPUT)
- Convolutional Layer. (CONV)
- ReLu or LReLu layer – The non-linear activation function. (RELU)
- Pooling Layer. (POOL)
- Fully connected layer with SoftMax activation function at the output. (FC)

INPUT → CONV → RELU → POOL → FC

In this the CONV→RELU→POOL block can be repeated several times and the FC can contain one or more layers. CNN is like a fully connected Neural Network. They are made up of neurons that have learnable weights. Each neuron receives some inputs (the raw pixel value), performs a dot product, and follows it with a non-linear activation function. The network expresses a single differentiable function and is trained with the backpropagation algorithm.

The Convolutional layer is like a filter of the matrix. The convolution (filter) slides across the matrix to produce the feature map. For example:

The value in the feature map is created by the calculation as seen above. The calculation for this example would be: (1x1)+(1x0)+(1x1)+(0x0)+(1x1)+(0x1)+(1x1)+(1x0)+(1x1) = 4. The yellow region in the green matrix is the receptive field. The receptive field is the image region that is currently being convolved with a filter. Strides represent the number of boxes in the matrices that have been moved. We represent them as vertical or horizontal strides. During these strides, the filter may go beyond the original image. When this happens, the

excess rows and columns are filled with 0 as Padding. Padding allows for better coverage of the image around the image and feature extraction.

Convolution

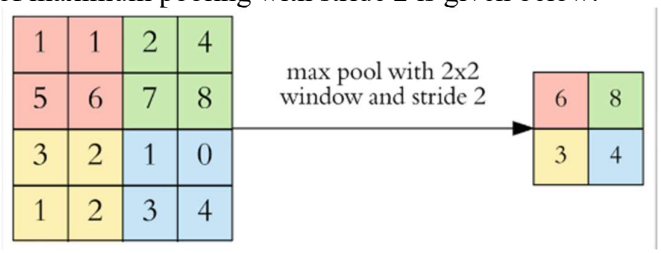
The main purpose of convolution is to use the filter as a feature detector. The feature map will show the image regions in which there are specific features. High values represent the presence of some feature and Low values represent the presence of no features. In CNNs the filters are not predesigned to detect a particular type of feature, their values are learned during training with the backpropagation algorithm.

ReLu Activation Function

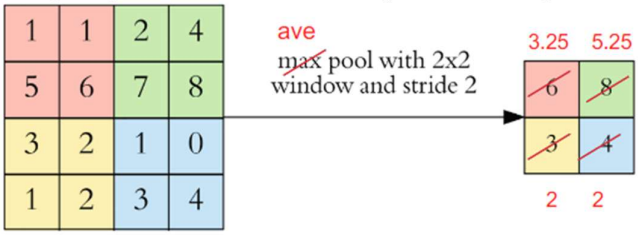
ReLu Activation Function is applied o the feature maps. It is simpler than sigmoid or tanh function as it gives faster computation. ReLu simplifies the gradient descent calculation. Using the ReLu function, the upper bounds are removed and there is no saturation of the output and vanishing gradient is decreased.

Pooling

The exact location of the feature is not important, and an approximation is required. We can use pooling layer to summarize the input of the previous layer. This reduces the number of parameters and help prevent overfitting. Pooling improves robustness to shifts in the receptive fields. An example of maximum pooling with stride 2 is given below:



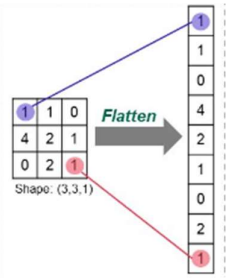
There are 2 types of pooling. Max pooling where the maximum value of the region is taken. An example of this is shown above. The other version of pooling is average pooling where the average value of the image region is taken. An example of this is given below:



The version of pooling to choose depends on the task. Max Pooling selects the max value (brightest pixel). This makes it useful for images with dark background. Average pooling smooths the image and hence could be more useful for images with white background.

Fully Connected Layer

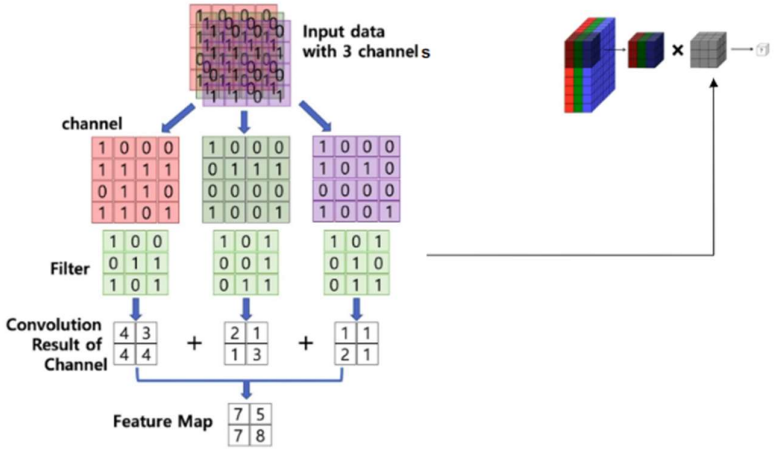
At the end of the CNN, we attach 1 or more fully connected layers. The first FC layer takes the output of the last layer (either the convolution layer or the pooling layer). It flattens out the output by transforming the multi-dimensional matrix of features into a vector that can be fed in the FC Layer. An example of flattening is given below:



The last FC layer outputs a n-dimensional vector where n is the number of classes. It typically uses a SoftMax function to output probabilities for each class. Essentially, the FC layers looks at the features extracted by the convolutional layers, combine them and output the probabilities for each class.

Multi-channel input CNNs

Colour images have 3 channels, red, green, and blue. The input of colour images is in the form of [32x32x3] where red is [32x32], green is [32x32], and blue is [32x32]. CNN architecture can be modified to work with multiple input channels. An example is given below:



Using multi-channel data increases the dimension of the filters too - the third dimension should be the same.

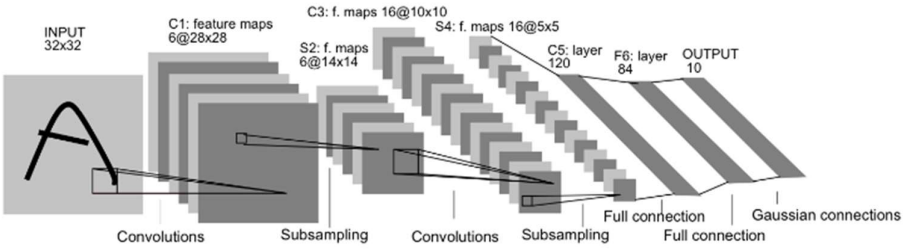
CNN Architectural Issues

- What is the right way to combine CNN components?
- How many convolutional layers?
- How many filters in each layer?
- What should be the size of these filters?
- How frequently should pooling be done?
- What should be the size of pooling layers?

CNN Architecture

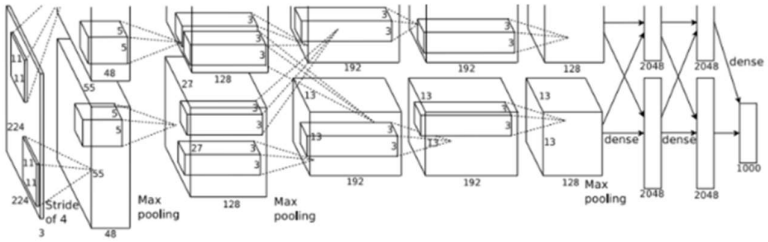
- LeNet-5

This CNN architecture is usually used for handwritten digit recognition. The CNN model consists of 2 convolutions, 2 max-pooling layers, and 3 fully connected layers. The image representation of the CNN architecture is given below:



- AlexNet

  The main purpose of this CNN architecture was to classify images into different categories. The input consists of 277x277 images, and the architecture consists of 5 convolution layers, 3 max-pooling layers, and 3 fully connected layers. An image of the CNN architecture is given below:



- VGG

  This CNN architecture is an improved version of AlexNet. It is a deeper network with simpler design. The convolutions have 3x3 filters with stride=1 and padding=1. The max-pooling layers are of 2x2 with stride=2.

- GoogLeNet

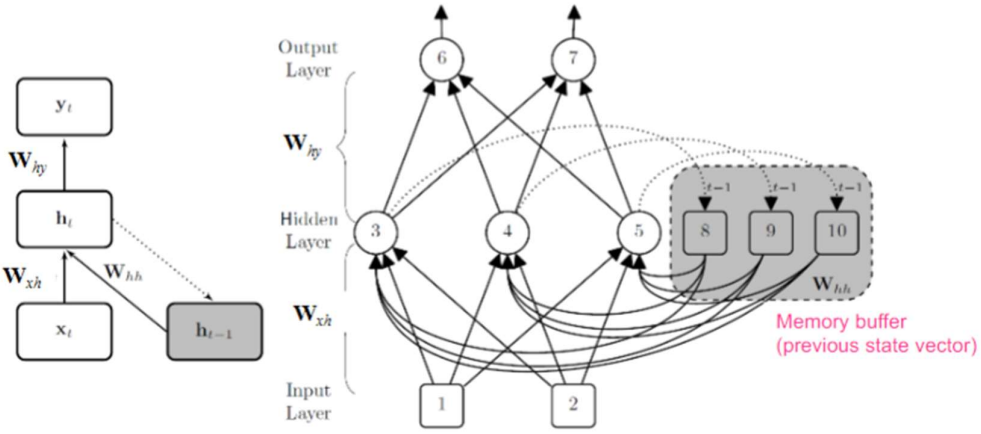
  It is also a deeper network with computational efficiency. It has 22 layers and 5 million parameters. It utilizes an efficient inception module with no FC layers.

Recurrent Neural Network

Recurrent Neural Networks are used for sequential data. Sequential data are text (natural language) data and the likes. Processing these types of data requires a model that can remember relevant information from the earlier in the sequence (needs memory). Recurrent Neural Networks can do exactly this. When processing the sequence, a RNN takes 1 element from the sequence at each point of time. RNN is called so because it contains feedback connections. Specifically, the output of a neuron at one time point is fed back into the same neuron at the next time point and this results in RNN having memory over past activations. RNN can hence capture long distance dependencies, and this is particularly useful for sequences. There are different types of architecture in RNN, these are:

Simple RNN

It is a feedforward Neural Network with 1 hidden layer. The hidden layer is extended with a memory buffer which stores the previous state of the hidden layer. At each time-step, the information from the memory is concatenated with the next input to each neuron. The architecture is as follows:



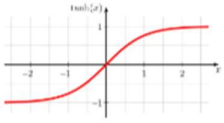
Here x_t is the input vector at time t, h_t is the hidden vector at time t, and y_t is the output vector at time t. There are a total of three weight matrices in this which are W_{xh} which represents the weights between the input vector and the hidden vector, W_{hy} which represents the weights between the hidden vector and the output vector, and finally the W_{hh} which are the weights between the memory buffer and the hidden layer.

Forward pass - calculating output

New state — Old state — Input vector

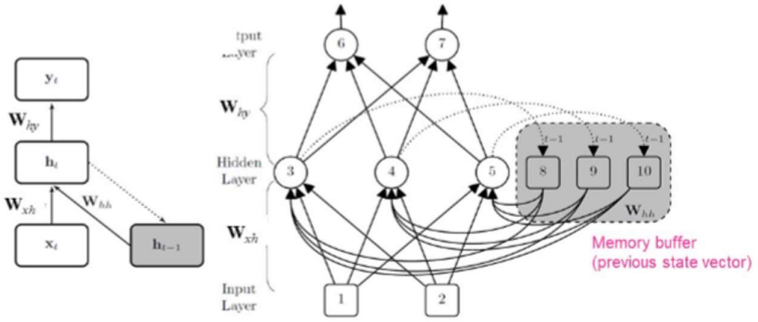
$$h_t = f_W(h_{t-1}, x_t)$$

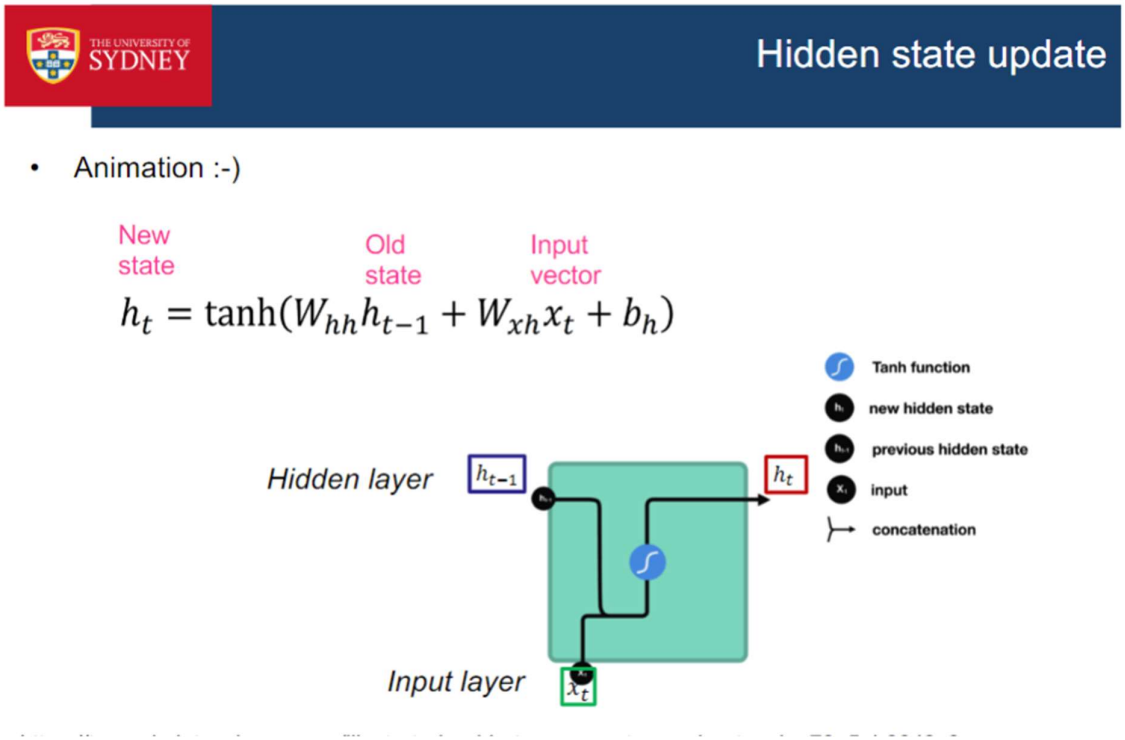
f_W – function with parameters W; tanh is typically used: output between 1 and -1



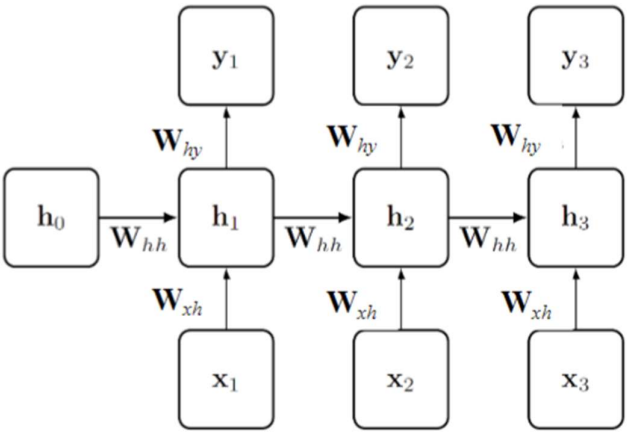
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$





The RNNs have cyclic connections. Meaning that the output of the hidden neuron at time (t-1) is fed back as input to the hidden neuron at time t. These cycles make it difficult to understand the information flow in RNN when many time steps are involved and hence, this is visualised it by unrolling the RNN. An example is as follows:



There are no cycles in the graph and the RNN is the same between the unfolded time steps. The weights do not change in this visualization as this is just a spread-out version of the cycle. h_0 is the initial state of the memory buffer when RNN is initialised. H represents the memory buffer with the numbers representing the cycle number. X representing the input vector and the Y representing the output vectors. W represents the weights, as described previously. As usual this Neural Network is also trained using the Backward Propagation Method. In this case, a variant called the Backpropagation through time. In this variant, the errors are backpropagated to all the parameters that contributed to it. After each character is applied, a weight update is calculated for each location of the network that was used and at the end of each epoch, the wright updates for the same locations are summed and the whole network is

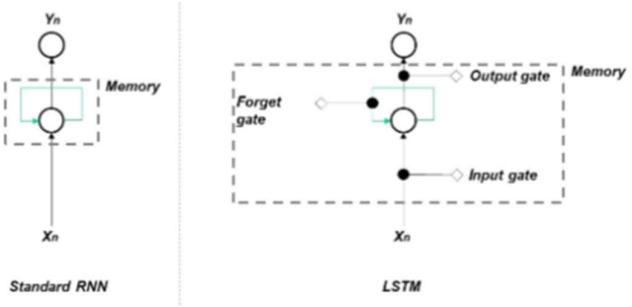
updated using the summed weight update. The backpropagation algorithm is applied multiple times until convergence (when all characters are classified correctly). After the training is completed, we can use the RNN to generate text (character after character) then feed a character to RNN and predict the next one and then feed the predicted character and predict the next one and so on.

RNN are flexible and can be deployed in different scenarios. It can be one to one (One input to one output), one to many (one input like an image and multiple output like a sequence of words), many to one (multiple input like a text document and single output like a positive or negative sentiment), many to many (multiple input like a sentence in English and multiple output like the translated sentence in Italian), etc.

There are shortcomings to RNN. RNN is susceptible to the vanishing gradient problem as during the backward pass, the error gradients will be multiplied by the weight matrix multiple times (once for each time step). This repeated multiplication might cause the gradient to vanish if the weights are too small. Another shortcoming is that even though RNN can learn long term dependencies, in practice this is limited.

LSTM Networks

Long Short-Term Memory networks are a type of RNN specifically designed to improve the ability of RNN to model long dependencies. The key idea is that the network can learn what to store and what to throw away and what to read from. The LSTM cell includes 3 gates: forget, input, and output.



The current time is represented as t. X is the input vector, Y is the output vector and H is the hidden vector. T-1 would be the previous short-term state. The current input vector and the previous short-term state are fed to 4 different FC layers. The 3 new layers have sigmoid activation function, and their outputs range from 0 to 1. Their outputs are fed to element-wise multiplication. If the output is 0, the gates are closed. If the output is 1, the gates are opened. That means that if the output is 0, it forgets things, if the output is 1, it memorizes things.

The Forget Gate layer decides what information from the long-term state should be thrown away or kept. It uses the information from the previous hidden state and information from the current input state. These are passed through the sigmoid function which outputs values between 0 and 1 for each number in the cell state. Values which are closer to 0 mean forget and closer to 1 mean remember.

The Input Gate Layer decides what new information should be stored in the cell state. A sigmoid layer (input gate layer) decides which value to update. A tanh layer (Simple RNN) creates a vector of new candidate state values that could be added to the state. These two are

combined using point wise multiplication. The old cell state is updated to produce a new cell state by combining the values of the forget and input layer.

The Output Gate Layer decides what exactly to output. The output will be based on the cell state, but it will be a filtered version. A sigmoid layer computes a value which determines the parts of the cell state to be output. The previously computed cell state goes through tanh function and is then multiplied by the sigmoid layer value so that output has parts which were pre-decided.

Other RNN architecture

- Gated Recurrent Unit (GRU)
- IRNN

WEEK 9

Clustering

Clustering is the process of dividing the data objects into groups (clusters) so that the objects in the same group are like each other within the cluster and dissimilar to the objects in other clusters. The similarity of clusters is computed using distance measures. Clustering is an unsupervised learning algorithm. Given a set of unlabelled examples and k desired number of clusters, the clustering algorithm clusters the examples into k clusters.

Supervised learning is when the class labels are given, and the goal is to build a classifier that can be used to predict the class of a new unlabelled example. Unsupervised learning is when there are no class labels, and the goal is to group similar examples together. Clustering is used as a stand-alone tool to group data or as a building block for other algorithms. A few applications for clustering are target marketing, customer loyalty, gene clustering, document clustering, clustering for understanding eating habits and dietary patterns of cohorts, colour compression, etc.

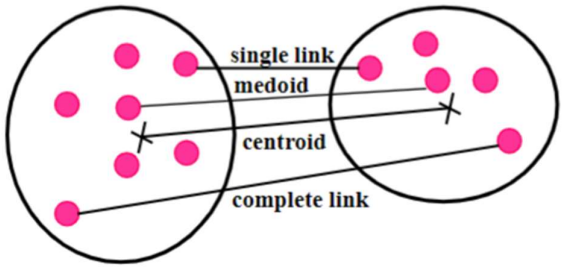
A good clustering will produce clusters with high cohesion (high similarity within clusters) and high separation (low similarity between the clusters). The similarity between 2 datapoints is measured using the distance measure. If the distance between the two datapoints is smaller, then the data points are similar. If the distance between the data points is high, then the data points are dissimilar. There are multiple types of distances that can be used such as Manhattan distance, Euclidean Distance, Cosine similarity, and many others.

Centroid and Medoid of clusters

Considering a cluster with N data points, the Centroid of the cluster would be the middle of the cluster. If the cluster is drawn out in the shape of the circle, then it would be the centre of the circle. It can be considered as the mean of data points. A medoid is the centrally located data point in the cluster. It can be considered as the median of the data points.

Distance between Clusters

- Centroid distance is the distance between 2 centroids of different clusters.
- Medoid distance is the distance between 2 medoids of different clusters.
- Single Link (MIN) is the smallest distance between elements from each cluster.
- Complete Link (MAX) is the largest distance between elements from each cluster.
- Average Link is the average distance between elements from each cluster.



Types of Clustering Algorithms

There are different types of clustering algorithms such as:

- Partitional
  - K-Means

    K-Means is a partitional clustering algorithm which is very popular and widely used. It requires the number of clusters "k" to be specified. It comprises of 3 main steps:
    - Choose k examples as the initial centroids of the clusters.
    - Form k clusters by assigning each example to the closest centroid.
    - At the end of each epoch recompute the centroid of the cluster, and continue until the stopping condition is satisfied (centroids do not change).

    The initial centroids are typically chosen randomly. The algorithm is sensitive to the initial centroids and different clusters will be produced. The closeness is measured by a distance measure and most of the convergence happens in the first few epochs. The complexity of the algorithm is given by O(n*k*i*d) where n is the number of points, k is the number of clusters, I is the number of iterations, and d is the number of attributes.

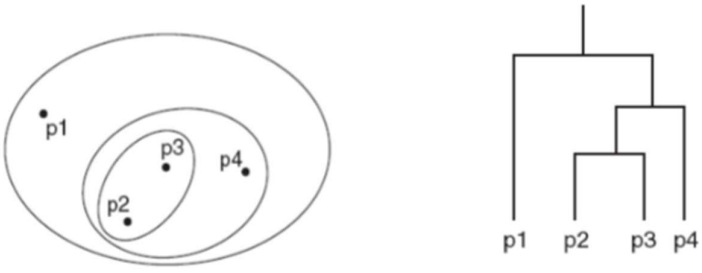
    There are certain methods of choosing good initial centroids. One is to run the K-means several time with different randomly selected initial centroids and evaluate each clustering using SSE. Select the clustering with the smallest Sum of Squared Error (SSE). Another method is called the K-means ++ which is a variation of K-means. It uses a new method for selecting the initial centroids whereas the rest is the same as standard K-means. K-means ++ selects centroids incrementally until k centroids have been selected and at every step each point has a probability to be selected as a centroid that is proportional to the square of its distance to its closest centroid.

    K-means can yield empty clusters as well. To solve this issue, one can choose different initial centroids usually the point farthest away from the current centroid, or use the K-means ++ approach, or choose a point from the cluster with the highest SSE. If there are several empty clusters, the methods can be repeated several times. Another issue of K-means is that when there are outliers, the resulting cluster centroids are less representative, and the SSE is higher. To solve this issue, one can just remove the outliers before clustering but sometime the outliers are important.

    Another variation of K-Means is Bisecting K-Means wherein to obtain k clusters, the data is split into 2 clusters and then one of them is chosen to split further and the cycle continues until k cluster are obtained. This makes the algorithm less sensitive to initialization problems.

    K-Means work well with clusters that are spherical, of equal density, equal size and are well spaced. It doesn't work well for natural clusters with

complex shape or with natural clusters with vastly different size. It does not work with natural clusters with different densities as well.
    - K-Medoids
- Model based clustering algorithms.
    - GMM
      GMM stands for Gaussian Mixture Model clustering which is a probabilistic clustering technique. It assumes that the data is generated by a mixture of normal distributions. In this algorithm we assume that the data is generated by a mixture of k Gaussian distributions and each distribution has 2 parameters, mean and standard deviation. One distribution corresponds to one cluster and starting from the initial values, the parameters are estimated iteratively. After each estimation, probability for each example is computed and then parameters are recomputed until they don't change. K-Means uses hard assignment whereas GMM uses probabilistic assignment. GMM can be seen as a generalization of K-Means. It is more flexible and it allows for elliptical clusters rather than circular for probabilistic assignment to each cluster rather than crisp.
- Hierarchical
  Hierarchical clustering produces a set of nested clusters which are organized as a hierarchical tree. It can be visualized as a dendrogram.



Nested clusters *Dendrogram*

  There are multiple advantages of hierarchical clustering, for example, there is no need to specify the number of clusters in advance or a desired number of clusters can be obtained by cutting the dendrogram at different levels. Hierarchical clustering typically uses Single Link, Complete Link, Average Link or Ward's method of measuring distances. There are 2 approaches for hierarchical clustering, and these are:
    - Agglomerative clustering
      Also called as the bottom-up clustering wherein the clusters merges iteratively. The algorithm starts with each item in its own cluster and iteratively merge until all items belong in one big cluster. The key operation in this type of clustering algorithm is computing the distance between 2 clusters. There are different versions of how the clusters are merged at each step.
    - Divisive clustering
      Also called the top-down clustering wherein the clusters split into two until all items are in their own clusters. This can be implemented based on computing the minimum spanning tree.

WEEK 10

- Density based.
    - DBSCAN

DBSCAN stands for Density Based Spatial Clustering of Applications with Noise. The clusters created are regions of high density, separated from one another by regions with low density. In contrast to K-Means, DBSCAN can find clusters with arbitrary and complex shape. The main idea of the algorithm was that in a cluster the density of the points around it should be higher than the threshold. We need to define the density and the neighbourhood of a point. The neighbourhood of a point is the area within a radius of the point. Density of a point is the number of points in the neighbourhood of a point including that point. And Density threshold MinPts is the minimum number of points in the EPS neighbourhood of a point. The density of a point depends on the neighbourhood Eps (Epsilon). If the epsilon is too big, the density will be less and if the Eps is too small, all the points will have density as 1.

There are 3 types of points in DBSCAN. These are core point, border point, and the noise point. The core point is the one which has at least MinPts points in its Eps neighbourhood. Core points are in the interior of a density-based cluster. Border point is not a core point but falls within the neighbourhood of a core point. A border point may fall in the neighbourhood of several core points. Finally, the noise point is neither a core point nor a border point.

The algorithm of DBSCAN is given as follows:

- Label points as core, border, or noise.
- Discard the noise points.
- Cluster the remaining points as follows:
    - Any 2 core points within the Eps of each other are put in the same cluster.
    - Any border point from the neighbourhood of a core point is put in the same cluster as the core point.
    - Ties need to be resolved when a border point belongs to the neighbourhood of more than 1 core points.

The Epsilon and MinPts need to be selected carefully as if the Eps is too big, then all the points will be core points, and if the Eps is too small then all the points will be noise points. The best method for choosing the Eps and the MinPts is by examining the distance from a point to its k nearest neighbour . The points for which the k-dist is less than Eps will be labelled as core points while the others would be labelled as noise or border points. This method requires the value of k.

DBSCAN cannot handle clusters with widely carrying density well and it produces different results depending on the parameters Eps and MinPts. The time complexity of DBSCAN is given by $O(n^2)$ and the space is given by $O(n)$.

- Grid Based

Grid based clustering is a density-based clustering algorithm which utilizes grids. The main idea is to break the data points into grid cells and then form clusters from the cells that are dense enough. The values of each attribute are split into intervals which are called grid cells. The common approach to do this is by splitting the values in equal width intervals. This leads the grid cells to have the same volume and the number of points will give the density of the cell. There are more sophisticated approaches as well such as breaking the values of an attribute in intervals such that there are equal data points in each grid, using clustering to determine intervals, break the initial values into large number of equal width intervals and then combine intervals with similar densities.

The grid cells will define the output of the clustering algorithm. The density of the cell is given by the amount of points/volume of the cell. Forming clusters from adjacent groups of dense cells is easy and just need the definition of adjacent.

- CLIQUE

  Clustering algorithms that use a few attributes from a pool of attributes is called subspace clustering algorithms. CLIQUE is called as the Clustering in Quest algorithm. It is a dimension growth-based clustering algorithm. The main idea of the algorithm is to find subspaces of high dimensionality where high density clusters exist. At each subspace, the data is partitioned into rectangular cells and the dense cells are identified based on a threshold. To efficiently find the dense cells at k-dim subspace, it uses the Apriori property from association rule mining and only considers the dense cells from the previous lower dimensional subspace (k-1).

  Apriori principle states that if a set of points forms a density-based cluster in k dimensional space, then the same set of points are part of the density-based cluster in all possible subsets of those 3 dimensions.

Evaluating Clustering Results

All clustering algorithms will find clusters even if the data is random and has no natural clusters. Hence, the results need to be evaluated to filter out the noise or catch any patterns. Clustering quality can be measured by 2 methods, the high similarity within the cluster using unsupervised measures, or comparing clustering results with correct cluster labels under supervised algorithm.

Unsupervised measures of evaluating clustering quality

- Measuring cohesion and separation.

  A good cluster produces clusters with high cohesion and high separation. Both are measured using distance measures. Cohesion is given by the formula: $cohesion(K_i) = \sum_{x \in K_i} dist(x, c_i)$ whereas the Separation is given by: $separation(K_i) = dist(c_i, c)$. Both can be measured as square distances as well in the form of SSE and BSE.

  Silhouette coefficient is a combination of cohesion and separation and is calculated for a point, cluster, and clustering. Silhouette coefficient for a point is given by: $s_i = (b_i - a_i)/\max(a_i, b_i)$. The values are always between -1 to 1, and the higher the number the better. The silhouette coefficient for a cluster is the average of the silhouette coefficient of all the data points in the cluster.
- Measuring the correlation between 2 similarity matrices.

  This measure fins if the similar items are in the same cluster or not. The main task of this measure is to evaluate the clustering quality using correlation. The correlation between the given similarity matrices is computed. There will be a high correlation for good clustering and low correlation for poor clustering.
- Visual inspection of sorted similarity matrix.

  In this the points are ordered based on their cluster and the similarity matrix is plotted using colours based on the similarity. Well defined blocks along the main diagonal indicate good clustering otherwise poor clustering.

Supervised measures of evaluating clustering quality

- Classification oriented measures
  Supervised measures will measure the difference between the true labels and the ones obtained by the clustering algorithm. If the difference is small, the algorithm is good. If the difference is big, the algorithm is bad.
  - Entropy & Purity
    High entropy means low purity and vice versa. To measure the homogeneity of the set of examples with respect to their class label. We use these measures to calculate the homogeneity of each cluster and based on this, the homogeneity of the overall clustering. Entropy is given by $entropy(K) = -\sum_i P_i . \log_2 P_i$. Entropy/Purity of the clustering is the sum of entropy/purity with respect to the dependent variable.
- Similarity oriented measures
  Calculates the correlation between the obtained cluster similarity index and the ideal cluster similarity index. The lesser the difference, the better the clustering. Usually there are binary matrices.
- Correlation between obtained and true similarity matrices.

Determining the good number of clusters is through using the elbow method wherein clustering algorithm is run for several k. Plot the SSE and look for distinct knee drop. Which would be the good number of clusters.

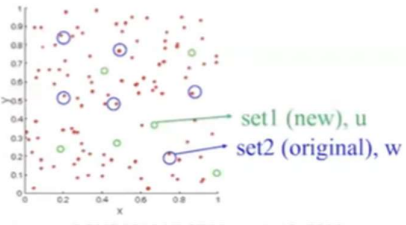
Clustering tendency

Given a dataset of m points, the main task is to estimate the clustering tendency (Is the data randomly distributed or not). This is usually done using the Hopkins statistic. The usage is given below:

- Generate 2 sets of p points (p<<m, heuristic: p=0.1*m):
  - Set 1: p random points
  - Set 2: p actual points from the dataset - sample randomly p points of the given m points
- For each point in both sets, find the closest point in the original dataset, i.e. find the nearest neighbor point
  - Let the distances to the nearest neighbor points are u_i for set 1 (newly generated points) and w_i for set 2 (original points)
- Hopkins statistic H:

$$H = \frac{\sum_{i=1}^{p} u_i}{\sum_{i=1}^{p} u_i + \sum_{i=1}^{p} w_i}$$

Note that there is a mistake in the textbook (Tan, p.589) - w_i is in the numerator, it should be u_i



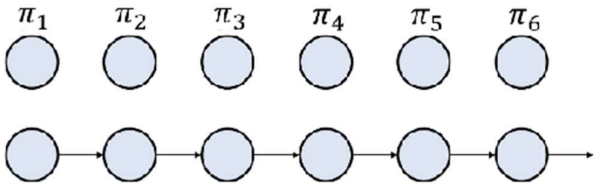
If the Hopkins value is near 1, that means the data is highly clustered, 0.5 means that set 1 and set 2 have roughly the same nearest neighbour distances, and near 0 means that data is neither clustered nor random and it is evenly spaced.

WEEK 11

Markov Model

Markov Chains was proposed by Andrey Markov. A Markov chain (Markov process) is a model describing a sequence of transitions from one state to another, in which the probability of each state depends on the previous state. The main assumption of Markov model is that the probability of any concrete state depends only on the previous state and not the older history.

$$P(\pi_i | \pi_1, \ldots, \pi_{i-1}) = P(\pi_i | \pi_{i-1})$$



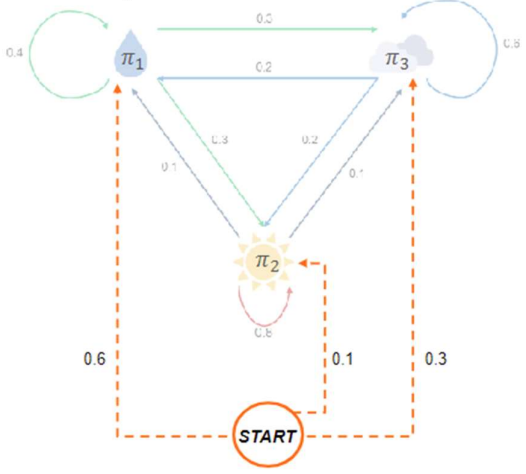
For Markov process questions, a transition probability matrix will be provided. The probabilities given in the table are all conditional probabilities. An example of this probability matrix is given below:

		π_t		
		Rainy	Cloudy	Sunny
π_{t-1}	Rainy	0.4	0.3	0.3
	Cloudy	0.2	0.6	0.2
	Sunny	0.1	0.1	0.8

Now if the previous 3 days were sunny, cloudy, and sunny and we must calculate the probability of it being sunny tomorrow, we calculate it using the formula:

$$P = \pi_t = \frac{s}{\pi_{t-1}} = Sunny\ to\ Sunny \rightarrow 0.8$$

We can represent this Markov chain in a graph wherein the nodes will represent the states, links represents the transition between states and weights will be the transitional probabilities. An example of this is given below using the above table:



We can use Markov Chains to predict sequences of states and not only single states. We can use the formula:

$$P(\pi_1, \ldots, \pi_k) = P(\pi_1) \prod_{i=1}^{k-1} P(\pi_{i+1} | \pi_i)$$

Hidden Markov Models

Markov Models are useful when probability of directly observable states is to be computed. Hidden Markov Models are used when are not observed directly or are hidden but can develop a judgement based on indirect observations. A Hidden Markov model is a probabilistic model that allow us to predict a sequence of hidden events from a set of observed events. The main components of HMM are as follows:

- A set of N possible hidden states: (π)
- A sequence of M observations: (X)
- A transition probability matrix: (A)
- An initial probability distribution: (A_0)
- An emission probability matrix representing conditional probabilities of observations of each state: (E)

Using this, the HMM model is given by:

$$\lambda = (\pi, A, A_0), X = x_1, x_2, x_3, \dots, x_M$$

There are 3 main questions of HMM, these are:

- Evaluation Problem
  Given the model and observational sequence, the question is: What is the probability of this sequence? There are multiple ways of solving this problem, such as:
  - Solution by Enumeration
    For the given observation sequence, compute the probability of each possible state sequence and sum these probabilities. This is a very inefficient task as the method is slow for tasks with long sequences and many states. Hence, we use the other solution.
  - Forward Algorithm
    Forward algorithm is a dynamic programming algorithm which stores a table of intermediate values (called forward probabilities) to make computation more efficient. The equation for forward probabilities is given by:

$$f_k(i) = e_k(x_i) \sum_j f_j(i-1)\, a_{jk}$$

This is the probability of the partial observation sequence up to time *i*

a_{jk}=P($\pi_i = k | \pi_{i-1} = j$)

a_{jk}: transition probability from previous state π_j to current state π_k

$e_k(x_i) = P(x_i | \pi_i = k)$

$e_k(x_i)$: emission probability of observation x_i given the current state k

    The probability of partial observation sequence can be easily computed, and it is also an efficient algorithm. The step for a forward algorithm is:
    - Initialization: Compute $f_k(1)$ for observation at time t=1.
    - Iteration: Compute $f_k(i) = e_k(x_i) \sum_j f_j(i-1)\, a_{jk}$
    - Termination
- Decoding Problem
  Given the model and observational sequence, the question is: What is the most likely sequence of hidden states? For this, we can use he Viterbi algorithm which is a dynamic programming algorithm which calculates the probability of the best path ending at each of the states. The Viterbi algorithm is efficient at each stage as it only needs to maintain the highest scoring path and not the list of all possible paths. The equations are given by:

$V_k(i) = e_k(x_i) \max_j V_j(i-1)\ a_{jk}$	Probability of most likely sequence of sates ending at state $\pi_i = k$
$a_{jk} = \mathrm{P}(\pi_i = k \mid \pi_{i-1} = j)$	a_{jk}: *transition probability from previous state π_j to current state π_k*
$e_k(x_i) = P(x_i \mid \pi_i = k)$	$e_k(x_i)$: *emission probability of observation x_i given the current state k*

The Viterbi score gives the probability of the best path ending with the final state and not the path. To determine the path, we use the back-pointer which is calculated during the Viterbi score. The pointer equation is:

$$Ptr_k(i) = \underset{j}{\operatorname{argmax}}\, V_j(i-1)a_{jk}$$

The Viterbi algorithm isas follows:

- Initialization: $V_k(1) = A_0(k)e_k(x_1)$
- Iteration:

  1) Viterbi score of state *k* at time *i*:

  $$V_k(i) = e_k(x_i) \max_j V_j(i-1)a_{jk}$$

  2) Back-pointer of state *k* at time *i*

  $$Ptr_k(i) = \underset{j}{\operatorname{argmax}}\, V_j(i-1)a_{jk}$$

- Termination and Trace back

  1) Determine the final state (at the last time step *m*):

  $$\pi_m = \underset{k}{\operatorname{argmax}}\, V_k(m)$$

  2) Trace-back the previous states:

  $$\pi_{i-1} = Ptr_{\pi_i}(i) \quad i = m, \ldots, 2$$

- Learning Problem

  Given the observational sequence, the question is: What is the model? For this question, we use the Expectation Maximization algorithm. The step for these is:
  - Initialize $\lambda = (\pi, A, A_0)$ to random values.
  - Compute probabilities for A, A_0 for all time steps, for all states π.
  - Re-estimate $\lambda = (\pi, A, A_0)$ based on the observations sequence X. Use the probabilities to predict the most likely observation sequence and compare with the actual observation sequence.
  - If $P(X|\lambda)$ increases, repeat from step 2, otherwise stop.

Applications of HMM

It is useful in speech processing, finance, bioinformatics, computer vision, information security, etc.

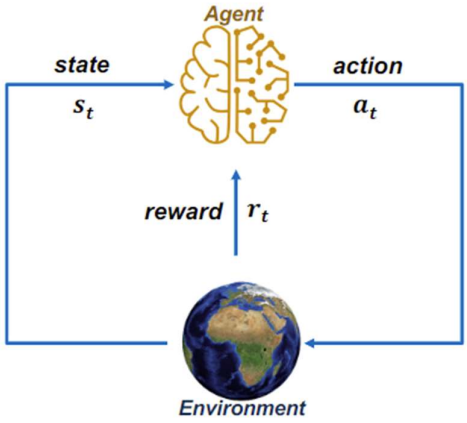
WEEK 12

Reinforcement Learning

Supervised Learning algorithms tries to make their output mimic the labels given in the training set. This is difficult to implement in sequential decision-making problems. This problem is solved by Reinforcement Learning. Reinforcement learning is a machine learning training method based on rewarding desired behaviours and/or punishing undesired ones. In general, a reinforcement learning agent can perceive and interpret its environment, take actions, and learn through trial and errors. An example of rewards would be:

- Making a humanoid robot walk.
  - Reward: For forward motion.

    - Punishment: For falling.
- Play games better than humans.
    - Reward: For increasing score.
    - Punishment: For decreasing score.



RL is a general-purpose framework for decision-making. The RL is for an agent with the capacity to act. Each action influences the agent's future state. Success is measured by a scalar reward signal. And the goal is to select actions to maximise future reward.

Components of Reinforcement Learning

- H_t → History of the sequence of observations, actions, rewards, etc. until time t.
- S_t → Information used to determine what happens next.

Markov's Decision Process

A Markov decision process is a discrete-time stochastic control process. It provides a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. It has the following:

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$$

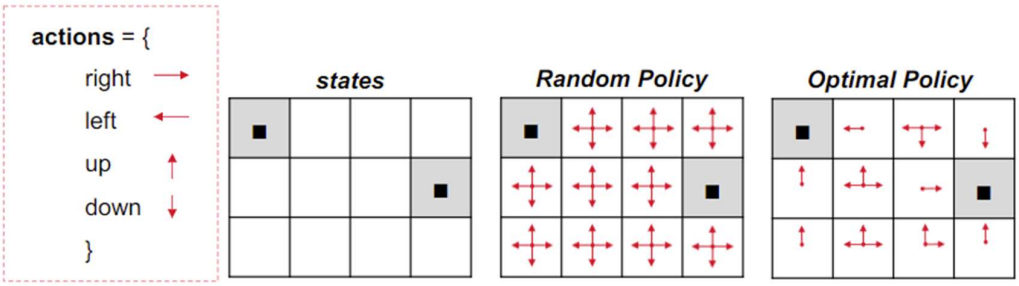
$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: **transition probability**
i.e. distribution over next state given (state, action) pair

γ : discount factor

There are 3 important components of a Reinforced Learning Agent in Markov's Decision Process:

- Policy: Agent's behaviour function.
- Value Function: How good is each state/action.
- Model: Agent's representation of the environment.

An example of Markov Decision Process is given below:



An optimal policy is one which will maximise the sum of rewards. This is given by the equation:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right] \quad \text{with} \quad s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

A few definitions to remember:

- State value function: How good is a state for a policy. The value function at state S, is the expected cumulative reward from following the policy from state S. This is given by the following equation:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- Action value function: How good is the state-action pair? The Q-value function at state S and action A, is the expected cumulative reward from acting A in state S and then following the policy. This is given by the equation:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Bellman equation

› *The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:*

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

› *Q* satisfies the following* ***Bellman equation****,* i.e., immediate reward plus discounted value of successor state

$$Q^*(s, a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

› ***Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of***

$$r + \gamma Q^*(s', a')$$

› *The optimal policy* π^* *corresponds to taking the best action in any state as specified by Q**

Q Learning

Q-Learning

$$Q^*(s,a) = \mathbb{E}\left\{R(s,a,s') + \max_{a'} \gamma Q^*(s',a')\right\}$$

Initialize $Q(s,a)$ arbitrarily.

Start with s.

Before taking action a, we calculate the current expected return as

$$Q(s,a)$$

After taking action a, and observing r and s', we calculate the target expected return as

$$R(s,a,s') + \max_{a'} \gamma Q(s',a')$$

$$\Delta Q(s,a) = R(s,a,s') + \max_{a'} \gamma Q(s',a') - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Delta Q(s,a)$$

Deep Q Learning

In deep Q learning we represent the state-value function by Q-network with weights W. It is an end-to-end learning of state-action values from raw pixels. The input state is a stack of raw pixels from the last 4 frames and the output layer are state-action values from all possible actions.

› Optimal Q-values obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left\{r + \gamma \max_{a'} Q(s',a') | s,a\right\}$$

› Treat right-hand size as a target and minimize MSE loss by SGD

$$I = \left(\underbrace{r + \gamma \max_{a'} Q_w(s',a')}_{\text{Target}} - Q_w(s,a)\right)^2$$

› Divergence issues using neural networks due to

- Correlations between samples
- Non-stationary targets

The algorithm for Deep Q learning is given below:

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t ϵ-greedy
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for

$$I = \left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right)^2$$

$$I = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left\{ \left(r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a) \right)^2 \right\}$$

› Q-learning is known to overestimate state-action values
  - The max operator uses the same values to select and evaluate an action

$$Q^*(s, a) = \mathbb{E}_{s'} \left\{ r + \gamma \max_{a'} Q(s', a') | s, a \right\}$$

› The upward bias can be removed by decoupling the selection from the evaluation
  - Current Q-network is used to select actions
  - Older Q-network is used to evaluate actions

$$I = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left\{ \left(r + \gamma Q_{\hat{w}_i}(s', \arg\max_{a'} Q_{w_i}(s', a')) - Q_{w_i}(s, a) \right)^2 \right\}$$

› Uniform experience replay samples transitions regardless of their significance

› Can weight experiences according to their significance

› Prioritized replay stores experiences in a priority queue according to the TD error

› Use stochastic sampling to increase sample diversity

$$|r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a)|$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

$$p_i = |\delta_i| + \epsilon$$

DL is a general-purpose framework for representation learning. The main purpose of this is that given an objective, learn the representation that is required to achieve objective directly from the raw inputs using minimal domain knowledge.

Applications of Reinforcement Learning

- AlphaGo and AlphaZero
  A general reinforcement learning algorithm that master's chess, shogi, and go, through self-play learning. AlphaGo was trained using the input values from human experts. It does an exhaustive search and then reduces breadth with the policy network and then reduces the depth with the value network. The reinforcement learning in AlphaGo is that the algorithm plays games against itself. The new policy network is trained to predict AlphaGo's moves, and the new value network is trained to predict the winner. Then the new policy/value network is used in the next iteration of AlphaGo Zero.

QUESTIONS

1. The regression line minimizes the sum of residuals.
   a) False.
2. If all residuals are 0, SST=SSR
   a) True.
3. If the value of correlation coefficient is negative, this indicates that the 2 variables are negatively correlated.
   a) True
4. The value of the correlation coefficient can be calculated given the value of R^2
   a) False
5. SSR represents an overall measure of the prediction error on the training set by using the regression line.
   a) False