MIT-ADSP
Conceptual Session - 2

Practical Data Science, Deep Learning & Recommendation Systems

Topics

- Decision Tree and Random Forest
  - Decision Tree
  - Bagging and Random forest
- Time Series
  - Basics of time series
  - Forecasting Models
- Neural Networks
  - Basics on NN and Activation functions
  - Forward and backward propagation
  - Gradient descent
  - Overfitting in NN
- CNN and GNN
  - Basics on CNN
  - Convolutional layer, Pooling layer, Padding and Stride
  - Transfer learning
  - Basics of GNN
- Recommendation Systems Part 1
  - Basics of Recommendation systems
  - Different recommendation systems (Popularity based and Collaborative filtering)
  - Singular Value Thresholding
- Recommendation Systems Part 2
  - Content based and Clustering based recommendation systems
  - Hybrid recommendation systems
  - Matrix estimation on time series data

Decision Tree and Random Forest

Back to first page

Contents

1. Decision Trees
2. Impurity Measures
3. Pruning
4. Cost complexity pruning
5. Ensemble Learning
6. Bagging
7. Random Forest
8. Hyperparameter tuning in Random forest
9. Advantages and disadvantages of random forest

Decision Tree

- A decision tree is one of the most popular and effective supervised learning techniques for classification problems, that works equally well with both categorical and continuous variables.
- It is a graphical representation of all the possible solutions to a decision that is based on a certain condition.
- In this algorithm, the training sample points are split into two or more sets based on the split condition over input variables.
- A simple example of a decision tree can be - A person has to decide on going out to play tennis or not by looking at the weather conditions.
    - If it's cloudy then he will go out to play.
    - If it's sunny, he will check the humidity level - if normal, he will go out to play.
    - If it's rainy, he further checks the wind speed - if that's weak, he will go out to play.

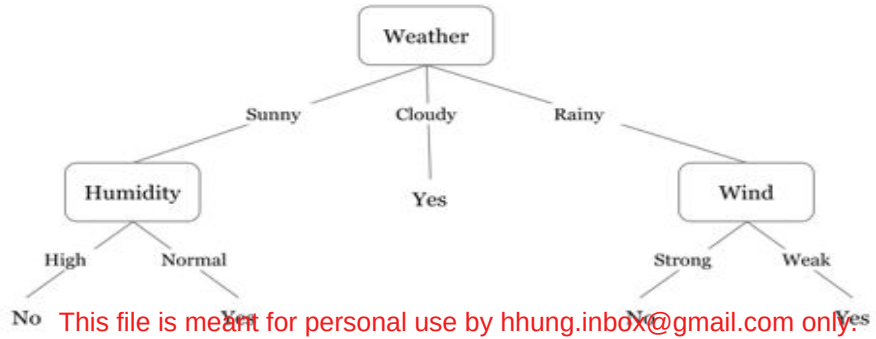


Image Source

Impurity Measures in Decision Trees

Impurity Measures: Decision trees recursively split features with respect to their target variable's purity. The algorithm is designed to optimize each split such that the purity will be maximized. Impurity can be measured in many ways such as Entropy, Information Gain, etc.

	GINI INDEX	ENTROPY	INFORMATION GAIN	VARIANCE
When to use	Classification Tree	Classification Tree	Classification Tree	Regression Tree
Formula	$G = 1 - \sum_{i=1}^{c}(p_i^2)$	$E = -\sum P(X).\log P(X)$	$IG(Y, X) = E(Y) - E(Y\|X)$	$V = \sum(x-\mu)^2/N$
Range	0 to 0.5 0 = most pure 0.5 = most impure	0 to 1 0 = most pure 1 = most impure	0 to 1 0 = less gain 1 = more gain	-
Characteristics	Easy to compute Non-additive	Computationally intensive Additive	Computationally intensive	The most common measure of dispersion

Pruning

- One of the problems with the decision tree is it gets easily overfit with the training sample and becomes too large and complex.
- A complex and large tree poorly generalizes to new sample data whereas a small tree fails to capture the information of the training sample data.
- Pruning may be defined as shortening the branches of the tree. It is the process of reducing the size of the tree by turning some branch node into a leaf node and removing the leaf node under the original branch.
- By removing branches we can reduce the complexity of tree which helps in reducing the overfitting of the tree.

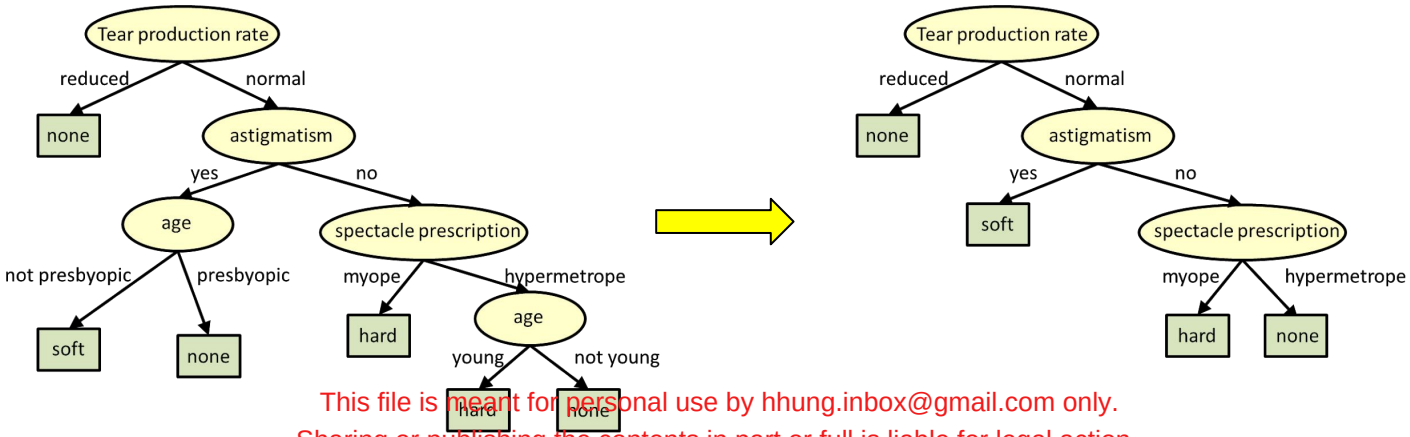


Image Source

Cost Complexity Pruning

- Cost Complexity Pruning is the most popular pruning technique for decision trees. It takes into account both the number of errors and the complexity of the tree.
- This technique is parametrized by the cost complexity parameter, ccp_alpha which reduces the complexity of the tree by controlling the number of leaf nodes, which eventually reduces overfitting. Greater values of ccp_alpha increase the number of nodes pruned.
- The complexity parameter is used to define the cost-complexity measure, $R\alpha(T)$ of a given tree T:

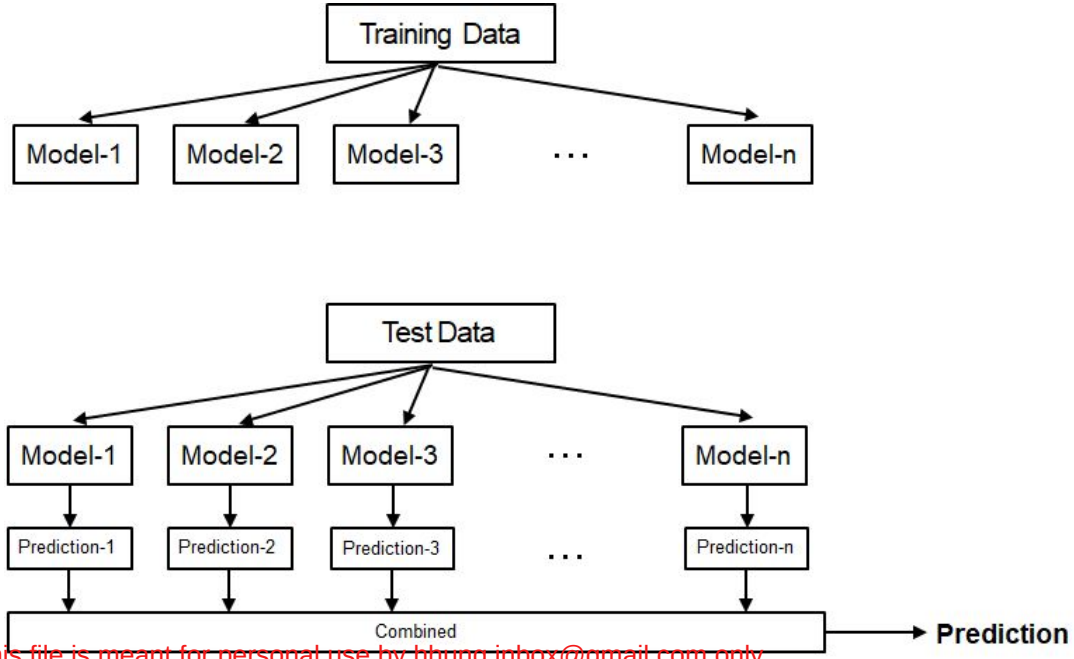
$$R\alpha(T)=R(T)+\alpha|T|$$

where |T| is the number of terminal nodes and R(T) is the total misclassification rate of the terminal nodes.

- Cost complexity pruning proceeds in the following stages:
  - A sequence of trees(T0, T1,...,Tk) for different values of alpha is built on the training data where T0 is the original tree before pruning and Tk is the root tree.
  - The tree Ti+1 is obtained by replacing one or more of the sub-trees in the predecessor tree Ti with suitable leaves.
  - The impurity of each pruned tree (T0, T1,...,Tk) is estimated and the best pruned tree is then selected based on the metric under consideration (using test data).

Ensemble Learning

- Ensemble Learning is a paradigm of machine learning methods for combining predictions from multiple separate models.
- The central motivation is rooted under the belief that a committee of experts working together can perform better than a single expert.



Bootstrap Aggregation (Bagging)

- Bagging is a technique of merging the outputs of various models to get a final result
- It reduces the chances of overfitting by training each model only with a randomly chosen subset of the training data. Training can be done in parallel.
- It essentially trains a large number of “strong” learners in parallel (each model is an overfit for that subset of the data)
- Then it combines (averaging or voting) these learners together to "smooth out" predictions.

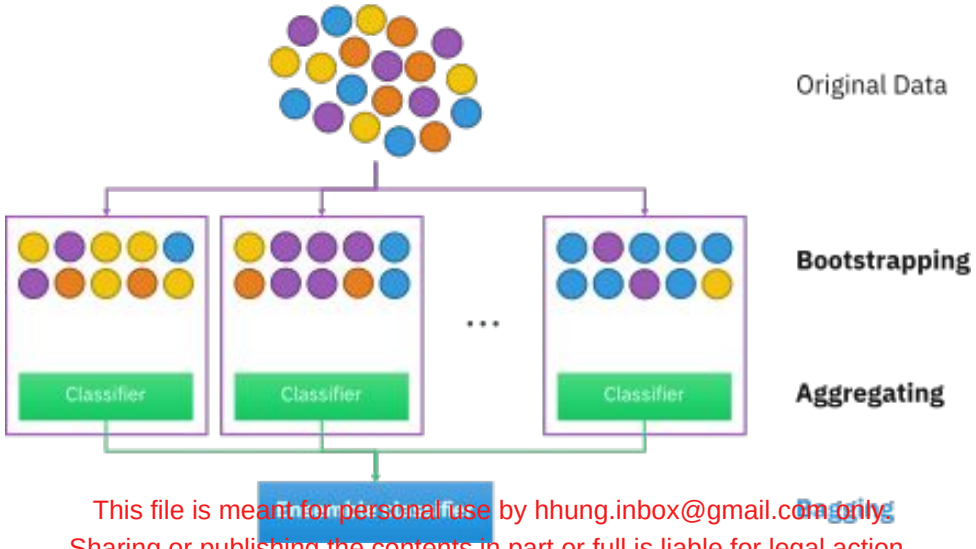


Image Source

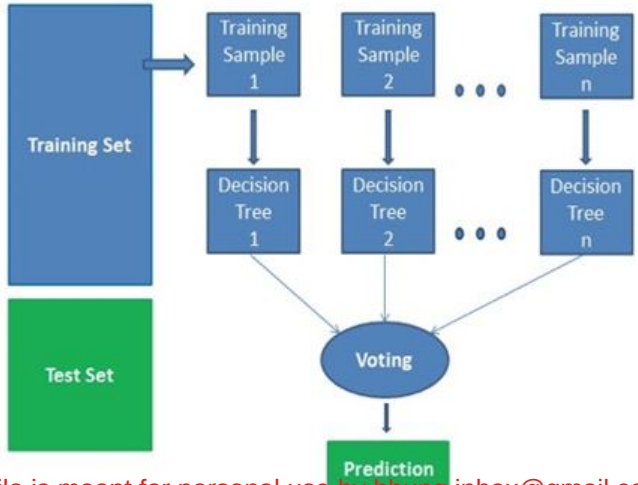
Random Forest

- Random Forest is a supervised machine learning algorithm which can be used for both classification and regression.
- It generates small decision trees using random subsamples of the dataset where the collection of the generated decision tree is defined as forest. Every individual tree is created using an attribute selection indicator such as entropy, information gain, etc.
- In classification, problem voting is done by each tree and the most voted class is considered the final result whereas in case of regression the average method is used to get the final outcome.
- Random Forest is used in various domains such as classification of images, feature selection and recommendation engines.

Steps involved in the Random Forest algorithm

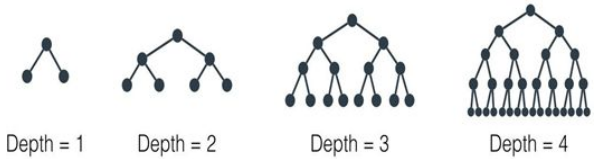
The following steps are involved in this algorithm:

1. Selection of a random subsample of a given dataset.
2. Using attribute selection indicators create a decision tree for each subsample and record the prediction outcome from each model.
3. Applying the voting/averaging method over predicted outcomes of individual models.
4. Considering the final results as the average value or most voted value.

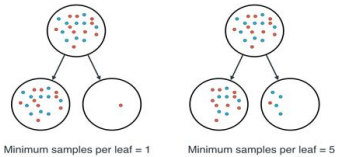


Hyperparameters in Random Forest

1. **Number of trees (n_estimators):**
   - It specifies the number of trees in the forest of the model.
   - The default value for this parameter is 10, which means that 10 different decision trees will be constructed in the random forest.



1. **Maximum Depth (max_depth):**
   - It specifies the maximum depth of the tree.
   - The default value is none which means the tree will expand until every leaf is pure



1. **The minimum number of samples per leaf (min_samples_leaf):**
   - It specifies the minimum number of samples required to be at a leaf node.
   - The default value is 1, which means that every leaf must have at least 1 sample that it classifies

1. **The minimum number of samples to split (min_samples_split):**
   - It specifies the minimum number of samples required to split an internal leaf node.
   - The default value for this parameter is 2, which means that an internal node must have at least two samples before it can be split to have a more specific classification.

Advantages and Disadvantages of Random Forest

Advantages:

- It can be used for both classification and regression problems.
- It is one of the most accurate algorithms because of the number of decision trees taking part in the process.
- It does not suffer from overfitting.
- It is used to select features of relatively more importance and helps in feature selection.

Disadvantages:

- The Random Forest algorithm is very slow compared to others because it calculates predictions for each decision tree for every sub sample and then votes on them to select the best one - which is time-consuming.
- It is difficult to explain the model as compared to a decision tree where you can easily make the decision following the path of the tree.

Time Series

Back to first page

Contents

- Time Series
- Decomposition of time series
- Stationarity time series
- Autoregressive models vs Moving Average models
- ARMA
- ARIMA

Time Series

- A time series is a sequence of measurements on the same variable collected over time.
- The measurements are made at regular time intervals.
- Examples include - predicting the number of churning customers, explaining seasonal patterns in sales, etc.

What is not a time series:

- Data collected on multiple items at the same point of time. for e.g. dow jones average on a single day
- When the time periods are not the same. for e.g. in a single time series both quarterly and yearly data

Features of a time series: The following are the features of a time series-

- Data is not independent
- Ordering is very important because there is dependency and changing the order will change the data structure

Yearly time series of per capita GDP of India ⟹

Decomposition of a time series

Great Learning
POWER AHEAD

Decomposition refers to the task of deconstructing the time series into several components like trend, seasonality etc.

There are two types of models used to decompose time series-

1. **Additive model:** It is useful when the seasonal variation is relatively constant over time.

$$\text{Observation} = \text{Trend} + \text{Seasonality} + \text{Error}$$

1. **Multiplicative model:** It is more realistic in nature and can be used when there are some variations over time

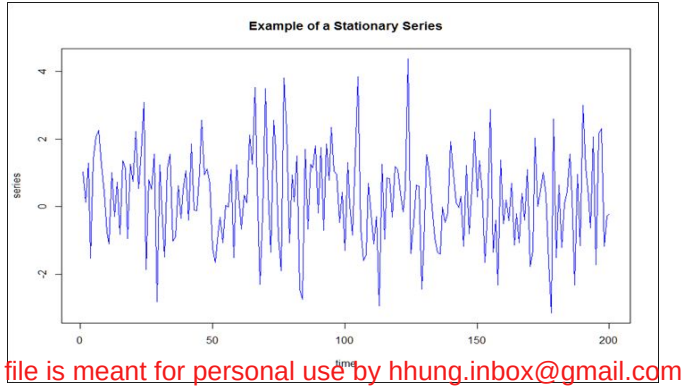
$$\text{Observation} = \text{Trend} * \text{Seasonality} * \text{Error}$$

Why do we need to decompose a time series:

- To compare the long-term movement of the series (Trend) vis-a-vis short-term movement (seasonality) to understand which has the higher influence
- To compare it amongst multiple sectors to avoid non-uniformity

Stationary time series

- In time series analysis, stationarity is a characteristic property of having **constant statistical measures** such as mean, variance, co-variance etc over a period of time.
- In other words, a time series is said to be stationary if the marginal distribution of y at a time p(yt) is the same at any other point in time. For time series analysis to be performed on a dataset, it should be stationary.
- It is also known as **white noise.**
- We can check the stationarity
  - By visualizing the rolling mean and standard deviation of the series
  - By using the Augmented Dickey-Fuller test



Autoregressive model vs Moving Average Model

Autoregressive models: Autoregressive models are based on the idea that the current value of the series, y_t, can be explained as a linear combination of p past values, $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$ of the same series.

An autoregressive model of order p, abbreviated AR(p), is of the form

$$AR(p): \quad y_t = f(y_{t-1}, y_{t-2}, \ldots y_{t-p})$$

The simplest AR process is AR(0), which has no dependence between the terms. In fact, AR(0) is essentially white noise. If y depends on more than one of its previous values then it is denoted by p parameters.

Moving Average models: While the autoregressive model considered the past values of the target variable for prediction, Moving average makes use of the white noise error terms. It can be represented as follows:

$$Y_t = \beta_0 + \varepsilon_t + \phi_1 \varepsilon_{t-1} + \phi_2 \varepsilon_{t-2} + \phi_3 \varepsilon_{t-3} + \ldots + \phi_q \varepsilon_{t-q}$$

Where, $\varepsilon t, \varepsilon t-1, \ldots, \varepsilon t-q$ are the white noise error terms.

ARMA Model

The Autoregressive Moving Average model is a combination of the autoregressive model and the moving average model which uses both the past values as well as the error terms to predict for the future time series.
If the process has terms from both an AR(p) and MA(q) process, then the process ARMA(p, q) can be expressed as

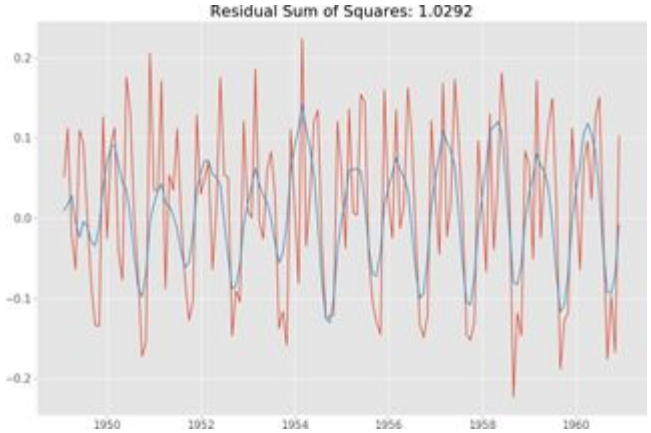
$$y_i = \phi_0 + \phi_1 y_{i-1} + \phi_2 y_{i-2} + \cdots + \phi_p y_{i-p} + \varepsilon_i + \theta_1 \varepsilon_{i-1} + \cdots + \theta_q \varepsilon_{i-q}$$

The ARMA model makes use of 2 parameters as given below:

p: Lag order or the number of past orders to be included in the model

q: The order of moving average

This plot shows the ARMA model fitted on the air passenger dataset. It gives the Residual Sum of Squares as 1.0292.



Source

ARIMA Model

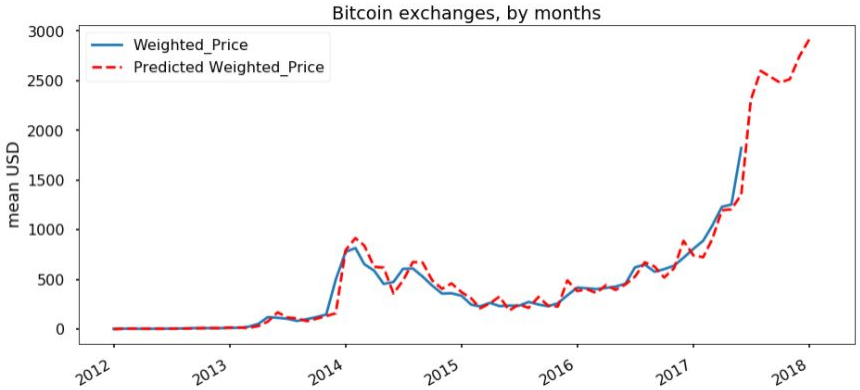
ARIMA stands for AutoRegressive Integrated Moving Average. It is a generalization of ARMA model and adds the notion of integration. The ARIMA model makes use of 3 parameters, which are given below:

p: Lag order or the number of past orders to be included in the model

d: The number of times differencing was applied in the original series in order to make the series stationary.

q: The order of moving average**.**

This plot shows the ARIMA model fitted on the bitcoin price prediction dataset.



Neural Networks

Back to first page

Content

- General Introduction to NN
- Reminder of nonlinear features
- Single unit and activation functions
- Multiple layers
- Architecture
- Cross Entropy Loss
- Gradient descent
- Basic Training Algorithms, SGD, Minibatch

Neural Networks

Artificial Neural Networks (ANNs) are inspired by biological neural networks and employ a collection of interconnected artificial neurons to extract patterns from given data.

They consist of three types of layers:

- **Input layer**
  - Represents dimensions of the input vector (one node for each dimension)
- **Hidden layer(s)**
  - Represents the intermediary nodes that divide the input space into regions with (soft) boundaries
  - Given enough hidden nodes, we can model an arbitrary input-output relation
  - It takes in a set of weighted input and produces output through an *activation function*
- **Output layer**
  - Represents the output of the neural network
  - Mostly, it doesn't have an activation function

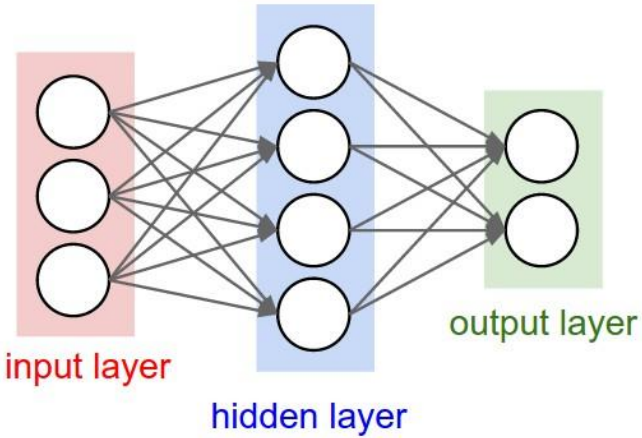
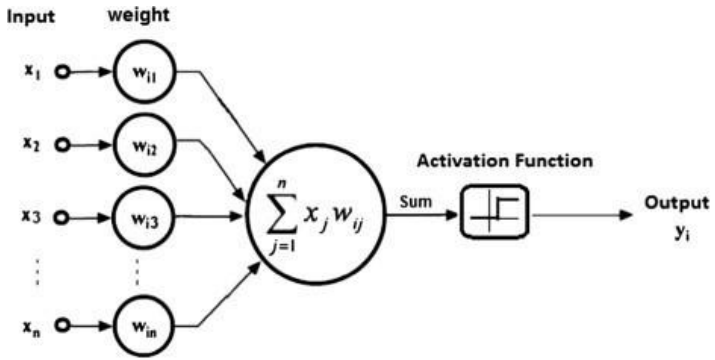


Image Source

Activation functions

- An artificial neural network works in three steps
  - First, it multiplies the input signals with corresponding weights
  - Second, it adds the weighted signals together
  - Third, it converts the result into another value using a mathematical transformation (activation function)
- For the third step, there are multiple mathematical functions available that can be used for the activation function.



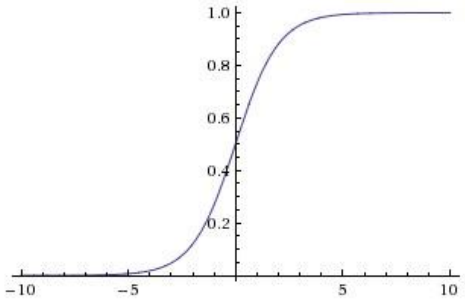
- The purpose of the activation function is to act like a switch for the neuron.
- The activation function is critical to the overall functioning of the neural network. Without it, the whole neural network will mathematically become equivalent to one single neuron!
- The activation function is one of the critical components that give neural networks the ability to deal with complex problems, by tackling the nonlinearity of the patterns in the data.

Source: Quora

Types of activation functions

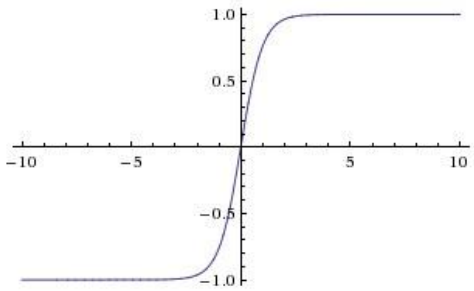
These are some activation functions that are generally used in neural networks:

1. The Sigmoid function
2. The Tanh function
3. The ReLU (Rectified Linear Unit) function



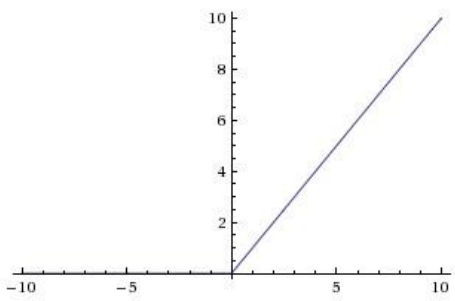
Sigmoid

- Range : 0 to 1
- Gives probabilities



Tanh

- Range : -1 to 1
- More steeper than sigmoid



ReLU

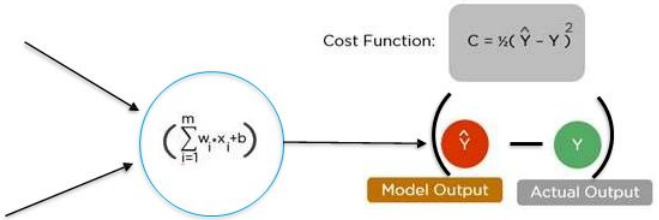
- Range : 0 to ∞
- Less computationally expensive

Image Source

Forward Propagation

- In the forward propagation step, the input data is propagated forward from the input layer through the hidden layer until it reaches the final/output layer where predictions are made.
- At every layer, data gets transformed in three steps in every neuron
  - Sum weighted input at every neuron by multiplying X by the hidden weight Wh and the bias
  - Apply the activation function on the sum
  - Pass the result to all the neurons in the next layer
- The last layer is the output layer, which may have a sigmoid function (for binary classification) or a softmax function (if the network is a multi-class classifier). The output layer gives the predictions of the neural network.

After getting the predictions, we use an optimization algorithm that helps us ***minimize*** error (cost) function **E(x)** which is simply a mathematical function dependent on the model's **learnable parameters** which are used in computing the target values (**Y**) from the set of *predictors* (**X**) used in the model.



Back Propagation

- Back propagation is the process of learning that the neural network employs to re-calibrate the weights at every layer and every node to minimize the error in the output layer
- During the first pass of forward propagation, the weights are random numbers
- The output of the first iteration is not always accurate. The difference between actual value / class and predicted value / class is the error
- All the nodes in all the preceding layers contribute to error and hence need to get their share of the error and correct their weights
- This process of allocating a proportion of the error (error gradient) to all the nodes in the previous layer is called back propagation
- The goal of back propagation is to adjust the weights in proportion to the error contribution and in an iterative process identify the optimal combination of weights
- At each layer, at each node, the gradient descent algorithm is applied to adjust the weights

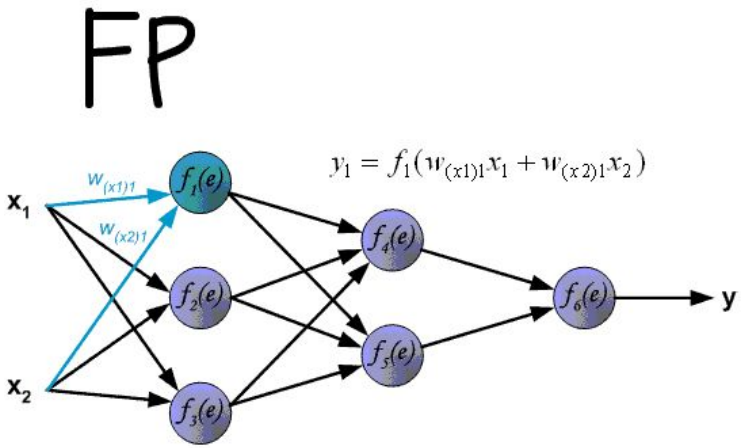
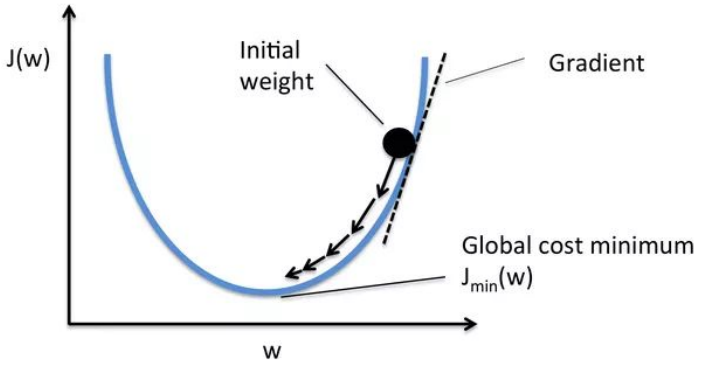


Image Source

Gradient Descent Algorithm

- The goal of optimization is to find a set of weights that minimizes the loss function
- Optimization functions usually calculate the **gradient** i.e. the partial derivative of loss function with respect to weights, and the weights are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minima of loss function
- The procedure of repeatedly evaluating the gradient and then performing a parameter update is called **Gradient Descent**



Learning Rate:

- It is a hyperparameter which determines the step size (the amount by which the weights are updated)
- We can try out different values of learning rate to improve the results

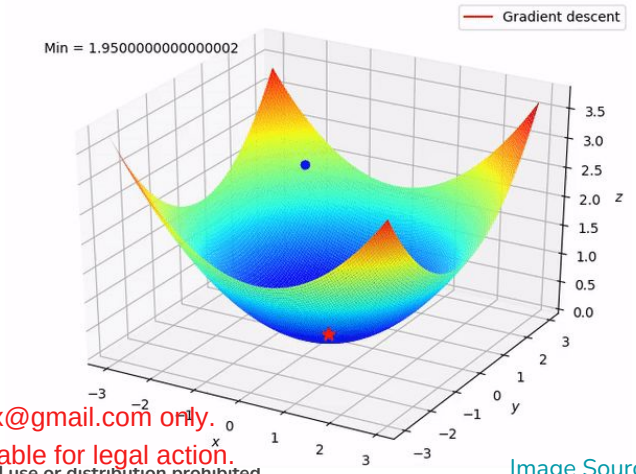


Image Source

Gradient Descent Variations

	Batch Gradient Descent	Stochastic Gradient descent	Mini Batch Gradient Descent
Working	It updates the parameter by calculating gradients of whole dataset	It updates the parameters by calculating gradients for each training example	It updates the parameters by calculating gradients for every mini batch of “n” training examples. It is a combination of batch and stochastic gradient descent
Advantages	It is computationally efficient and gives stable convergence	It is faster in learning than batch gradient descent and gives immediate performance insights	It is computationally efficient, fast to learn and gives stable convergence
Disadvantages	It learns very slowly and the chances of getting stuck in a local minima are high	It is computationally intensive and can give noisy gradients	It adds up one more hyperparameter to tune.

Overfitting in NN

Deep neural networks are prone to overfitting since they try to capture complex patterns in the data but they may also capture the noise in the data instead.

We can use the Ridge and Lasso regression techniques (that we have covered in previous modules) to reduce overfitting. There are also some other ways to reduce overfitting in a Deep Neural Network:

- **Batch Normalization:** Batch normalization is a technique for improving the performance and stability of neural networks. The idea is to normalize the inputs of each layer in such a way that they have a mean output activation of zero and standard deviation of one. This is analogous to how the inputs to networks are standardized.

  This approach leads to faster learning rates since normalization ensures there's no activation value that's too high or too low, as well as allowing each layer to learn independently of the others.

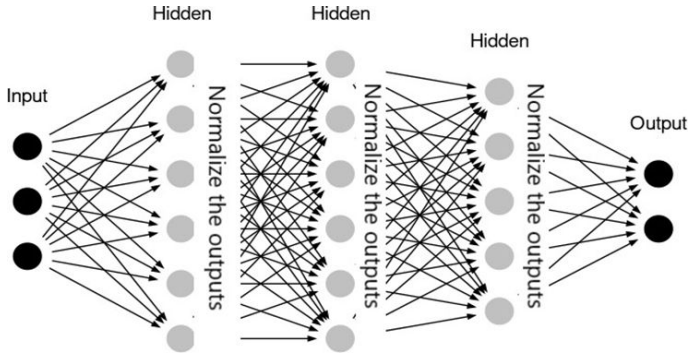
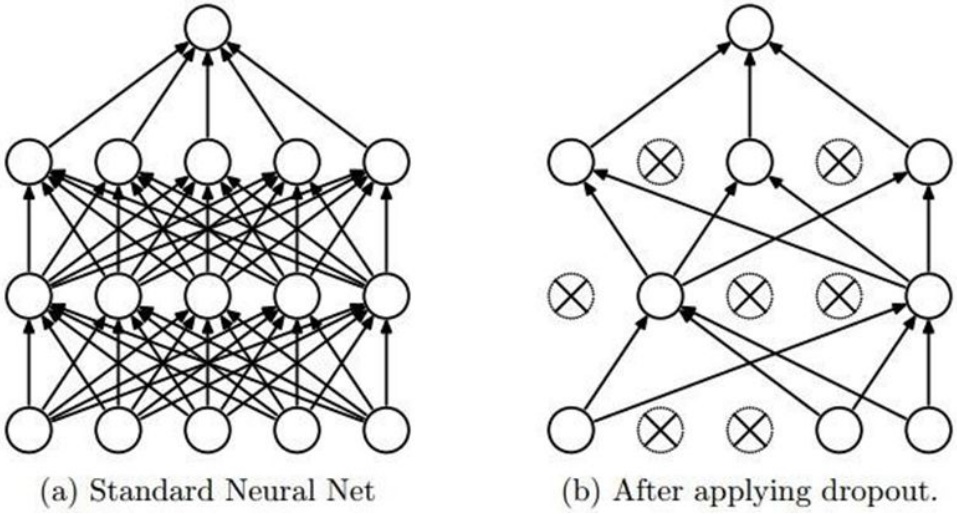


Image Source

Overfitting in NN

- **Dropout:** A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of the training data.

- In dropout, we randomly shut down some fraction of a layer's neurons at each training step by zeroing out the neuron values.
- The fraction of neurons to be zeroed out is known as the dropout rate, rd.
- The remaining neurons have their values multiplied by 1/(1-rd) so that the overall sum of the neuron values remains the same.



(a) Standard Neural Net (b) After applying dropout.

Source: (Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014)

CNN & GNN

Back to 1st page

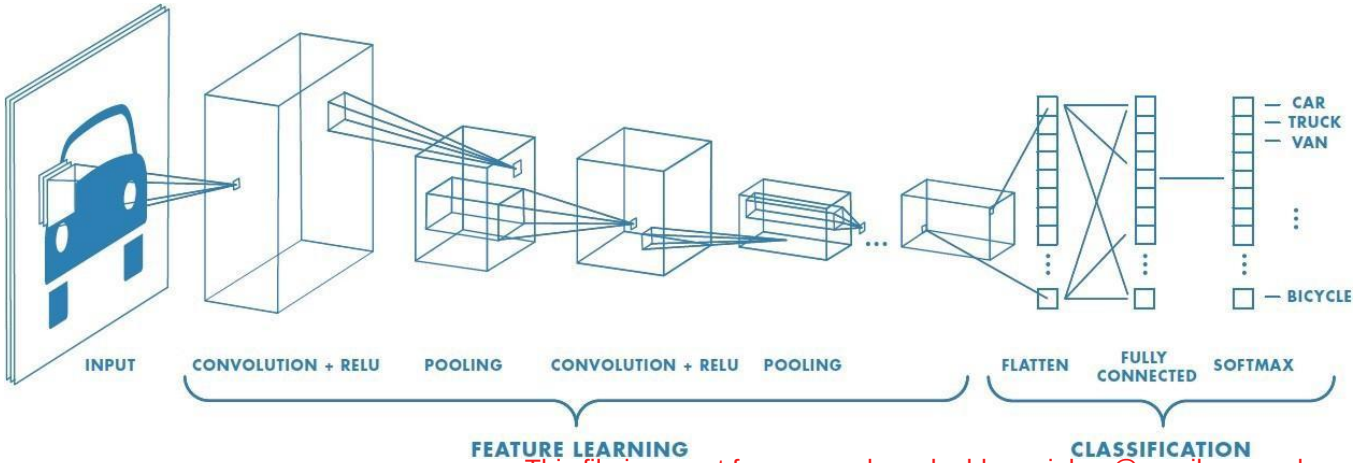
Contents

- Basics of Convolutional Neural Networks
- Locality, Translation invariance
- Filters/Convolutions
- Pooling layer
- Architecture of CNN
- Illustration of what CNN learns

Convolutional Neural Networks

CNNs are a special type of neural network designed to work with image data. CNNs use convolutional layers - hidden layers which perform convolution operations. They have some different characteristics to ANNs:

1. Unlike ANNs, CNNs capture the spatial structure of the image
2. CNNs follow the concept of parameter sharing i.e. one filter is applied over the whole image, because of which they are much more computationally efficient.



The first part in this architecture is the convolutional layer followed by the pooling layer and the second part is the fully connected layer. This whole architecture is called a convolutional neural network.

Image Source

The convolutional layer - Filter/Kernel

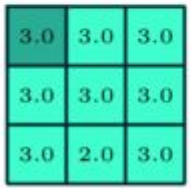
- A convolution operation uses a small array of numbers called a filter/kernel on the input image.
- Each filter is designed to identify a specific feature in the input space of the image, such as horizontal edges, vertical edges etc.
- A CNN is able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters.
- The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are important for getting a good prediction.

- This image shows how the convolution operation works in a CNN
- It uses a 3x3 filter on a 5x5 image
- The resulted feature is a 3x3 image which is the convolved feature

Image Source

Pooling layer in CNNs

- After a convolution operation, we usually perform pooling to reduce the dimensions of the feature map
- It enables us to reduce the number of parameters, which both reduces the training time and the overfitting
- Pooling layers downsample each feature map independently, reducing the height and width, but keeping the depth same.



There are two types of pooling - Max and Average

- Max pooling just takes the maximum value whereas average pooling takes the average value in the pooling window
- Contrary to the convolution operation, pooling has no parameters.
- In these gifs, we can see how both max and average pooling work

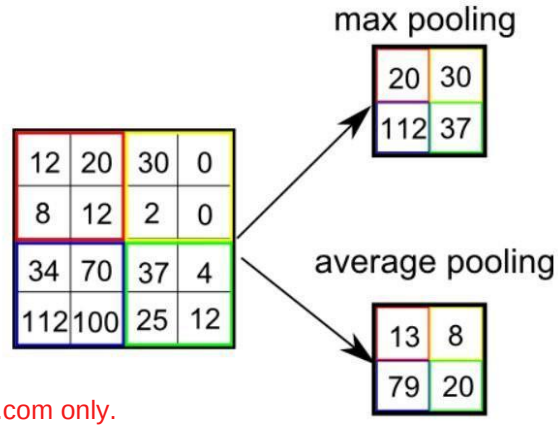
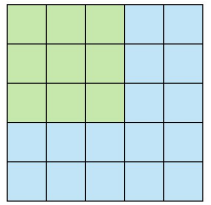


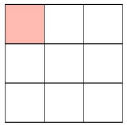
Image Source

Padding & Stride in CNNs

- **Stride** specifies how much we move the filter at each step. By default the value of the stride is 1 and is represented by the first figure
- We can also increase the value of stride if we want less overlap between the filters. It also makes the resulting feature map smaller since we are skipping over some locations.
- The second figure demonstrates the stride 2



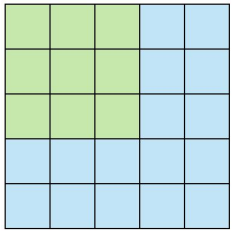
Stride 1



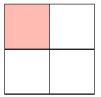
Feature Map

We see that after using stride, the size of the feature map is smaller than the input. If we want to maintain the same dimensions, we can use **padding** to surround the input with zeros.

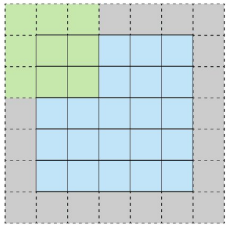
- The grey area around the input in the third figure is the padding.
- We either pad with zeros or the values on the edge, to match the dimensions of the feature map with the input.
- Padding is commonly used in CNNs to preserve the size of the feature maps, otherwise they would shrink at each layer, which is not desirable.



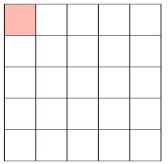
Stride 2



Feature Map



Stride 1 with Padding

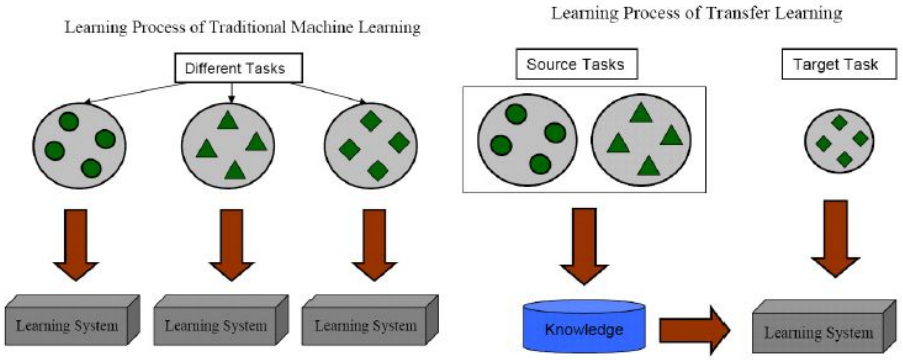


Feature Map

Image Source

Transfer Learning

- If you are using a specific NN architecture that has been trained before, you can use these pretrained parameters/weights instead of random initialization to solve your problem
- It can help you boost the performance of the NN.
- The pretrained models might have trained on large datasets like ImageNet, and taken a lot of time to learn those parameters/weights with optimized hyperparameters. This can save you a lot of time.
- If you have enough data, you can fine tune all the layers in your pretrained network but don't random initialize the parameters, leave the learned parameters as they are and learn from there.



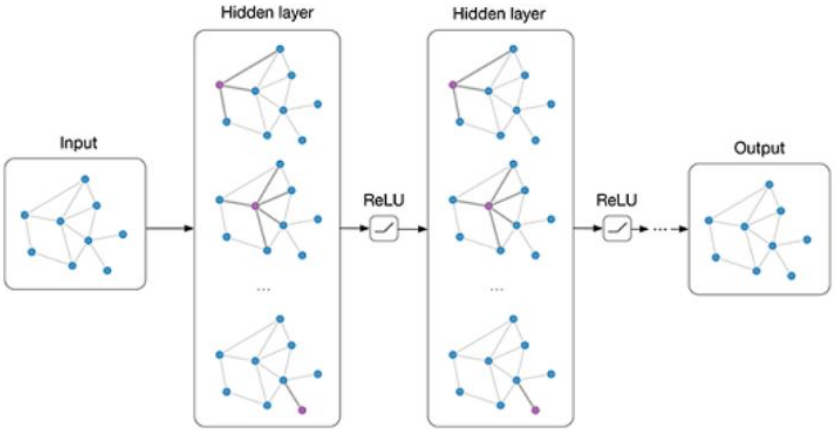
(a) Traditional Machine Learning (b) Transfer Learning

Graph Neural Networks

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure. It takes graphs as an input.

We need GNNs because CNNs do not perform well on graph data for the following reasons:

- Graphs do not have a fixed order of neighbours
- They do not have fixed node ordering
- They are arbitrary in size



GNNs are used when we want to:

- Classify the nodes
- Classify the graphs
- Predict the edges and other features

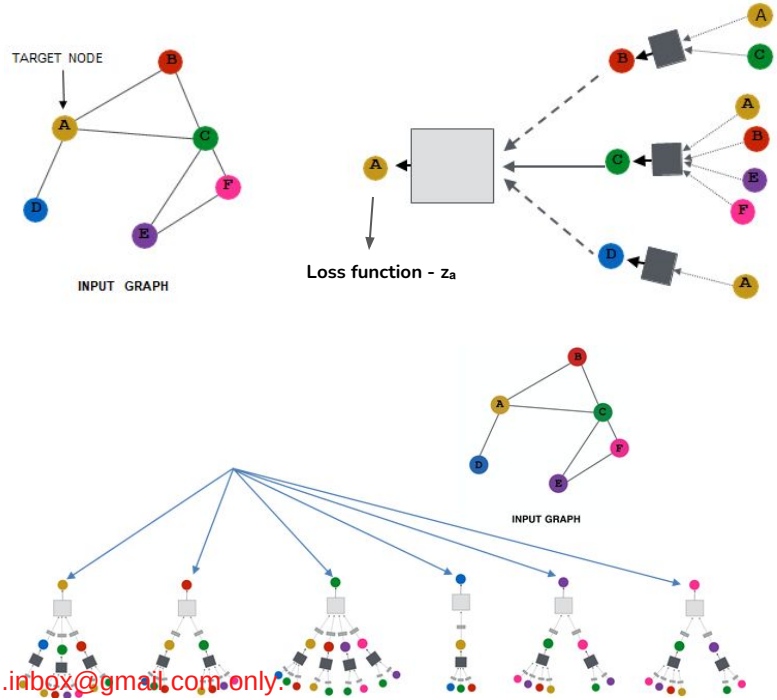
This image shows how GNNs work and predict the output.

Learning a GNN

Each node has a set of features defining it. In the case of social network graphs, this could be age, gender, country of residence, political leaning, and so on. Each edge may connect nodes together that have similar features. It shows some kind of interaction or relationship between them.

GNNs work in the following steps-

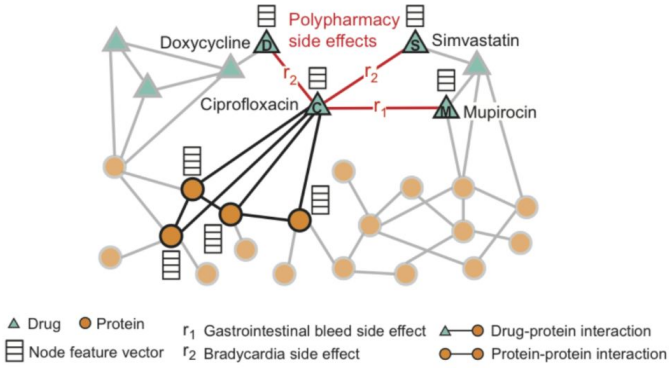
- We generate node embeddings based on local neighborhoods.
- The intuition behind neighborhood aggregation is that nodes aggregate information from their neighbors using neural networks.
- Then we specify a loss function on the output embedding for a target node (node A for example)
- We define a computational graph for each target node and each node defines a unique computational graph; we then train this on all the data points.
- Then we can generate embeddings for nodes that we have not even trained on.

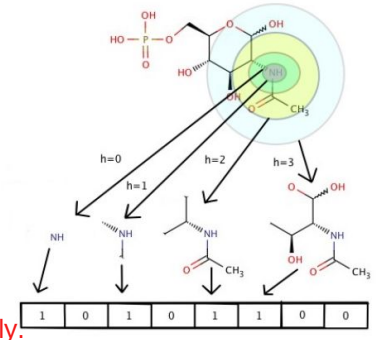


Applications of GNNs

These are some of the real-life applications of GNNs:

1. **Predicting side effects due to drug interactions -** We can predict what happens when two or more than two drugs interact with each other.
2. **Node Importance in Knowledge Graphs -** Using knowledge and product graphs to capture the relationships between product data and the critical context that humans have but machines lack. Such graphs enable machines to excel at downstream applications like product recommendations and question answering.
3. **Using GNNs in chemistry:** GNNs can be applied in both scenarios: learning about existing molecular structures as well as discovering new chemical structures. This has had a significant impact in computer-aided drug design.





Recommendation Systems - Part 1

Back to 1st page

Contents

- Introduction to the Recommendations, evaluation of specific metrics, sparsity of data, time varying data
- Examples of datasets
- Modelling process and simple solutions
- Improving solutions, Clustering
- Collaborative Filtering
- Singular Value Thresholding

Recommendation Systems

Recommender systems aim to predict users' interests and recommend product items that quite likely are interesting for them.

Why are they useful?

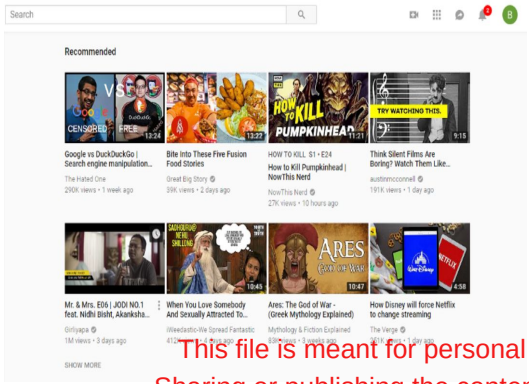
- Help users find item of their interest
- Help item provider deliver their items to the right user
- Identify products most relevant to the user
- Personalized content
- Help websites improve user engagement.

Some facts about recommendation systems

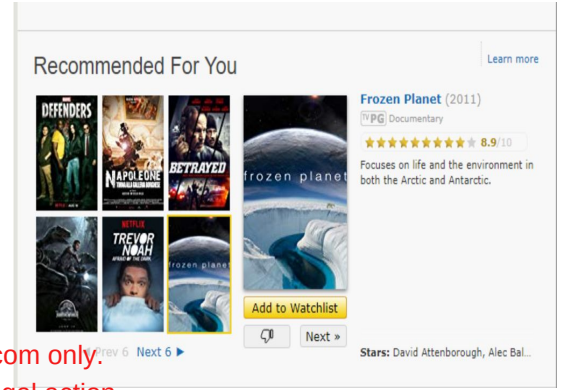
- Netflix – 2/3 movies watched are from recommendations
- Google News – 38% more click-through due to recommendations
- Amazon – 35% sales from recommendations

Source: (Celma & Lamere, ISMIR 2007)

Youtube recommendations



IMDB recommendations



Popularity-based recommendation systems

It is a type of recommendation system which suggests products/items based on the popularity or trend. These systems understand the products or movies which are in trend or are most popular among users and directly recommend those.

Advantages:

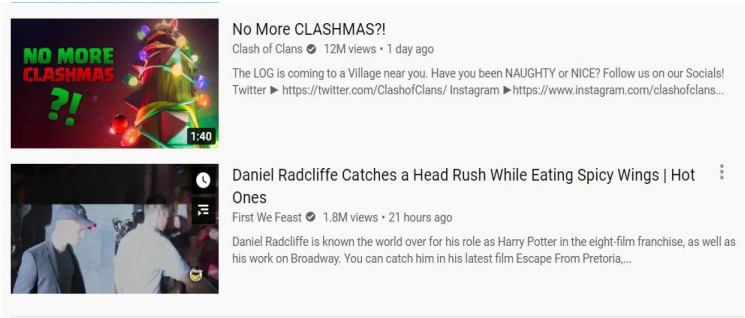
- No cold start problem
- No need for the user's historical data.

Disadvantages:

- Not personalized to users
- Only takes popularity into account

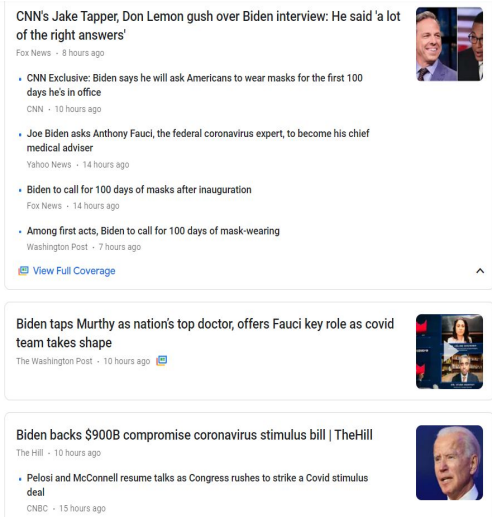
Examples:

1. Google News
2. YouTube trending videos



YouTube

Google News

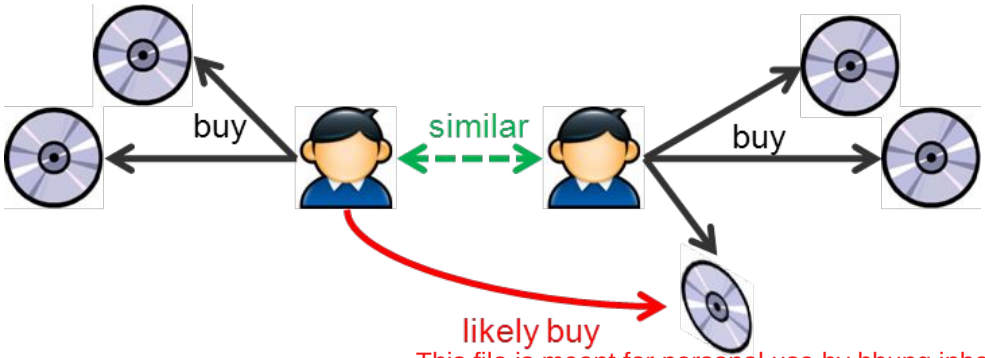


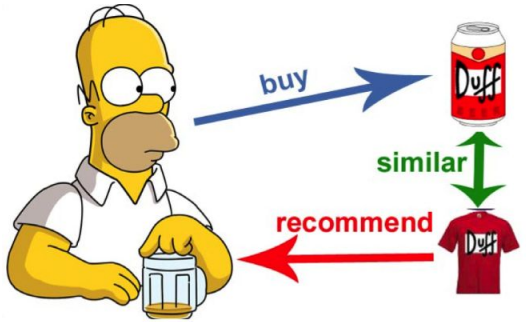
Similarity measures

	Cosine Similarity	Pearson Correlation	Euclidean Distance
Formula	$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$	$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$	$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$
Range	0 to 1	-1 to 1	>=0
When to use	When the data is sparse (many ratings are undefined)	When there is user-bias / different ratings scales of users in the data	When the data is not sparse and the magnitude of the feature values is significant

Collaborative Filtering

- The main idea behind collaborative filtering is that **if a user's likes/dislikes are similar to another user's likes/dislikes, then their tastes are similar.** We can use this to recommend a product that other similar users liked.
- It is based on the assumption that a person who liked something in the past will also like it in the future
- It is of two types-
  - **User-User collaborative filtering -** It is based on the search of similar users in terms of interactions with items.
  - **Item-Item collaborative filtering -** It is based on the search of similar items in terms of user-item interactions.

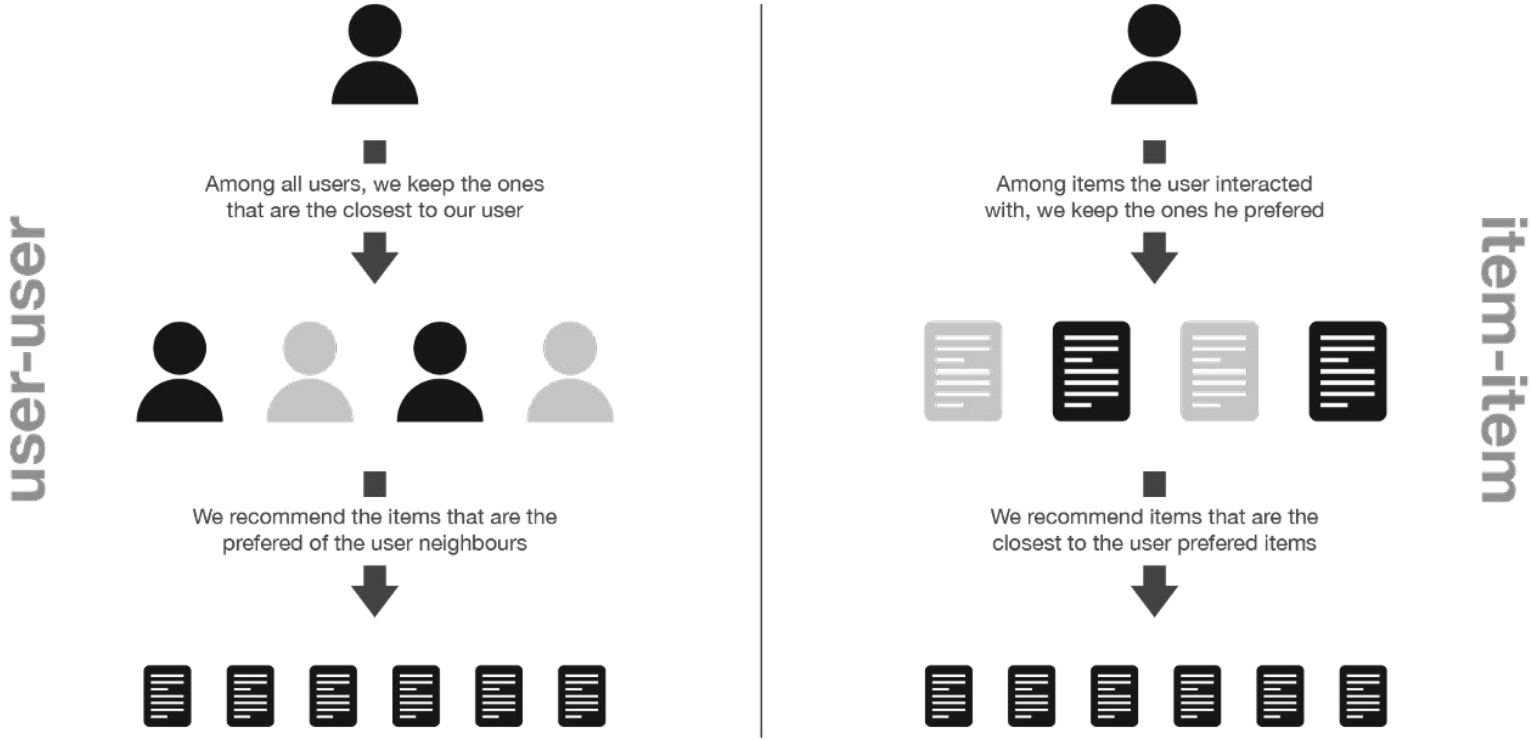




Source: djone

Source: medium

User-User vs Item-Item CF



Source: towardsdatascience

Advantages & Disadvantages of Collaborative Filtering

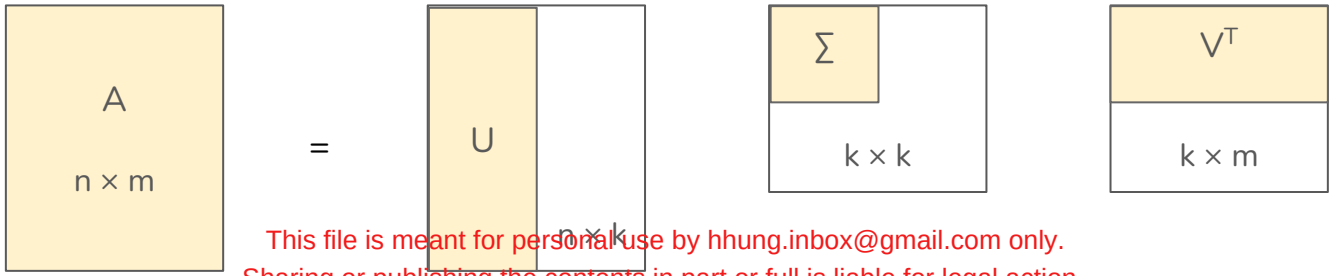
Advantages	Disadvantages
No domain knowledge needed	Cold start problem - can't handle new data
Personalized to users	Sparsity - Unable to compute ratings if there are a very less number of user preferences
Adaptive to the preferences changes over time	Scalability - Computationally expensive to scale

Matrix Factorization (SVD)

- Sparsity and scalability can be the biggest two challenges with CF methods
- Matrix Factorization decomposes the original sparse matrix to low-dimensional matrices with latent features and less sparsity.
- It gives us how much a user is aligned with a set of latent features, and how much a movie fits into this set of latent features
- It uses **Singular Value Decomposition** to factorize the matrix. For a user-movies n × m matrix, it is given by

$$A = U \sum V^T$$

Where U is an n × k user-latent feature matrix, V is an m × k movie-latent feature matrix. ∑ is an k × k diagonal matrix containing the singular values of original matrix, simply representing how important a specific feature is to predict user preference.



Recommendation Systems - Part 2

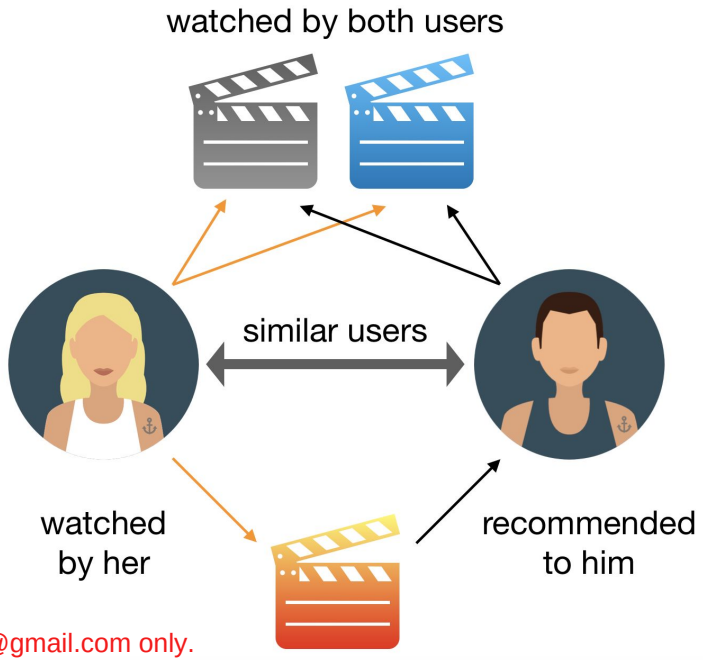
Back to 1st page

Contents

- Matrix Estimation with content based
- Clustering based recommendation system
- Hybrid recommendation systems
- Matrix Estimation over time

Content-based recommendation systems

- Content based recommenders work solely with the past interactions of a given user and do not take other users into consideration.
- The main idea behind content-based RS is that **If a user likes an item then he/she will also like a "similar" item**
- Uses information of all products and preferences of just that one user



Advantages

- Content-based recommender systems don't require a lot of user data.
- Does not suffer from cold start
- Less expensive to build and maintain

Challenges

- Unavailability of features which explain Items and user preferences
- Recommendations will likely be direct substitutes, and not complements, of the item the user interacted with.
- Less dynamic

Source: towardsdatascience

Clustering-based recommendation systems

We can also apply unsupervised methods (clustering) to build recommendation systems. Each cluster would be assigned to typical preferences, based on preferences of customers who belong to the cluster. Customers within each cluster would receive recommendations computed at the cluster level.

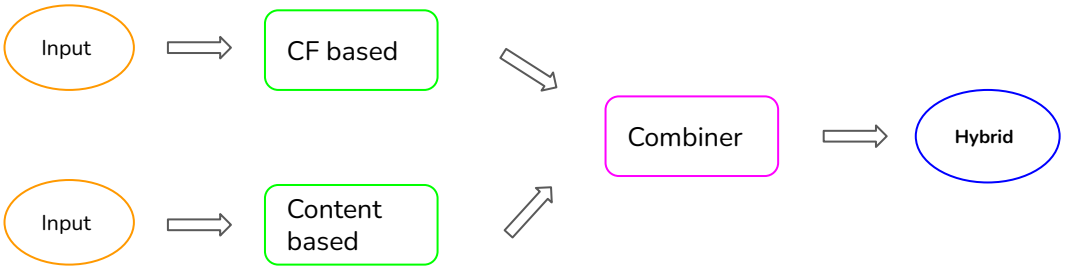
It works in the followings steps:

1. Compute similarity between each pair of users
2. Obtain representation of each user in low dimensional space
3. Perform the k-means clustering algorithm and find the k number of clusters

This image shows the clusters of users who have similar preferences so that they will receive the same product recommendations.

Hybrid recommendation systems

- Hybrid recommender systems are a special type of recommendation system that combine different methods, generally content and collaborative filtering methods.
- Hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa)
- These methods can also be used to overcome some of the following problems in recommender systems
  - Avoid the problem of recommending only similar products
  - Overcome the problem of false nearest neighbors
  - More personalized recommendations and specific to the user needs
  - Computationally faster system than the traditional recommendation system algorithms



Matrix estimation on time series data

- Matrix estimation can be used to fill missing values and forecast the time series data.
- Let's say you are given the following time series: X(1), X(2), ?, ?, X(5).....X(L)..... X(T). This matrix can be represented in the form of matrix as given below:
- Take some value L, say 3 and take first L values and put them as a column in a matrix. Keep repeating this for T values. After that, We can apply different matrix estimation methods to fill in the missing values and then forecast next L values in the dataset(**X(T+1), X(T+2), X(T+3)**).

X(1)	?	.	.	.	.	.	.	X(T+1)
X(2)	X(5)	.	.	.	.	.	.	X(T+2)
?		.	.	.	.	.	X(T)	X(T+3)

- The good thing about this process is, unlike the traditional forecasting methods which assume stationarity, we don't have any assumptions here and can apply this method for forecasting.

Great Learning

Happy Learning !