Notes on Data Science and Machine Learning

Raghav G. Jha [1]

Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, USA
Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5

Abstract

These notes cover some basic ideas and theory behind the field of machine learning. These notes are by no means even close to being exhaustive and simply meant for quick reference and interview preparation. Partly inspired by the amazing 100-page textbook.[2]

Contents

[1] raghav.govind.jha@gmail.com
[2] The Hundred-Page Machine Learning Book. by Andriy Burkov, Quebec City, Canada, 2019

Section 1

Introduction

> "I have no data yet. It is a capital mistake to theorise before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts." – Sherlock Holmes, A Scandal in Bohemia

It is no exagerration to say that all aspects of human life is related to data. It is then an interesting question to ask: how much the machine can learn and think like humans using that data. These questions all belong to the field of artificial intelligence (AI).

Machine learning (ML) originally emerged as a sub-discipline of AI research which tries to accomplish task using computers without explicitly programming them. A sub-field of ML is deep learning, whose goal is to mimic neural networks in our brain and enable computer to function similarly.

What we buy, what we buy online and many others are all data-driven and uses machine learning (ML) methods. What movies are recommended to you on Netflix or what Amazon recommends you to buy in bundle are all determined by various algorithms of ML. The pricing of airline tickets and the web history or what links you visit all determines what you are being shown online and how much you pay.

These notes are a brief introduction to the fundamentals and methods used in machine learning. Usually, ML is classified into three parts: Supervised, Unsupervised, Reinforcement learning. The canvas of ML is shown in Fig. 1.

Machine Learning is a subfield of Data Science that deals with using existing data to help systems automatically learn new skills to perform different tasks without having rules to be explicitly programmed. Deep Learning, on the other hand, is a field in Machine Learning that deals with building Machine Learning models using algorithms that try to imitate the process of how the human brain learns from the information in a system for it to attain new capabilities. In Deep Learning, we make heavy use of deeply connected neural networks with many layers.

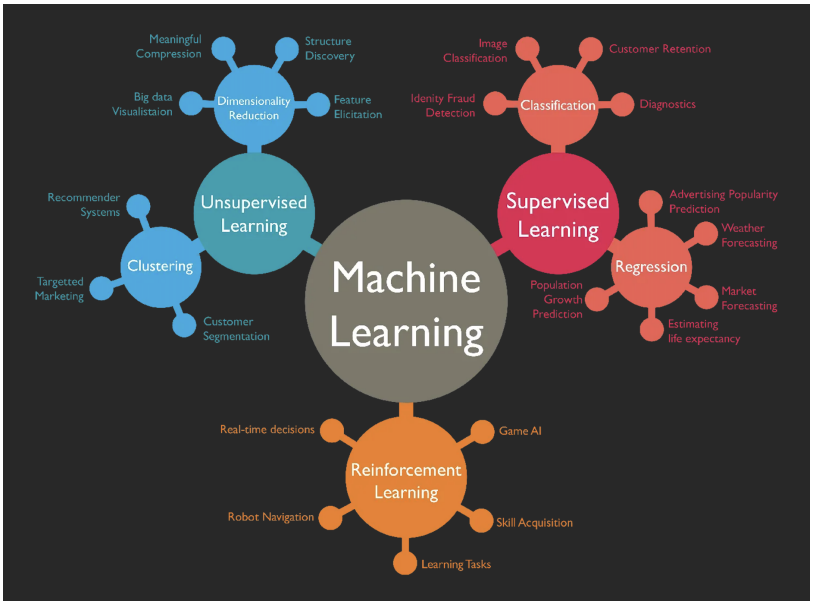


Figure 1. A cartoon showing the landscape of machine learning.

Section 2

Some famous ML algorithms

Some popular ML algorithms are:

- Linear regression: This gives a relationship between input x and an output variable y, also referred to as independent and dependent variables. Consider an example where you are required to arrange a few plastic boxes of different sizes on separate shelves based on their corresponding weights. The task is to be completed without manually weighing the boxes. Instead, you need to guess the weight just by observing the height, dimensions, and sizes of the box. In short, the entire task is driven based on visual analysis. Thus, you have to use a combination of visible variables to make the final arrangement on the shelves. Linear regression in machine learning is of a similar kind, where the relationship between independent and dependent variables is established by fitting them to a regression line. This line has a mathematical representation given by the linear equation $y = wx + c$, where y represents the dependent variable, w = slope, x = independent variable, and b = intercept. We need to find the best fit by minimizing the errors using Ordinary Least Squares (OLS) as:
$$\text{Min.} \sum_{i=1}^{N} (y_i - w_i x_i) \tag{2.1}$$
- Logistic regression: The dependent variable is of binary type (dichotomous) in logistic regression. This type of regression analysis describes data and explains the relationship between

one dichotomous variable and one or more independent variables. Logistic regression is used in predictive analysis where pertinent data predict an event probability to a logit function. Thus, it is also called logit regression. Mathematically, logistic regression is represented by the equation:

$$y = \exp(b_0 + b_1 x)/1 + \exp(b_0 + b_1 x)) \tag{2.2}$$

where x = input value, y = predicted output, b_0 = bias or intercept term, b_1 = coefficient for input x. Logistic regression could be used to predict whether a particular team will win (1) a tournament or not (0), or whether a lockdown will be imposed (1) due to rising COVID-19 cases or not (0) or whether a tumor is benign or not. Thus, the binary outcomes of logistic regression facilitate faster decision-making as you only need to pick one out of the two alternatives.

- Decision trees: With a decision tree, you can visualize the map of potential results for a series of decisions. It enables companies to compare possible outcomes and then take a straightforward decision based on parameters such as advantages and probabilities that are beneficial to them. Decision tree algorithms can potentially anticipate the best option based on a mathematical construct and also come in handy while brainstorming over a specific decision. The tree starts with a root node (decision node) and then branches into sub-nodes representing potential outcomes. Each outcome can further create child nodes that can open up other possibilities. The algorithm generates a tree-like structure that is used for classification problems. For example, consider the decision tree below that helps finalize a weekend plan based on the weather forecast.

  Gradient Boosting is similar to AdaBoost (Adaptive Boost) in that they both use an ensemble of decision trees to predict a target label. However, unlike AdaBoost, the Gradient Boost trees have a depth larger than 1. In practice, you'll typically see Gradient Boost being used with a maximum number of leaves of between 8 and 32. We know that `error = bias + variance`. Boosting is based on weak learners i.e., high bias and low variance. In terms of decision trees, *weak learners are shallow trees*, sometimes even as small as decision stumps (trees with two leaves). Boosting reduces error mainly by reducing bias (and also to some extent variance, by aggregating the output from many models). On the other hand, Random Forest uses as you said fully grown decision trees (low bias, high variance). It tackles the error reduction task in the opposite way: by reducing variance. The trees are made uncorrelated to maximize the decrease in variance, but the algorithm cannot reduce bias (which is slightly higher than the bias of an individual tree in the forest). Hence the need for large, unpruned trees, so that the bias is initially as low as possible. Please note that unlike Boosting (which is sequential), RF grows trees in parallel.

  In addition, the random forest approach is a bagging method where deep trees, fitted on bootstrap samples, are combined to produce an output with lower variance.

  Random Forest is an ensemble learning method that creates multiple decision trees and combines their predictions to make a final prediction. In random forest, each decision tree

is built on a *randomly selected subset* of the training data, and a *randomly selected subset of features* is used for each tree. This helps to reduce overfitting and increase the accuracy of the model.

Gradient Boosting, on the other hand, is also an ensemble learning method that combines multiple weak learners to make a strong learner. In Gradient Boosting, a sequence of decision trees is built, where each tree tries to correct the errors of the previous tree. The algorithm starts by building a simple model (say mean in regression problems) and then gradually improves it by adding more complex models. Gradient Boosting typically requires more training time than Random Forest, but it can often achieve higher accuracy.

In summary, the main differences between Random Forest and Gradient Boosting are:

- Random Forest builds multiple decision trees independently, while GB builds a sequence of decision trees, where each tree tries to correct the errors of the previous tree.
- Random Forest uses random subsets of data and features to build each tree, while GB uses all the data but focuses on the instances that were misclassified by the previous trees.
- Random Forest is less prone to overfitting, while Gradient Boosting can overfit if the number of trees or their complexity is too high.
- Random Forest is typically faster to train than Gradient Boosting, but Gradient can achieve higher accuracy.

- Support Vector Methods: Support vector machine algorithms are used to accomplish both classification (SVM, M for Machine) and regression (SVR, R for regression) tasks. With SVR, we can then give our model some flexibility in finding the predicted values, as long as the error is within that range. In contrast to OLS discussed above in Linear regression, the objective function of SVR is to minimize the coefficients — more specifically, the norm of the coefficient vector — not the squared error. The error term is instead handled in the constraints, where we set the absolute error less than or equal to a specified margin, called the maximum error, ϵ. We can tune epsilon to gain the desired accuracy of our model. Our new objective function and constraints are as follows:

$$\text{Min.}|\mathbf{w}|^2 \tag{2.3}$$

with contraints: $|y_i - w_i x_i| \leq \epsilon$. We can add another hyperparameter ξ. This is called a slack variable and the idea is simple – for any value that falls outside of ϵ, we can denote its deviation from the margin as ξ. We know that these deviations have the potential to exist, but we would still like to minimize them as much as possible. Thus, we can add these deviations to the objective function.

$$\text{Min.}|\mathbf{w}|^2 + C\sum_{i=1}^{N}|\xi_i| \tag{2.4}$$

with contraints: $|y_i - w_i x_i| \leq \epsilon + |\xi_i|$. These are supervised machine learning algorithms that plot each piece of data in the n-dimensional space, with n referring to the number of features. Each feature value is associated with a coordinate value, making it easier to plot the features. Moreover, classification is further performed by distinctly determining the hyper-plane that separates the two sets of support vectors or classes. A good separation ensures a good classification between the plotted data points. In short, SVMs represent the coordinates for individual observations. These are popular machine learning classifiers used in applications such as data classification, facial expression classification, text classification, steganography detection in digital images, speech recognition, and others.

SVM often admits the kernel trick. Kernels are used in SVM to map the original input data into a higher-dimensional space where it will be easier to find patterns in the data and train the model with better performance (or making the data more linear!). If we have binary class data like a half-moon patterns of blue and red, when plotted in 2D space, a linear SVM kernel will not be able to differentiate the two classes well but we can use RBF (radial basis function) kernel which maps the data into a particular higher-dimensional space where the two classes are clearly separable.

Typically without the kernel trick, in order to calculate support vectors and support vector classifiers, we need first to transform data points one by one to the higher dimensional space, and do the calculations based on SVM equations in the higher dimensional space, then return the results. The 'trick' in the kernel trick is that we design the kernels based on some conditions as mathematical functions that are equivalent to a dot product in the higher dimensional space without even having to transform data points to the higher dimensional space. i.e we can calculate support vectors and support vector classifiers in the same space where the data is provided which saves a lot of time and calculations.

- Naive Bayes: This refers to a probabilistic machine learning algorithm based on the Bayesian probability model and is used to address classification problems. The fundamental assumption of the algorithm is that features under consideration are independent of each other and a change in the value of one does not impact the value of the other. For example, you can consider a ball, a cricket ball, if it is red, round, has a 7.1-7.26 cm diameter, and has a mass of 156-163 g. Although all these features could be interdependent, each one contributes to the probability that it is a cricket ball. This is the reason the algorithm is referred to as 'naïve'. Each variable in the dataset is independent of the other. This kind of assumption is unrealistic for real-world data. However, even with this assumption, it is very useful for solving a range of complicated problems, e.g., spam email classification, etc. Let's look at the mathematical representation of the algorithm. If X, Y = probabilistic events, $P(X)$ = probability of X being true, $P(X|Y)$ = conditional probability of X happening in case Y has occured. Then, Bayes' theorem is given by the equation:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.5}$$

A naive Bayesian approach is easy to develop and implement. It is capable of handling massive datasets and is useful for making real-time predictions. Its applications include spam filtering, sentiment analysis and prediction, document classification, and others.

- kNN (k Nearest Neighbor): KNN algorithm is used for both classification and regression problems. It stores all the known use cases and classifies new use cases (or data points) by segregating them into different classes. This classification is accomplished based on the similarity score of the recent use cases to the available ones. KNN is a supervised machine learning algorithm, wherein 'K' refers to the number of neighboring points we consider while classifying and segregating the known n groups. The algorithm learns at each step and iteration, thereby eliminating the need for any specific learning phase. The classification is based on the neighbor's majority vote. The algorithm uses these steps to perform the classification: For a training dataset, calculate the distance between the data points that are to be classified and the rest of the data points. Choose the closest 'K' elements based on the distance or function used. Consider a 'majority vote' between the K points–the class or label dominating all data points reveals the final ranking. The real-life applications of KNN algorithms include facial recognition, text mining, and recommendation systems such as Amazon, Netflix, and others. This algorithm is known as **instance-based** learning. These work by memorising the training dataset. The term "non-parametric" refers to not making any assumptions on the underlying data distribution. These methods do not have any fixed numbers of parameters in the model. Similarly in KNN, the model parameters grow with the training data by considering each training case as a parameter of the model. KNN is a non-parametric algorithm. If KNN is used for a regression problem, the mean (or median) of the y variables of the K closest observations will be used for predictions. If KNN is used for a classification problem, it's the mode (the value that appears most often) of the variables y of the K closest observations that will be used for predictions. The basic steps are as follows:

  Use a value of N and do the following for every new sample we want to classify:

  - Calculate the distance with every sample from the dataset.
  - Pick the N closest samples from the dataset.
  - Returns the mode of the labels of those N samples **OR** the mean (or median).

- k-Means: This is a distance-based unsupervised machine learning algorithm that accomplishes clustering tasks. In this algorithm, you classify datasets into clusters (K clusters) where the data points within one set remain homogenous, and the data points from two different clusters remain heterogeneous.

  The clusters under k-Means are formed using these steps:

  1. Initialization: The k-means algorithm selects centroids for each cluster (k number

of points).

2. Assign objects to centroid: Clusters are formed with the closest centroids (k clusters) at each data point.
3. Centroid update: Create new centroids based on existing clusters and determine the closest distance for each data point based on new centroids. Here, the position of the centroid also gets updated whenever required.
4. Repeat: Repeat the process till the centroids do not change.

K-Means clustering is useful in applications such as clustering Facebook users with common likes and dislikes, document clustering, segmenting customers who buy similar e-commerce products, etc. Clusters are evaluated based on some similarity or dissimilarity measure such as the distance between cluster points. If the clustering algorithm separates dissimilar observations apart and similar observations together, then it has performed well. Some popular metrics evaluation metrics for clustering algorithms are:

Silhouette score: It measures the similarity of observation to its own cluster compared to other clusters. The score ranges from -1 to 1, with values closer to 1 indicating a stronger clustering structure. Calinski-Harabasz Index: It measures the ratio of the between-cluster variance to the within-cluster variance. Higher values indicate a better clustering solution. Davies-Bouldin Index: It measures the average similarity between each cluster and its most similar cluster. Lower values indicate a better clustering solution. Adjusted Rand Index: It measures the similarity between the true class labels and the predicted cluster labels, adjusted for the chance. Higher values indicate a better clustering solution. Confusion matrix: It can be used to evaluate the accuracy of clustering models by comparing the predicted clusters to the true classes.

Dunn's Index:

$$\text{min.min.}\Big(\frac{\delta(X_i, X_j)}{\text{max.}\Delta X_k}\Big)$$

intercluster distance, intracluster (within cluster) is given by ΔX_k.

We list some advantages of the k-means clustering algorithm:

- Easy to comprehend, robust, and fast.
- Efficient with $\mathcal{O}(tknd)$ complexity where t is number of iterations, n is number of objects/points and d is the dimensionality of each object. Usually we have: $k, t, d \ll n$.
- Best result when the data sets are distinct and well separated from each other.

Now, we note some drawbacks:

- The algorithm requires the apriori specification of the number of cluster centres.

  - Cannot resolve clusters with highly overlapping data.
  - The algorithm is not invariant to non-linear transformations, i.e., different representations of data reveal different results.
  - The algorithm fails for categorical data and is applicable only when the mean is defined.
  - The algorithm fails for a non-linear data set.
  - Unable to handle noisy data and outliers.

- Random forest: Random forest algorithms use multiple decision trees to handle classification and regression problems. It is a supervised machine learning algorithm where different decision trees are built on different samples during training. These algorithms help estimate missing data and tend to keep the accuracy intact in situations when a large chunk of data is missing in the dataset. Random forest algorithms follow these steps:

  1. Select random data samples from a given data set.
  2. Build a decision tree for each data sample and provide the prediction result for each decision tree.
  3. Carry out voting for each expected result.
  4. Select the final prediction result based on the highest voted prediction result.

  This algorithm finds applications in finance, ecommerce (recommendation engines), computational biology (gene classification, biomarker discovery), and others. It uses the random subspace method, also called attribute bagging or feature bagging. It is an ensemble learning method that attempts to reduce the correlation between estimators in an ensemble by training them on *random* samples of features instead of the entire feature set.

  Side Note: Bagging is an ensemble learning method. It stands for bootstrap aggregating. In this technique, we generate some data using the bootstrap method, in which we use an already existing dataset and generate multiple samples of the N size. This bootstrapped data is then used to train multiple models in parallel, which makes the bagging model more robust than a simple model. Once all the models are trained, when it's time to make a prediction, we make predictions using all the trained models and then average the result in the case of regression, and for classification, we choose the result, generated by models, that have the highest frequency (or majority voting).

There are primarily three different types of neural networks algorithms in deep learning (DL) that form the basis for most pre-trained models in DL:

- Artificial Neural Networks (ANN): Artificial neural networks are machine learning algorithms that mimic the human brain (neuronal behavior and connections) to solve complex problems. ANN has three or more interconnected layers in its computational model that process the input data. The first layer is the input layer or neurons that send input data to deeper

layers. The second layer is called the hidden layer. The components of this layer change or tweak the information received through various previous layers by performing a series of data transformations. These are also called neural layers. The third layer is the output layer that sends the final output data for the problem. ANN algorithms find applications in smart home and home automation devices such as door locks, thermostats, smart speakers, lights, and appliances. They are also used in the field of computational vision, specifically in detection systems and autonomous vehicles.

- Convolution Neural Networks (CNN): Convolutional neural networks (CNN) are all the rage in the deep learning community right now are being used across different applications and domains, and they're especially prevalent in image and video processing projects. They mostly deal with two-dimensional objects (height and width of image). Used in image recognition, face detection, image analysis etc.

- Recurrent Neural Networks (RNN): Recurrent neural networks refer to a specific type of ANN that processes sequential or temporal data. Here, the result of the previous step acts as the input to the current step. This is facilitated via the hidden state that remembers information about a sequence. It acts as a memory that maintains the information on what was previously calculated. The memory of RNN reduces the overall complexity of the neural network. *RNN analyzes time series data and possesses the ability to store, learn, and maintain contexts of any length.* RNN is used in cases where time sequence is of paramount importance, such as speech recognition, language translation, video frame processing, text generation, and image captioning. RNNs are a kind of feedforward network, in which information from one layer passes to another layer, and each node in the network performs mathematical operations on the data. These operations are temporal, i.e., RNNs store contextual information about previous computations in the network. It is called recurrent because it performs the same operations on some data every time it is passed. However, the output may be different based on past computations and their results. Useful in speech recognition, text translation, NLP etc. Unlike CNN, they can have different input and output sizes.

GNN: Graph neural networks (GNNs) to solve graph prediction tasks. A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).

2.1 Exploratory Data Analysis (EDA)

EDA is a crucial step in the data science process. We can have numerical or categorical data. CD can further be of ordinal (size of shirt: L, XL, XXL) or nominal type (color of shirt: Blue, Green, Red). We can convert categorical into numerical data using two common methods: 1. Integer encoding, 2. One-Hot Encoding.

In integer encoding, ach unique category value is assigned an integer value. For example, 'red' is 1, 'green' is 2, and 'blue' is 3. This is called a label encoding or an integer encoding and is easily

reversible. For some variables, this may be enough. Ordinal variables like the example above would be a good example where a label encoding would be sufficient.

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. Using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories). In this case, a *one-hot encoding* can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the "color" variable example, there are 3 categories and therefore 3 binary variables are needed. A '1' value is placed in the binary variable for the color and '0' values for the other colors. For example:

$$\begin{array}{ccc} \textit{Blue} & \textit{Green} & \textit{Red} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \tag{2.6}$$

```python
# import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# load the dataset
df = pd.read_csv("titanic.csv")

# print the first few rows of the DataFrame
print(df.head())

# print the DataFrame's shape
print(df.shape)

# print the DataFrame's data types
print(df.dtypes)

# check for missing values
print(df.isnull().sum())

# visualize the distribution of a numeric column
plt.hist(df['Age'])
plt.show()

# visualize the distribution of a categorical column
df['Sex'].value_counts().plot(kind='bar')
plt.show()
```

```
29
30  # calculate basic statistics for a numeric column
31  print(df['Fare'].describe())
32
33  # upto some decimal places
34  pd.set_option('display.float_format', lambda x: '%.3f' % x)
35
36  # calculate the correlation between two numeric columns
37  print(df['Fare'].corr(df['Survived']))
38
39  # group the data by a categorical column and calculate statistics
40  grouped_df = df.groupby('Pclass')['Survived'].mean()
41  print(grouped_df)
42
43  # create a scatter plot to visualize the relationship between two numeric columns
44  plt.scatter(df['Age'], df['Fare'])
45  plt.xlabel('Age')
46  plt.ylabel('Fare')
47  plt.show()
48
49  # create a box plot to visualize the distribution of a numeric column
50  plt.boxplot(df['Fare'])
51  plt.ylabel('Fare')
52  plt.show()
53
54  # create a bar plot to visualize the mean of a numeric column for each category of a
        categorical column
55  df.groupby('Sex')['Age'].mean().plot(kind='bar')
56  plt.ylabel('Average Age')
57  plt.show()
58
59  # create a pivot table to summarize the data
60  pivot_table = df.pivot_table(index='Sex', columns='Pclass', values='Fare', aggfunc='mean')
61  print(pivot_table)
62
63  # create a heatmap to visualize the pivot table
64  plt.pcolor(pivot_table, cmap='Reds')
65  plt.colorbar()
66  plt.show()
67
68  # create a pairplot to visualize the relationships between multiple numeric columns
69  import seaborn as sns
70  sns.pairplot(df, vars=['Age', 'Fare', 'SibSp'])
71  plt.show()
72
73  # create a bar plot to visualize the count of a categorical column
74  df['Embarked'].value_counts().plot(kind='bar')
75  plt.ylabel('Count')
```

```
plt.show()

# create a countplot to visualize the count of a categorical column by the categories of another categorical column
sns.countplot(x='Sex', hue='Pclass', data=df)
plt.show()

# create a point plot to visualize the mean of a numeric column by the categories of a categorical column
sns.pointplot(x='Sex', y='Age', data=df)
plt.ylabel('Average Age')
plt.show()

# create a violin plot to visualize the distribution of a numeric column by the categories of a categorical column
sns.violinplot(x='Sex', y='Age', data=df)
plt.ylabel('Age')
plt.show()

# create a box plot to visualize the distribution of a numeric column by the categories of a categorical column
sns.boxplot(x='Sex', y='Age', data=df)
plt.ylabel('Age')
plt.show()

# create a swarm plot to visualize the distribution of a numeric column by the categories of a categorical column
sns.swarmplot(x='Sex', y='Age', data=df)
plt.ylabel('Age')
plt.show()

# create a faceting grid to visualize the distribution of multiple numeric columns by the categories of a categorical column
g = sns.FacetGrid(df, col='Sex')
g.map(plt.hist, 'Age')
plt.show()

# create a heatmap to visualize the correlation between multiple numeric columns
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap='RdYlGn', annot=True)
plt.show()

# create a lag plot to check for autocorrelation in a numeric column
from pandas.plotting import lag_plot
lag_plot(df['Fare'])
plt.show()

# create an autocorrelation plot to visualize the autocorrelation in a numeric column
```

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df['Fare'])
plt.show()

# create a scatter plot matrix to visualize the relationships between multiple numeric columns
from pandas.plotting import scatter_matrix
scatter_matrix(df[['Age', 'Fare', 'SibSp']], alpha=0.2, figsize=(6, 6))
plt.show()

# create a regression plot to visualize the relationship between two numeric columns
sns.regplot(x='Age', y='Fare', data=df)
plt.show()

# create a barplot to visualize the mean of a numeric column by the categories of a
    categorical column
sns.barplot(x='Sex', y='Age', data=df)
plt.ylabel('Average Age')
plt.show()

# create a pointplot to visualize the mean and confidence interval of a numeric column by the
    categories of a categorical column
sns.pointplot(x='Sex', y='Age', data=df, ci=95)
plt.ylabel('Average Age')
plt.show()

# create a lmplot to visualize the relationship between two numeric columns and the
    categories of a categorical column
sns.lmplot(x='Age', y='Fare', hue='Sex', data=df)
plt.show()

# create a factorplot to visualize the distribution of a numeric column by the categories of
    a categorical column
sns.factorplot(x='Sex', y='Age', data=df)
plt.ylabel('Average Age')
plt.show()

# create a boxenplot to visualize the distribution of a numeric column by the categories of a
    categorical column
sns.boxenplot(x='Sex', y='Age', data=df)
plt.ylabel('Age')
plt.show()

# create a distplot to visualize the distribution of a numeric column
sns.distplot(df['Fare'])
plt.show()

# create a kdeplot to visualize the kernel density estimate of a numeric column
sns.kdeplot(df['Fare'])
```

```
plt.show()

# create a rugplot to visualize the distribution of a numeric column
sns.rugplot(df['Fare'])
plt.show()

# create a jointplot to visualize the relationship between two numeric columns and their
    distributions
sns.jointplot(x='Age', y='Fare', data=df)
plt.show()
```

2.2 Data preprocessing

We list some steps for data preprocessing below:

- Handling missing values: This technique is used when there are missing values in the dataset. There are various ways to handle missing values, such as filling them with the mean, median, or mode of the column, or dropping rows with missing values. The appropriate method will depend on the specific dataset and the goal of the analysis.
- Encoding categorical variables: This technique is used when the dataset contains categorical variables, which are variables that can take on a limited number of categories. One-hot encoding is a common method for encoding categorical variables, which creates a new binary column for each category. This is useful for inputting categorical variables into machine learning models, which typically only accept numerical input.
- Standardizing numeric columns: This technique is used to scale the values of a numeric column so that they have zero mean and unit variance. This is often useful when the numeric columns have different scales and the machine learning model will be sensitive to this difference in scales.
- Normalizing numeric columns: This technique is used to scale the values of a numeric column so that they have a minimum value of 0 and a maximum value of 1. This is often useful when the numeric columns have different scales and the machine learning model will be sensitive to this difference in scales.
- Binning numeric columns: This technique is used to divide the values of a numeric column into bins. This is useful for turning a continuous numeric column into a categorical column, which can be useful for certain types of analysis or machine learning models.
- Applying min-max scaling: This technique is used to scale the values of a numeric column so that they have a minimum value of 0 and a maximum value of 1. This is often useful when the numeric columns have different scales and the machine learning model will be sensitive to this difference in scales.

- Applying robust scaling: This technique is used to scale the values of a numeric column using the median and interquartile range. This is often useful when the data contains outliers, as it is less sensitive to the influence of outliers compared to other scaling methods.
- Applying power transformations: Power transformations are a class of functions that can be used to transform the values of a numeric column in order to stabilize or improve the assumptions of certain statistical models. Power transformations can be useful for correcting the skewness of a distribution, as skewed distributions can cause problems when fitting certain types of models.
- Applying quantile transformations: This technique is used to transform the values of a numeric column so that they have a uniform or normal distribution. This can be useful for improving the assumptions of certain machine learning models, which may assume that the predictor variables are normally distributed.
- Applying box-cox transformations: This technique is used to transform the values of a numeric column so that they are approximately normally distributed. This can be useful for improving the assumptions of certain machine learning models, which may assume that the predictor variables are normally distributed.

```
# create a copy of the original DataFrame
df_preprocessed = df.copy()

# handle missing values in the DataFrame
df_preprocessed['Age'].fillna(df_preprocessed['Age'].median(), inplace=True)
df_preprocessed.dropna(inplace=True)

# encode categorical variables using one-hot encoding
df_preprocessed = pd.get_dummies(df_preprocessed, columns=['Sex', 'Pclass'], prefix=['sex',
    'pclass'])

# standardize the values of a numeric column
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_preprocessed['Age_scaled'] = scaler.fit_transform(df_preprocessed[['Age']])

# normalize the values of a numeric column
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()
df_preprocessed['Age_normalized'] = normalizer.fit_transform(df_preprocessed[['Age']])

# bin the values of a numeric column
from sklearn.preprocessing import KBinsDiscretizer
```

```
25
26  discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal')
27  df_preprocessed['Age_binned'] = discretizer.fit_transform(df_preprocessed[['Age']])
28
29  # apply a min-max scaling to a numeric column
30  from sklearn.preprocessing import MinMaxScaler
31
32  scaler = MinMaxScaler()
33  df_preprocessed['Age_scaled'] = scaler.fit_transform(df_preprocessed[['Age']])
34
35  # apply a robust scaling to a numeric column
36  from sklearn.preprocessing import RobustScaler
37
38  scaler = RobustScaler()
39  df_preprocessed['Age_scaled'] = scaler.fit_transform(df_preprocessed[['Age']])
40
41  # apply a power transformation to a numeric column
42  from sklearn.preprocessing import PowerTransformer
43
44  transformer = PowerTransformer(method='yeo-johnson')
45  df_preprocessed['Age_transformed'] = transformer.fit_transform(df_preprocessed[['Age']])
46
47  # apply a quantile transformation to a numeric column
48  from sklearn.preprocessing import QuantileTransformer
49
50  transformer = QuantileTransformer(output_distribution='normal')
51  df_preprocessed['Age_transformed'] = transformer.fit_transform(df_preprocessed[['Age']])
52
53  # apply a box-cox transformation to a numeric column
54  from scipy.stats import boxcox
55
56  df_preprocessed['Age_transformed'], lambda_ = boxcox(df_preprocessed['Age'])
```

Section 3

Deep Learning

Deep learning uses artificial neural networks to perform sophisticated computations on large amounts of data. It is a type of machine learning that works based on the structure and function of the human brain. By deep, we mean that there are many layers separating the input and output (hidden layers). Deep learning algorithms train machines by learning from examples. Industries such as health care, eCommerce, entertainment, and advertising commonly use deep learning. A neural network is structured like the human brain and consists of artificial neurons, also known as nodes. These nodes are stacked next to each other in three layers: The input layer, the hidden layer(s), the output layer. The most basic neural network is the 'perceptron' also known as forward pass or forward propagation. Irrespective of the number of features in the sample (input), the out

of perceptron is a single value (predicted class). The output comes from multiplying the input vector by the weights (w) and passing the result through the activation function which is a step function in this case.

Here is the list of top 10 most popular deep learning algorithms:

1. Convolutional Neural Networks (CNNs)
2. Long Short Term Memory Networks (LSTMs)
3. Recurrent Neural Networks (RNNs)
4. Generative Adversarial Networks (GANs)
5. Radial Basis Function Networks (RBFNs)
6. Multilayer Perceptrons (MLPs)
7. Self Organizing Maps (SOMs)
8. Deep Belief Networks (DBNs) based on Restricted Boltzmann Machines(RBMs)
9. Deep Autoencoders based on 'autoencoders'.

Long Short Term Memory networks – usually just called 'LSTMs' – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

3.1 Architectures

3.1.1 LeNet

The basic architecture consists of: `CONV => RELU => POOL) * 2 => FC => RELU => FC => SOFTMAX`. Fully connected (FC) is also known as 'dense' layer.

Types of problems: Regression, Classification, Clustering

1. Classification: Logistic regression, k-NN, Support Vector Machine (SVM), Naive Bayes, Decision Tree, Random Forest Classifiers
2. Regression: Linear (Linear, Lasso, ..), Polynomial Regression, Support Vector Regression (SVR), Decision Tree Regression, Random Forest Regression
3. Clustering: These are unsupervised algorithms. k-Means Clustering (KMC), Hierarchical Clustering (HC)

The two clustering models have their own pros and cons. For example, the pros of k-means is that it is *simple to understand, easily adaptable, works well on small or large datasets, fast, efficient and performant* while the cons is *choice of number of cluster is crucial, though can be optimally*

selected. The pros of HC is the fact that the *optimal number of clusters can be obtained by the model itself, practical visualisation with the dendrogram*. The drawback is that it is *not appropriate for large datasets*. These can be done using following commands: `import scipy.cluster.hierarchy as sch, dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))`. Fitting HC to the dataset can be achieved by: `from sklearn.cluster import AgglomerativeClustering`

Unsupervised learning can be divided into three parts: Clustering, Dimensional Reduction based on Principal Component Analysis (PCA), and Association Problems. Association problems are based on identifying sequences: such as what clothes do I wear together?

Principal Component Analysis (PCA) can be thought of as fitting a p-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small. PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. It is method for feature extraction.

Logistic regression is a classification algorithm traditionally limited to only two-class classification problems. If you have more than two classes then LDA (Linear Discriminant Analysis) is the preferred linear classification technique.

PCA is an unsupervised learning algorithm while LDA is a supervised learning algorithm. This means that PCA finds directions of maximum variance regardless of class labels while LDA finds directions of maximum class separability.

LDA makes some assumptions as below:

- The data is Gaussian, that each variable is is shaped like a bell curve when plotted.
- Each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.

LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made. The model uses Bayes Theorem to estimate the probabilities.

Some extensions of LDA

- Quadratic Discriminant Analysis (QDA): Each class uses its own estimate of variance (or covariance when there are multiple input variables).
- Flexible Discriminant Analysis (FDA): Where non-linear combinations of inputs is used such as splines.
- Regularized Discriminant Analysis (RDA): Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

Closely related idea if that of KPCA (kernel principal component analysis) which is nonlinear dimension reduction method. This transforms the data that is not linearly separable onto a new,

lower-dimensional subspace that is suitable for linear classifiers. In Scikit-learn, it can be called as: `from sklearn. decomposition import KernelPCA`

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
sklearn_lda = LDA(n_components=2)
X_lda_sklearn = sklearn_lda.fit_transform(X, y)
```

```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
pca = PCA(n_components=2)
pca.fit(X)
PCA(n_components=2)
print(pca.explained_variance_ratio_)
print(pca.singular_values_)
```

Between supervised and unsupervised learning, we have something which is - reinforcement learning (RL). It is the case where we have no data at all, like city for a self-driven car. In this sense, the entire knowledge about roads and traffic rules won't teach the car how to drive itself. The goal is to minimize errors and not to predict the moves! Survival of the fittest or rather the algorithm is the sole motive of RL.

Q-learning is a commonly used model-free approach which can be used for building a self-playing PacMan agent. It revolves around the notion of updating Q values which denotes value of performing action a in state s. The following value update rule is the core of the Q-learning algorithm.

3.2 Data cleaning

It is part of Data wrangling which includes importing, cleaning, structuring, handling missing values, text mining.

It is important aspect of data science and can be summarised below in some steps:

- Missing Values: Identify missing values in the dataset. Decide on an appropriate method to handle missing values, such as imputation or deletion. Impute missing values with appropriate values based on the method selected, such as mean, median, or mode. In Pandas we can use: `df.isna()` and `df.isna().sum()` to check which enteries are not available.

  Alternatively, for mean imputation we can use: in Pandas– `df.fillna(df.mean())`

```
def show_missing(df):
    """Return a Pandas dataframe describing the contents of a source dataframe
    including missing values."""

    variables = []
    dtypes = []
```

```
    count = []
    unique = []
    missing = []
    pc_missing = []

    for item in df.columns:
        variables.append(item)
        dtypes.append(df[item].dtype)
        count.append(len(df[item]))
        unique.append(len(df[item].unique()))
        missing.append(df[item].isna().sum())
        pc_missing.append(round((df[item].isna().sum() / len(df[item])) * 100, 2))

    output = pd.DataFrame({
        'variable': variables,
        'dtype': dtypes,
        'count': count,
        'unique': unique,
        'missing': missing,
        'pc_missing': pc_missing
    })

    return output
```

- Duplicates: Identify and remove duplicate records in the dataset. Verify that all duplicates have been removed
- Outliers: Identify and handle outliers or extreme values in the data Decide on an appropriate method to handle outliers, such as deletion, transformation, or imputation. Impute or delete outliers based on the method selected such as RANSAC
- Data Format: Verify that all data is in the correct format. Convert data into the appropriate format, such as converting dates into a common format. Handle inconsistent data formats
- Data Validity: Verify all data is valid and consistent. Check for errors (incorrect values or typos) and correct them
- Data Standardization and Normalization: Data is generally processed in one of two ways: data standardization or data normalization, sometimes referred to as min-max scaling. The two most popular techniques for scaling numerical data prior to modeling are normalization and standardization. Normalization scales each input variable separately to the range 0-1, which is the range for floating-point values where we have the most precision. Standardization scales each input variable separately by subtracting the mean (called centering) and dividing by the standard deviation to shift the distribution to have a mean of zero and a standard deviation of one. This can be achieved in PANDAS as

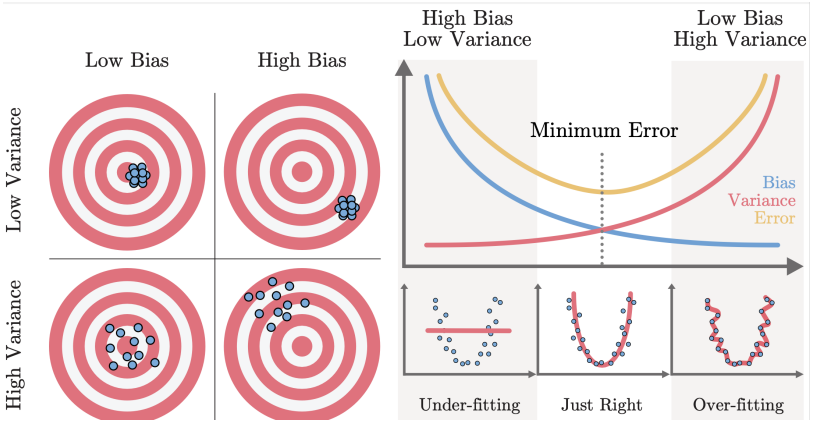


Figure 2. Diagrammatic representation of bias and variance and how we can over/under fit the model. Definition: Variance indicates how much the estimate of the target function will alter if different training data were used. In other words, variance describes how much a random variable differs from its expected value. Variance is based on a single training set. Variance measures the inconsistency of different predictions using different training sets - it's not a measure of overall accuracy. Bias is the amount that a model's prediction differs from the target value, compared to the training data. Bias error results from simplifying the assumptions used in a model so the target functions are easier to approximate.

Section 4

Definitions:

In k-fold cross-validation, we divide the dataset into k equal parts. After this, we loop over the entire dataset k times. In each iteration of the loop, one of the k parts is used for testing, and the other $k-1$ parts are used for training. Using k-fold cross-validation, each one of the k parts of the dataset ends up being used for training and testing purposes. An improvement of this is stratified k-fold cross-validation.

A special case of k-fold cross-validation is LOOCV (leave-one-out cross-validation) where we do not leave one training set for test but rather choose $k = n$ so that only one training example is used for testing during each iteration.

4.1 Confusion matrix

A matrix that lays out the performance of a learning algorithm. It has four elements, **TP** (true positive), **TN** (true negative), **FP** (false positive) and **FN** (false negative).

$$\mathbf{ERR} = \frac{FP + FN}{FP + FN + TP + TN} \tag{4.1}$$

whereas the prediction accuracy is 1-ERR. We also have FPR, TPR.

$$\mathbf{FPR} = \frac{FP}{FP + TN} = 1 - \mathbf{TNR} \tag{4.2}$$

$$\mathbf{TPR} = \frac{TP}{FN + TP} \tag{4.3}$$

A

REC (recall) is equal to TPR while PRE (precision) is defined as:

$$\mathbf{PRE} = \frac{TP}{FP + TP} \tag{4.4}$$

All same meaning: [Sensitivity, Recall, Hit rate, or True positive rate] and [Specificity, Selectivity or True negative rate (TNR)]

F1 score is the harmonic mean of **PRE** and **REC**, and it is calculated using the following formula:

$$F1 = \frac{2 \cdot \mathbf{PRE} \cdot \mathbf{REC}}{\mathbf{PRE} + \mathbf{REC}}$$

The F1 score is used when the recall and the precision are equally important.

4.2 Gradient Descent (SGD)

In most of the problems in machine learning, we have to optimize the loss function. The optimizer is an algorithm that adjusts the weights to minimize the loss. Nearly all of the optimization algorithms used in deep learning belong to a family called stochastic gradient descent. They are iterative algorithms that train a network in steps. One step of training goes is as follows:

- Sample some training data and run it through the network to make predictions.
- Measure the loss between the predictions and the true values.
- Adjust the weights in a direction that makes the loss smaller.

SGD is little different and has advantages of finding global minimum over GD methods. Why is it called SGD? – It is called so because, the gradient is a vector that specifies in what direction the weights need to move. More precisely, it tells us how to change the weights to make the loss change fastest. We call our process gradient descent because it uses the gradient to descend the loss curve towards a minimum. Stochastic means 'randomness' or with a noise. Each step of SGD is taken in a direction which is not necessarily steepest downhill direction due to added noise.

The mechanism for introducing this randomness is simple. At each step, the algorithm chooses a random subset (called 'batch') of training data and computes gradient from this examples alone. Update rule is:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial l_i[\phi_t]}{\partial \phi} \tag{4.5}$$

where $\mathcal{B}_t$ is a set containing rhe indices of the input/output pairs in current batch and l_i is the loss due to the ith pair. α is known as the learning rate (aka step size).

Steepest descent is a special case of gradient descent where the step size is chosen to minimize the objective function value. Gradient descent refers to any of a class of algorithms that calculate the gradient of the objective function, then move 'downhill' in the indicated direction; the step length can be fixed, estimated (e.g., via line search).

Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

- Overfitting: The model captures the training data well but fails to generalize to unseen data. High variance, Low bias.
- Underfitting: The model does not capture the training data well. High variance, high bias.

We define variance as measure of how much deviation occurs for particular case if we retrain the model multiple times using different subsets of training data.

How to reduce overfitting

We can either do regularization or *dimensional reduction via feature selection.*

4.3 Activation function

An activation function is a function used in artificial neural networks which outputs a small value for small inputs, and a larger value if the inputs exceed a threshold. If the inputs are large enough, the activation function "fires", otherwise it does nothing. In other words, an activation function is like a gate that checks that an incoming value is greater than a critical number.

Activation functions are useful because they add non-linearities into neural networks, allowing the neural networks to learn powerful operations. If the activation functions were to be removed from a feedforward neural network, the entire network could be re-factored to a simple linear operation or matrix transformation on its input, and it would no longer be capable of performing complex tasks such as image recognition. *A neural network with no activation functions is equivalent to a linear regression model, and can perform no operation more complex than linear modeling.*

Well-known activation functions used in data science include the rectified linear unit (ReLU) function, and the family of sigmoid functions such as the logistic sigmoid function, the hyperbolic tangent, and the arctangent function. Two commonly used activation functions: the rectified linear unit (ReLU) and the logistic sigmoid function. The ReLU has a hard cutoff at 0 where its behavior changes, while the sigmoid exhibits a gradual change. Both tend to 0 for small x, and the sigmoid tends to 1 for large x.

4.3.1 ReLU derivative

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{4.6}$$

The reason for the derivative being undefined at $x = 0$ is that its left- and right derivative are not equal. This sometimes creates problem with 'backpropagation' since when training a neural network, a neuron can become 'trapped' in the zero region and backpropagation will never change its weights. Because of the zero gradient problem faced by the ReLU, it is sometimes common to use an adjusted ReLU function called the parametric rectified linear unit, or PReLU:

$$f(\alpha, x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{4.7}$$

This has the advantage that the gradient is nonzero at all points (except 0 where it is undefined).

$$f(x) = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)} \tag{4.8}$$

Softmax's i^{th} component is defined as: $\frac{\exp(x_i)}{\sum_j \exp(x_j)}$ The function Softmax S is also a non-linear function but it is special in that it usually is the last operation done in a network. It takes in a vector of real numbers and returns a probability distribution with components summing to 1.

The backpropagation computes $L(\theta)$ using Chain rule and then weights are updated as: $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta)$. η is also referred to as learning rate. Just replacing standard SGD with an optimizer like Adam or RMSProp will boost performance a lot.

However, there are cases where the gradient gets stuck at zero or alternatively explodes. One technique that can help with the exploding gradient problem is gradient clipping, where the gradients are capped at a maximum value to prevent them from growing too large. In addition, weight regularization can also help prevent the gradients from exploding and improve the stability of the network. Another way to control this is to include batch normalization. This will normalize the input to the network and prevent the values from reaching the edges where the gradients are too large. We do this in tensor networks remember! ?

For zero gradient issue, we can use different activation function like ReLU. One of the recent ways to resolve the vanishing gradient problem is with residual neural networks, or ResNets (not to be confused with recurrent neural networks). Some others are – multi-level hierarchy and LSTM.

Section 5

Stuff from Binomial

In image processing and learning, test-time augmentation, is a technique that relies on augmenting the original image before running it through a model. By doing this, we can give the model more opportunities to compute the correct prediction.

When we run three pictures through the model, we will get back three different vectors. A good approach is to average all three vectors and use the result to compute the correct class. By doing this, we can take advantage of each picture to make the final decision.

5.1 Fun with matrices

Non-negative matrix factorization (Lee, 2000).

```
import numpy as np
X = np.array([[1, 2], [2, 2], [3, 1.2], [4, 1], [5, 0.8], [6, 1]])
from sklearn.decomposition import NMF
model = NMF(n_components=2, init='random', random_state=0)
W = model.fit_transform(X)
H = model.components_
```

5.2 Back propagation

Backpropagation can be thought of as a computationally efficient approach to compute the partial derivatives of a complex cost function in multilayer NNs. The challenge in the parameterization of NN is that we are typically dealing with a very large number of weight coefficients in a high-dimensional feature space. In contrast to cost functions of single-layer NNs such as Adaline (single layer), which we have seen in previous chapters, the error surface of an NN cost function is not convex or smooth with respect to the parameters. There are many bumps in this high-dimensional cost surface (local minima) that we have to overcome in order to find the global minimum of the cost function.

Automatic differentiation comes with two modes, the forward and reverse modes. Backpropagation is simply a special case of reverse-mode automatic differentiation. The trick of reverse mode is that we start from right to left: we multiply a matrix by a vector, which yields another vector that is multiplied by the next matrix and so on. Matrix-vector multiplication is computationally much cheaper than matrix-matrix multiplication, which is why backpropagation is one of the most popular algorithms used in NN training.

We start by computing:

- $textbf\delta^{\text{out}} = \mathbf{a}^{\text{out}} - \mathbf{y}$

where $\mathbf{y}$ is the vector of true class labels. Now, we compute error in the hidden layer as:

$$textbf\delta^{(\text{h})} = textbf\delta^{\text{out}} \left(\mathbf{W}^{\text{out}}\right)^{T} \otimes \frac{\partial \phi(z^{(h)})}{\partial z^{(h)}} \tag{5.1}$$

where derivative is taken of the activation function (sigmoid etc.)

Remember: SGD algorithm requires gradients to be calculated for each variable in the model. Back-propagation is an automatic differentiation algorithm that can be used to calculate the gradients for the parameters in neural networks. Together, the back-propagation algorithm and SGD

algorithm can be used to train a neural network. Something like 'Stochastic Gradient Descent with Back-propagation'. The term back-propagation is often misunderstood as meaning the whole learning algorithm for multi-layer neural networks. Actually, back-propagation refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient.

Fill in .....

Section 6

API

An application programming interface (API) is a computing interface which defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. The way I like to understand an API is that it's an "online function," a function that I can call online. For example: one can have a movie API, which returns names of drama movies when we pass the 'animation' genre as input. The advantage of using such a sort of mechanism is that the API user doesn't get access to the whole dataset or source code and yet they can get all the information they need.

Consider `http://www.omdbapi.com/?i=tt3896198&apikey=9ea43e94`

Endpoint: In the above API call, the endpoint is : `http://www.omdbapi.com/`. Simply this is the location of where the function code is running. API Access Key: Most of the public APIs will have some access key, which you can request. For OMDB API, I had to register and get the API key which is `9ea43e94`.

?Operator: This operator is used to specify the parameters we want to send to the API or our "online function." Here we give two params to our API i.e., IMDB movie ID and API Access Key using the ? operator. Since there are multiple inputs, we use & operator also.

One of the most common use cases for Data Science is how to create an API for getting a model's prediction? Let us assume that we have a Titanic Survival model in place that predicts if a person will survive or not. And, it needs a person's age and sex as input params to predict. We will create this API using FastAPI in two ways: GET and PUT. Don't worry; I will explain each as we go. What is GET? — In a GET request, we usually try to retrieve data using query parameters that are embedded in the query string itself. For example, in the OMDB API, we use the GET request to specify the movie id and access key as part of the query itself. What is PUT? — An alternative to the GET request is the PUT request, where we send parameters using a payload, as we will see in the second method. The payload is not part of the query string, and thus PUT is more secure. It will become more clear when you see the second part. But before we go any further, we need to install FastAPI and uvicorn ASGI server with:

```
    pip install fastapi
pip install uvicorn
```

Section 7

Quantum meets Machine Learning = QML

The idea of QML was first introduced in [1]. We would not talk about unsupervised QML or quantum reinforcement learning and will only make note of supervised QML.

Check out - QISKIT ML
TensorFlow Quantum [2]
And of course, we have Penny Lane developed at Xanadu by Schuld et al.

7.1 Acronyms used in QML

- QNNA: Quantum nearest-neighbor algorithm
- QDTC: Quantum decision tree classifiers
- QKM: Quantum kernel methods
- QSVM: Quantum support vector machines
- QNNC: Quantum neural network classifiers

In QML, we have two data encoding methods. Amplitude encoding (AE) and block encoding (BE).

- AE: The data vector is encoded in the amplitude of the quantum state and then passed to quantum neural network. A 2^n-dimensional vector can be encoded in n qubits as is well-known. AE based QNNs are regarded as 'kernel methods' [3]. Encoding is a very important step! It is often a bottleneck for the runtime.
- BE: This is more practical on NISQ devices. The initial quantum state is fixed and the input data is required to be encoded into the QNN circuit classically in a way similar to the encoding of the variational parameters. There is no need for QRAM (quantum RAM).

For QML, we need to remember that there are some conditions on the preprocessing i.e., unit length of input vector which is because of the normalized state. The result of QML is a measurement made by the observer.

By 2023!

Tensor networks for QML

One of the interesting things is whether we can use the idea of tensor networks to understand QML better. Much work has been done and a good reference page is this. I suspect that there should already be work on encoding using these structures.

[3] Kernel methods owe their name to the use of kernel functions (positive-definite kernel or kernel function is a generalization of a positive-definite function or a positive-definite matrix.), which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space.

Section A

Some probability recap

Question 1: What is the probability to get two heads consecutively when a coin is tossed 5 times. What about general n?

Question 2: Suppose $P(A)$ is the probability that A passes an exam, $P(B)$ is the probability that B passes an exam, $P(A \cap B)$ is that both passes, $P(A \cup B)$ is at least one passes. We have:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Question 3:

$$P(TrmandE) = P(E)P(T|E)$$

. Where T is tsunami with probability 10^{-4} and E is earthquake with probability 10^{-4}. Both are independent so $P(T|E) = P(T)$.

Section B

Some statistics recap

- Central limit theorem: According to CLT, we can treat sampling distribution of any population as normal irrespective of the distribution function of the population. This theorem needs the sample size to be large enough.
- R-squared (R^2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. $R^2 = 1$ means that all variation in dependent can be explained. R-Squared only works as intended in a simple linear regression model with one explanatory variable. With a multiple regression made up of several independent variables, the R-Squared must be adjusted. The adjusted R-squared compares the descriptive power of regression models that include diverse numbers of predictors. Every predictor added to a model increases R-squared and never decreases it. Thus, a model with more terms may seem to have a better fit just for the fact that it has more terms, while the adjusted R-squared compensates for the addition of variables and only increases if the new term enhances the model above what would be obtained by probability and decreases when a predictor enhances the model less than what is predicted by chance. In an overfitting situation, an incorrectly high value of R-squared is obtained, even when the model actually has a decreased ability to predict. This is not the case with the adjusted R-squared.
- p-value: The p-value is a probability that the observed spatial pattern was created by some random process. When the p-value is very small i.e., $p < 0.05$, it means it is very unlikely

that the observed spatial pattern is the result of random processes, so you can reject the null hypothesis. In other words, *a p-value is a number describing how likely it is that your data would have occurred under the null hypothesis of your statistical test.* The p value can only tell you whether or not the null hypothesis is supported. It cannot tell you whether your alternative hypothesis is true, or why. The threshold value for determining statistical significance is also known as the α-value. All statistical tests have a null hypothesis. For most tests, the null hypothesis is that there is no relationship between your variables of interest or that there is no difference among groups. For example, in a t test, the null hypothesis is that the difference between two groups is zero. Recall that a t-test is a statistical test that is used to compare the means of any two groups.

- z-score: z-scores are standard deviations. For example a $z = 2$ means that the result is 2 standard deviations from the mean. Both z-scores and p-values are associated with the standard normal distribution. Very high or very low (negative) z-scores, associated with very small p-values, are found in the tails of the normal distribution. The higher or lower the z-score, the more unlikely the result is to happen by chance and the more likely the result is meaningful.

Kernel density estimation is the process of estimating an unknown probability density function of a random variable using a kernel function $K(u)$. This is important since if we know the distribution (can predict or fit it well), then we know everything about the distribution function. While a histogram counts the number of data points in somewhat arbitrary regions, a kernel density estimate is a function defined as the sum of a kernel function on every data point. The kernel function typically exhibits the following properties: 1) Symmetry i.e., $K(u) = K(-u)$, 2) Normalized to 1 i.e., $\int K(u)du = 1$, 3) Expectation value $E[K] = 0$, 4) Monotonically decreasing such that $K' < 0$ when $u > 0$.

B.1 A/B test

Section C

Recommender Systems

A recommender system (RS), or a recommendation system, is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems are utilized in a variety of areas, with commonly recognized examples taking the form of playlist generators for video and music services, product recommenders for online stores, etc. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. Recommender systems are utilized in order to make better product suggestions to customers, or personalized recommendations to friends.

RSs leverage machine learning algorithms in order to make better predictions about a user's preferences. There are a number of different machine learning algorithms that can be used in a

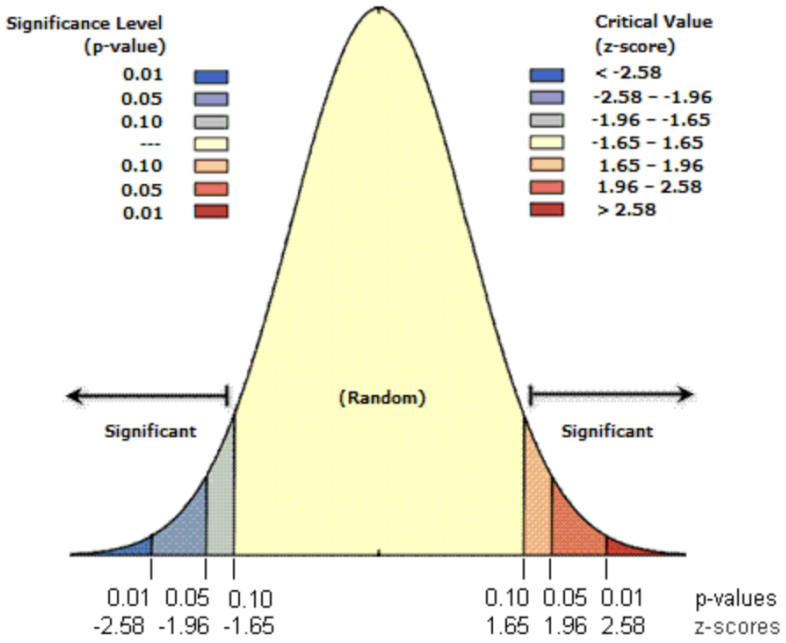


Figure 3. —

recommender system. Each algorithm has its own strengths and weaknesses, and the best algorithm for a particular application will depend on the nature of the data. The most common is the linear regression algorithm. The linear regression algorithm is used to find the best linear approximation to a data set. In a recommender system, this algorithm is used to predict how a user will rate an item based on their past ratings. Other machine learning algorithms that can be used in RS, include some of the following:

- Neural Networks: A neural network is a type of machine learning algorithm that is similar to the brain. It is composed of interconnected neurons that can learn to recognize patterns. Neural networks are often used for prediction tasks, like recommender systems.
- K-NearestNeighbor (K-NN): The K-NN algorithm is often used for recommender systems because it is able to handle large amounts of data and can produce good predictions. The K-NN algorithm works by finding the k nearest neighbours of a given item. The neighbours are then used to vote on the rating of the item. The algorithm then uses the average of the votes to predict the rating of the item. The K-NN algorithm is often used for Recommender Systems because it is able to handle large amounts of data and can produce good predictions.
- Bayesian inference: Bayesian inference is a type of machine learning algorithm that is used to make better predictions. It is often used in recommender systems because it can handle large amounts of data. The Bayesian inference algorithm works by using a probability model

to predict the rating of an item. The algorithm uses the past ratings of a user to build the probability model. This allows the algorithm to make better predictions about a user's preferences.

- Dimensionality reduction: Dimensionality reduction is a type of machine learning algorithm that is used to reduce the number of dimensions in a data set. It is often used in Recommender Systems because it can help to reduce the amount of data that needs to be processed. The dimensionality reduction algorithm works by finding a lower dimensional representation of the data. This can be done by using techniques like Principal Component Analysis (PCA).These are just some of the machine learning algorithms that can be used in Recommender Systems. Each algorithm has its own strengths and weaknesses, and the best algorithm for a particular application will depend on the nature of the data.

The following is a list of benefits / value of building recommender system and why businesses must consider:

- Recommender systems help businesses make money by increasing sales and predicting what customers want.
- Recommender systems can improve customer satisfaction by providing relevant recommendations. They can improve customer loyalty by suggesting new items that the customer may like.
- Recommender systems can reduce the amount of time it takes to find the right item for a customer.
- Recommender systems can help businesses learn more about their customers' preferences.

There are different types of RS. Some of them are listed below:

- Content-based recommendation: The most common type of recommender system is the content-based recommender system. A content-based recommender system is a type of recommender system that relies on the similarity between items to make recommendations. For example, if you're looking for a new movie to watch, a content-based recommender system might recommend movies that are similar to ones you've watched in the past. If you have watched thriller, then it is almost certain that you will be suggested other thriller movies. Content-based recommender systems are commonly used in music, books, and movies. They can be used to recommend products, services, or even websites. Content-based recommender systems are based on the idea that if you like one item, you're likely to like other items that are similar to it. To build a content-based recommender system, you need to first define what similarity means. This is where machine learning comes in. A content-based recommender system can use machine learning to learn the similarity between items. Once it has learned the similarity between items, it can make recommendations accordingly. Content-based recommender systems focus on the attributes of items in order to make recommendations. This

can often lead to issues with scalability, as the system needs to be constantly updated with new content in order to make accurate recommendations. Collaborative filtering systems, on the other hand, focus on the relationships between users and items. This can often result in problems with sparsity, as it can be difficult to find enough users who have rated a given item. The hybrid approach seeks to overcome these limitations by combining the two approaches.

- Collaborative filtering based recommendation: Another type of recommender system is the collaborative filtering recommender system. A collaborative filtering recommender system is a type of machine learning algorithm that makes predictions about what a user might want to buy or watch based on the past behavior of other users. It explores similarity between users and/or items. The algorithm looks at the items that other users with similar taste have purchased or rated highly, and recommends those items to the new user. The main advantage of collaborative filtering is that it doesn't require any information about the users or items; all it needs is a dataset of past user behavior. Collaborative filtering is one of the most popular techniques for building recommender systems, and is used by major companies such as Amazon, Netflix, and Spotify.

- Hybrid recommender system: A third type of recommender system is the hybrid recommender system. The hybrid approach has become increasingly popular in recent years as it offers the potential to overcome some of the limitations of each individual approach. A hybrid recommender system is a type of recommender system that combines both content-based and collaborative filtering approaches. The hybrid approach takes advantage of both content-based and collaborative filtering by using them to supplement each other. For example, a hybrid recommender system might first identify a set of items that are similar to the item the user is interested in, and then use collaborative filtering to identify which of those items the user is most likely to enjoy. This approach can provide more accurate recommendations than either method used alone. The hybrid approach has been shown to be more effective than either method used alone, as it is able to leverage the strengths of both approaches. Hybrid recommender systems are often more scalable and efficient than pure content-based or collaborative filtering systems.

Amazon's recommender system is based on a combination of collaborative filtering and content-based algorithms. It uses past customer behavior to make recommendations for new products. Amazon's recommender system is one of the most complex and sophisticated in the world. Netflix's recommender system is also based on a combination of collaborative filtering and content-based algorithms. However, Netflix takes things a step further by also incorporating machine learning into its algorithm. This allows Netflix to make predictions about what a user might want to watch based on the behavior of other users. Spotify's recommender system is based on collaborative filtering. It uses past user behavior to make recommendations for new songs to listen to.

In short, recommender systems are a type of machine learning based systems that are used to predict the ratings or preferences of items for a given user. There are three main types of Recommender Systems: collaborative filtering, content-based, and hybrid. Some of the most

popular examples of Recommender Systems include the ones used by Amazon, Netflix, and Spotify. Collaborative filtering systems use past user behavior to make recommendations for new products. Content-based systems focus on the attributes of items in order to make recommendations. The hybrid approach is a combination of both content-based and collaborative filtering approaches. Hybrid recommender systems are often more scalable and efficient than pure content-based or collaborative filtering systems.

In the most basic setup, the RS is made up of five core components: as shown in Fig. below.

1. Data Collection and Processing
2. Recommender Model
3. Recommendation post-processing
4. Online modules (tells us what needs to be stored in logs in order to both report how your system is performing and perhaps to also learn from the usage and interactions)
5. User interface (For example, it's good practice to explain to users why they are being recommended an item (e.g. 'You may like watching this movie because you liked movies X and Y), as this makes the recommender's decisions more transparent.)

C.1 Matrix Factorization

In collaborative filtering, matrix factorization is the state-of-the-art solution for sparse data problem, it is also known as 'Netflix Prize Challenge'.

For this neighborhood models are the most common approach.

Matrix Factorization are part of *latent factor models*. These compress user-item matrix into a low-dimensional representation in terms of latent factors. One advantage of using this approach is that instead of having a high dimensional matrix containing abundant number of missing values we will be dealing with a much smaller matrix in lower-dimensional space. A reduced presentation could be utilized for either user-based or item-based neighborhood algorithms that are presented in the previous section. There are several advantages with this paradigm. It handles the sparsity of the original matrix better than memory based ones. Also comparing similarity on the resulting matrix is much more scalable especially in dealing with large sparse datasets.

We can use a popular latent factor model named Singular Value Decomposition (SVD). There are other matrix factorization frameworks more specific to CF. An important decision is the number of factors to factor the user-item matrix. The higher the number of factors, the more precise is the factorization in the original matrix reconstructions. Therefore, if the model is allowed to memorize too much details of the original matrix, it may not generalize well for data it was not trained on. Reducing the number of factors increases the model generalization.

C.1.1 Dithering

There is a relatively low cost but high value technique for improving the quality of our recommendations known as 'dithering'. It's a technique that re-orders a list of recommendations by slightly

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

Figure 4. —

shuffling them around based on some noise. It is typically implemented in the recommendation post-processing component.

Section D

Natural Language Processing (NLP), ChatGPT and all that..

NLP is a powerful machine learning technique that can help you create chatbots, analyze sentiment, summarize text, classify data, and recognize speech.

D.1 TF, IDF, TF-IDF

TF-IDF stands for term frequency-inverse document frequency and it is a measure, used in the fields of information retrieval (IR) and machine learning, that can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).

Let us take an example of two documents: A: 'The car is driven on the road' and B: 'The truck is driven on the highway'. Then we have the following situation.

TF is just the ratio of number of times a word over the total words in the document. For example, in the document A above, car has TF of 1/7. Inverse document frequency (IDF) looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where t is the term (word) we are looking to measure the commonness of and N is the number of documents (d) in the corpus (D). The denominator is simply the number of documents in which the term, t, appears in.

$$IDF(t, D) = \log\left(\frac{d}{\text{Count}(t)}\right)$$

logits: Logits in deep learning refer to the vector of raw, unnormalized predictions generated by the last linear layer of a neural network before it is passed through a softmax activation function. Logits are used to compute the probabilities of the output classes through the softmax function. The higher the value of a logit for a particular class, the higher the probability of that class being the correct output.

D.2 Algorithms for NLP

The famous 'no-free lunch' says that there is no single algorithm that works for all cases. Same is true for NLP. But in particular, following algorithms are popular:

- Naive Bayes: a probabilistic algorithm that is often used for text classification tasks such as spam filtering
- Support Vector Machines (SVM): a popular algorithm for text classification tasks that involves finding a hyperplane that maximally separates the data points in the input space.
- Decision Trees: a simple yet effective algorithm that involves recursively splitting the data based on the most informative features until the desired classification accuracy is achieved.
- Random Forests: an ensemble learning method that combines multiple decision trees to improve classification accuracy.
- Gradient Boosting: a machine learning technique that involves combining multiple weak models to create a stronger model.
- Recurrent Neural Networks (RNN): a type of neural network that can model sequence data such as text by processing each word in a sentence one at a time. This includes LSTM.
- Convolutional Neural Networks (CNN): a type of neural network that is often used for image processing, but can also be applied to text data by treating the input as a two-dimensional grid of word embeddings.

The choice of algorithm depends on the specific task, the size and complexity of the dataset, and other factors such as computational resources and time constraints. Experimentation with different algorithms and hyperparameters is often necessary to determine the best approach for a given problem.

Unless you have been living in the mountains with no internet, you must have read or experimented with ChatGPT. In this post, and probably subsequent ones, we will try to explain what makes it tick and the foundations behind this progress. First some acronyms used:

- ML: Machine learning
- LLM: Large Language Model
- GPT: Generative Pre-training Transformer

- LSTM: Long-Short-Term-Memory

and Tranformer models

ChatGPT is an extrapolation of a class of ML models known as LLM. LLMs takes in huge quantities of text data and infer relationships between words in that text. The capability of LLMs increase with increase in size of their input datasets and parameter space.

In NLP, one is interested in making inferences from large data of text. The most basic and traditional method was S-to-S (sequence to sequence) models. However, when we have large amount of text this was not optimal since the model forgets the learning from the earlier parts as it moves along a sequence. This means that there is loss of information/learning. LSTM and GRU's did try to salvage this situation by discarding information that was not useful along the way to remember important information. Though, it still wasn't enough. Then in 2014, a paper introduced 'attention mechanisms'. The authors suggested assigning relative importance to each word in a sentence such that the model focuses on important words and ignores the rest (much like importance sampling in Monte Carlo methods). It emerged to be a massive improvement over encoder-decoder models.

The most basic training of language models involves predicting a word in a sequence of words. This is observed as either next-token-prediction (you must have noticed this in Gmail when composing an email when it suggest auto completion) and masked-language-modeling (where a sentence like 'Bill ??? running', is suggested completion through words as 'likes', 'hates' etc. ??? denotes the mask!). These basic-sequencing techniques are often implemented through LSTM. However, it has some drawbacks!

While running might be associated to 'hate', it might be that Bill is a marathon runner and actually loves it. There might be other sentences in the text which affirms this like 'Bill runs thrice a day'. However, if it is masked, LSTM might actually assign 'hate' to complete the sentence as: 'Bill hates running'. This means that the model is unable to give more importance to some surrounding words than others. The other issue is that the input data is processed individually and sequentially rather than as a whole big chunk. This means that when an LSTM is trained, the window of context is fixed. This limits the complexity of the relationships between words and the meanings that can be derived. So, suppose we have two sentences – 'Stochastic gradient is a very powerful tool in machine learning' , and 'Stochastic gradient which is related to conjugate gradient is a method frequently used in inverting fermionic operators in lattice field theory'. Now, if we train the model, and ask 'What is stochastic gradient?'. It will not say anything about lattice field theory since it doesn't see the whole corpus of data.

Because of these limitations, in 2017, a paper titled 'Attention is all you need' was posted which introduced the idea of 'transformers'. The title is named because the central idea of this architecture is to rely on attention (or, more precisely, self-attention) instead of the feedback loop found in RNNs. Remember that LSTM networks are a type of RNN that uses special units in addition to standard units. Unlike LSTMs, transformers can process all input data at once! Using self-attention mechanism, the model can give different weights to different parts of the input data in relation to any position of the language sequence. This feature enabled massive improvements

in putting meaning into LLMs and enables processing of significantly larger datasets and tackle dependencies between them.

All GPT models make use of the transformer architecture, which means they have an encoder to process the input sequence and a decoder to generate the output sequence. Both the encoder and decoder have a multi-head (The 'multi-head' attention is an evolution of self-attention) attention mechanism that allows the model to differentially weight parts of the sequence to infer meaning and context. In addition, the encoder leverages masked-language-modeling to understand the relationship between words and produce more comprehensible responses. The self-attention mechanism that drives GPT works by converting tokens (pieces of text, which can be a word, sentence, or other grouping of text) into vectors that represent the importance of the token in the input sequence.

The text below was generated by Chat GPT-3 with the prompts: Write notes about Large language models (LLM) in LaTeX, Explain Transformer LLM, What is the Maths behind Transformer LLM and so on. It even made a schematic diagram (in TikZ) for me on instruction which was not very ordered and black and white. It would be interesting to try this with GPT-4.

Large Language Models (LLMs)

Large Language Models (LLMs) are machine learning models that use deep neural networks to learn the statistical patterns and structure of natural language data. LLMs are designed to generate text that is similar in style, tone, and content to human language, and can be used for a wide range of natural language processing tasks.

Architecture

LLMs are typically based on a type of neural network known as a transformer, which is designed to process sequential data such as text. The transformer network is composed of multiple layers of self-attention mechanisms, which allow the model to attend to different parts of the input sequence and learn the relationships between them.

Training Data

LLMs are trained on large amounts of natural language data, typically in the form of text corpora such as books, articles, and web pages. The training data is preprocessed to remove noise, and then fed into the model in batches. The model is trained to predict the next word in a sequence of words, given the previous words as context.

Applications

LLMs have a wide range of applications in natural language processing, including but not limited to:

- Text generation and summarization
- Language translation
- Question-answering systems
- Sentiment analysis
- Named entity recognition
- Text classification
- LLMs are particularly useful in tasks where the input data is unstructured and the output is expected to be in natural language form.

Challenges

LLMs also present several challenges, including:

- Computational complexity: LLMs require large amounts of computational power to train and run, making them expensive to use.
- Bias: LLMs can reflect the biases and prejudices present in the training data, leading to unfair or inaccurate results.
- Privacy: LLMs can be used to generate text that appears to be written by a specific person, leading to potential privacy concerns.

Future Directions

LLMs are an active area of research, with ongoing efforts to improve their accuracy, efficiency, and fairness. Researchers are also exploring new applications of LLMs, such as generating creative writing, generating code, and creating interactive conversational agents.

Fine-tuning

LLMs can be fine-tuned on specific tasks by further training the model on a smaller dataset that is specific to that task. This allows the model to adapt to the specific language and patterns of the task, leading to improved performance.

Pretrained Models

Pretrained LLMs, such as OpenAI's GPT series and Google's BERT, are available for general use and can be fine-tuned for specific tasks. These models have been trained on massive amounts of data, making them capable of generating high-quality text in a wide range of contexts.

Ethical Considerations

LLMs raise important ethical considerations, particularly with regard to bias and privacy. Because LLMs learn from large datasets that may contain biases and prejudices, the output of the model may reflect these biases, leading to unfair or discriminatory results. Additionally, LLMs can be used to generate text that appears to be written by a specific person, leading to potential privacy concerns.

Interpretability

LLMs can be difficult to interpret, as the internal workings of the model are complex and opaque. Researchers are exploring ways to make LLMs more transparent and interpretable, such as through attention visualization and feature attribution techniques.

Open Research

LLMs are the subject of active research, with ongoing efforts to improve their accuracy, efficiency, and fairness. Many LLMs are open-source, allowing researchers and developers to build upon existing models and advance the field.

Multilingual LLMs

Multilingual LLMs are models that have been trained on multiple languages, allowing them to generate text in different languages and perform cross-lingual tasks. Multilingual LLMs can be useful in multilingual societies and in tasks that involve multiple languages.

Domain-Specific LLMs

Domain-specific LLMs are models that have been trained on text data from a specific domain, such as legal documents or scientific papers. These models are designed to generate text that is specific to the domain and can be useful in tasks that require specialized knowledge.

Combining LLMs with other NLP Techniques

LLMs can be combined with other natural language processing (NLP) techniques, such as sentiment analysis and named entity recognition, to create more complex and powerful NLP models. This approach is known as neural architecture search and can lead to significant improvements in performance.

Limitations

LLMs have limitations, particularly in tasks that require more than just statistical patterns in language. LLMs struggle with tasks that require common sense reasoning or knowledge that is not explicitly present in the training data. Additionally, LLMs can produce text that is fluent but not necessarily coherent, leading to potential issues in certain applications.

Impact on Society

LLMs have the potential to transform many aspects of society, including education, journalism, and entertainment. However, the widespread use of LLMs also raises important ethical considerations, such as the potential for misuse and the impact on employment. It is important for researchers and developers to consider the social and ethical implications of LLMs in their work.

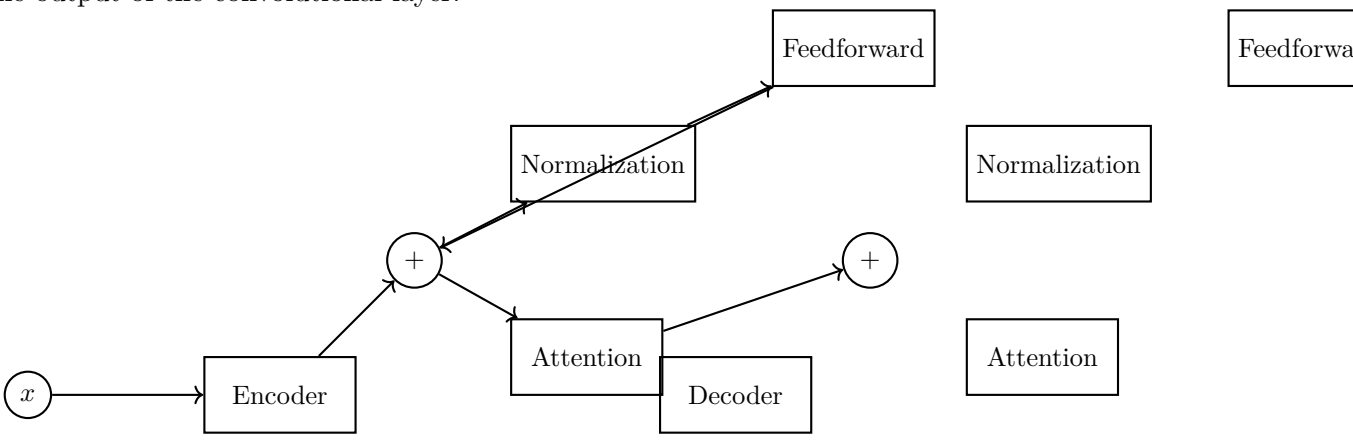
Section E

Details about Transformer LLM

Transformer LLM refers to a type of Large Language Model that is based on the Transformer architecture. The Transformer is a neural network architecture that was introduced by Vaswani et al. in 2017 and has been used extensively in natural language processing (NLP) tasks, including machine translation and language modeling. Transformer LLMs use a self-attention mechanism to process input sequences of variable length and produce output sequences of variable length. The self-attention mechanism allows the model to attend to different parts of the input sequence at each step of the computation, allowing it to capture long-range dependencies and improve performance on NLP tasks. One example of a Transformer LLM is the GPT (Generative Pre-trained Transformer) series developed by OpenAI. GPT models have been pre-trained on large

amounts of text data and can be fine-tuned for a variety of downstream NLP tasks, such as language generation, summarization, and question answering. Transformer LLMs have shown significant improvements in NLP tasks and have been used in a variety of applications, including chatbots, language translation, and sentiment analysis. Transformers are a type of deep learning model widely used in Natural Language processing tasks, such as machine translation and text summarization. One of the key characteristics of transformers is their use of attention mechanisms. Attention allows the model to focus on specific input parts, which is useful when processing long data sequences. Another characteristic of transformers is their ability to handle variable-length sequences in parallel. This is useful when dealing with natural language data, which often has varying lengths. We can train transformers to perform multiple tasks at once using a process known as multitask learning. In multitask learning, a single model is trained to perform multiple tasks simultaneously using a shared set of parameters. This allows the model to learn common features relevant to all tasks, which can improve the model's performance on each task.

Transformers **do not use** convolutional layers to process input data. Convolutional layers work by sliding a small window, or kernel, across the input data and performing an element-wise multiplication between the values in the kernel and the values in the input data. The resulting values are then summed and passed through a nonlinear activation function before being output as the output of the convolutional layer.



Training

Training a Transformer LLM involves pre-training the model on a large corpus of text data using an unsupervised learning approach, such as the masked language modeling task. During pre-training, the model learns to predict missing words in a sentence, which helps it to develop a general understanding of language. After pre-training, the Transformer LLM can be fine-tuned on specific downstream NLP tasks, using supervised learning techniques. Fine-tuning involves training the model on a smaller dataset that is specific to the task, which allows the model to adapt to the language and patterns of the task.

Architecture

The Transformer LLM architecture consists of an encoder and a decoder, each composed of multiple layers of self-attention and feedforward neural networks. The encoder processes the input sequence and produces a sequence of hidden states, while the decoder generates the output sequence based on the encoder's hidden states. The self-attention mechanism allows the model to learn which parts of the input sequence to attend to at each step of the computation. The feedforward neural networks apply a non-linear transformation to the hidden states, allowing the model to capture complex patterns in the input sequence.

Advantages

Transformer LLMs have several advantages over traditional recurrent neural network (RNN) based language models. They can process input sequences in parallel, making them faster and more efficient. They can also capture long-range dependencies in the input sequence, which is particularly useful for NLP tasks that require understanding of context. Additionally, Transformer LLMs can be fine-tuned on specific tasks with relatively small amounts of data, making them useful in practical applications.

Limitations

Transformer LLMs are not without limitations. They can be computationally expensive to train and require large amounts of data to achieve state-of-the-art performance. Additionally, they may struggle with tasks that require common sense reasoning or knowledge that is not explicitly present in the training data.

Future Directions

Research on Transformer LLMs is ongoing, with ongoing efforts to improve their performance, efficiency, and interpretability. Researchers are also exploring ways to incorporate external knowledge into LLMs, such as through external memory or attention mechanisms, in order to improve their ability to reason and understand context. The Transformer LLM is based on the Transformer architecture, which is a type of neural network that uses self-attention to process input sequences. Here's a brief overview of the mathematics involved in the Transformer LLM:

Self-Attention

At the heart of the Transformer LLM is the self-attention mechanism, which allows the model to attend to different parts of the input sequence at each step of the computation. Self-attention can be represented mathematically as follows:

Given an input sequence of length n, we first transform each input element into a vector of dimension d. These vectors are arranged into a matrix $X \in \mathbb{R}^{n \times d}$. We then compute three matrices: $Q \in \mathbb{R}^{n \times d}$, $K \in \mathbb{R}^{n \times d}$, and $V \in \mathbb{R}^{n \times d}$, which are called the query, key, and value matrices, respectively. The self-attention mechanism then computes a weighted sum of the value

vectors, where the weights are determined by the dot product between the query and key vectors, normalized by a scaling factor:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

here, softmax is the softmax function, which normalizes the dot product scores into a probability distribution over the input sequence.

Encoder

The Transformer LLM uses a stack of self-attention layers in the encoder, which allows the model to attend to different parts of the input sequence at different levels of abstraction. Each self-attention layer consists of two sub-layers:

A multi-head self-attention layer, which applies the self-attention mechanism described above to the input sequence, using different sets of query, key, and value matrices (known as "heads"). A position-wise feedforward layer, which applies a non-linear transformation to each element of the sequence independently. These sub-layers are connected through a residual connection and layer normalization, which helps to prevent vanishing gradients and improves training stability.

Decoder

The decoder in the Transformer LLM is similar to the encoder, but also includes an additional multi-head self-attention layer that attends to the output sequence generated so far. This allows the decoder to use context from the output sequence to generate the next element in the sequence.

Training

To train the Transformer LLM, we typically use a variant of stochastic gradient descent (SGD) to minimize a loss function that measures the discrepancy between the model's predictions and the true output. The loss function depends on the specific task and can be chosen based on the type of data and desired performance metric.

During training, the weights of the model are updated using backpropagation, which computes the gradient of the loss with respect to the model's parameters. The use of layer normalization and residual connections in the Transformer architecture helps to prevent vanishing gradients and improve training stability.

Section F

SQL

For someone starting with learning data science, SQL (pronounced 'sequel') can be bit tricky. There are two websites which are particularly nice to develop these skills:

Learn SQL

SQL Zoo

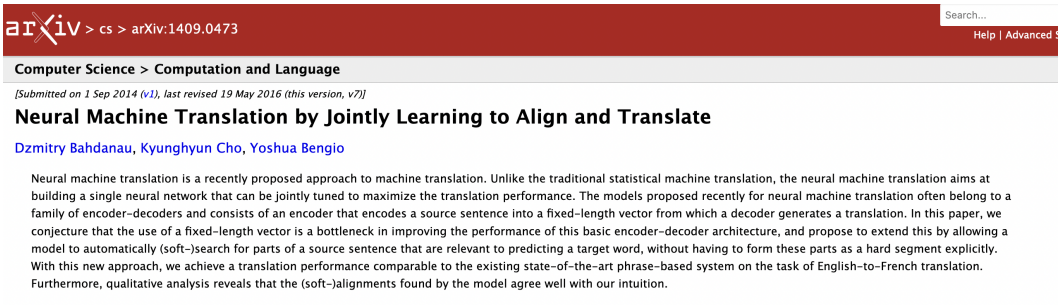


Figure 5. The paper which introduced 'attention mechanisms' in 2014.

Order is crucial: **Six Operations to Order: SFWGHO**

These are first letters (to remember) of SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY.

F.1 Main commands

- **SELECT**: The SELECT statement is used to retrieve data from one or more tables in a database. You should master using SELECT to filter, sort, and group data using different functions such as WHERE, ORDER BY, and GROUP BY. Here is an example of a SELECT statement:

```
SELECT column1, column2, column3 FROM table_name WHERE condition;
```

- In this example, column1, column2, and column3 are the names of the columns that you want to retrieve data from, and table_name is the name of the table that contains the data. The WHERE clause is optional but is used to specify a condition that must be met in order for the query to retrieve data. Here's an example that selects all records from a table called 'customers' where the customer's age is greater than or equal to 18:

```
SELECT *
FROM customers
WHERE age >= 18;
```

- **JOIN**: The JOIN statement is used to combine data from two or more tables in a database. You should master using JOIN to retrieve data from multiple tables and specify the type of join (e.g. INNER, LEFT, RIGHT, FULL OUTER) as appropriate. An INNER JOIN returns only the rows where there is a match between the columns in both tables. Here is an example:

```
SELECT orders.order_id, customers.customer_name
FROM orders
INNER JOIN customers
ON orders.customer_id = customers.customer_id;
```

In this example, the orders table and the customers table are joined using the customer_id column. The resulting table will only include the order_id and customer_name columns where there is a match between the customer_id columns in both tables.

- **LEFT JOIN**: A LEFT JOIN returns all the rows from the left table and the matching rows from the right table. If there is no match in the right table, the result will contain NULL values. Here is an example:

```
SELECT customers.customer_name, orders.order_id
FROM customers
LEFT JOIN orders
ON customers.customer_id = orders.customer_id;
```

  In this example, the customers table is the left table and the orders table is the right table. The customer_id column is used to join the tables. The resulting table will include all the rows from the customers table and the matching rows from the orders table. If there is no match in the orders table, the order_id column will contain NULL values.

- **RIGHT JOIN**: A RIGHT JOIN returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, the result will contain NULL values. Here is an example:

```
SELECT customers.customer_name, orders.order_id
FROM customers
RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

  In this example, the orders table is the left table and the customers table is the right table. The customer_id column is used to join the tables. The resulting table will include all the rows from the orders table and the matching rows from the customers table. If there is no match in the customers table, the customer_name column will contain NULL values.

- **OUTER JOIN**: An OUTER JOIN in SQL is used to return all the rows from one or both tables, including the non-matching rows. There are two types of OUTER JOINs: LEFT OUTER JOIN and RIGHT OUTER JOIN.

```
SELECT customers.customer_name, orders.order_id
FROM customers
LEFT OUTER JOIN orders
ON customers.customer_id = orders.customer_id;

* Now RIGHT OUTER JOIN

SELECT customers.customer_name, orders.order_id
```

```
FROM customers
RIGHT OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

In this example, the orders table is the left table and the customers table is the right table. The customer_id column is used to join the tables. The resulting table will include all the rows from the orders table and the matching rows from the customers table. If there is no match in the customers table, the customer_name column will contain NULL values. Some databases may not support RIGHT OUTER JOINs, but you can achieve the same result by using a LEFT OUTER JOIN and swapping the order of the tables.

- **WHERE**:

```
SELECT name, department, salary
FROM employees
WHERE department = 'Sales' AND salary > 50000;
```

- **GROUP BY**:

```
SELECT department, AVG(salary) as avg_salary
FROM employees
GROUP BY department;
```

- **HAVING**:

```
SELECT customer_id, SUM(quantity) AS total_quantity
FROM orders
GROUP BY customer_id
HAVING SUM(quantity) >= 50;
```

These are Window functions.

- **HAVING**:

```
ROW_NUMBER() \\
SUM()\\
RANK()\\
AVERAGE()\\
Basic query: SELECT column1, column2, ..., ####() OVER (ORDER BY column1)
AS row_num
FROM table_name;
```

Replace hashes by either four of these like RANK().

- **UNION**: Suppose we have two tables named "customers" and "employees", both with columns for "name" and "city". We want to create a list of all people (both customers

and employees) who live in New York City. We can use the UNION operator to combine the results of two SELECT statements, one for each table:

```
SELECT name, city
FROM customers
WHERE city = 'New York'
UNION
SELECT name, city
FROM employees
WHERE city = 'New York';
```

- **CREATE**: The CREATE statement is used to create a new database table, view, or other database objects. You should master using CREATE to create new tables, views, and other database objects. Here's an example of using the CREATE statement in SQL. Suppose we want to create a new table called "customers" with columns for "id", "name", "email", and "phone". We can use the CREATE statement to do this:

```
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100),
  phone VARCHAR(20)
);

INSERT INTO customers (id, name, email, phone)
VALUES (1, 'John Doe', 'johndoe@example.com', '555-555-1234');
```

- **INSERT**: As name suugests

```
INSERT INTO students (id, name, major, gpa)
VALUES (1234, 'John Doe', 'Computer Science', 3.5);
```

- **UPDATE**: As name suugests

```
UPDATE students
SET major = 'Mathematics', gpa = 3.7
WHERE id = 1234;
```

- **DELETE**: This query below would remove the row with an ID of 1234 from the "students" table. The DELETE statement specifies the name of the table we want to delete from, followed by the WHERE clause to specify which rows we want to delete. In this case, we want to delete the row with an ID of 1234, so we specify "WHERE id = 1234".

```
DELETE FROM students
WHERE id = 1234;
```

SQL is case insensitive.

- FIND SECOND HIGHEST SALARY: `SELECT MAX(SALARY) FROM Employee WHERE SALARY < (SELECT MAX(SALARY) FROM Employee);`
- NTH HIGHEST SALARY: `Select Salary from EMPLOYEE order by Salary DESC limit n-1,1;`
- `select * from city where population > 100000 and CountryCode="USA"`
- `select * from city where ID="1661"`
- `SELECT NAME FROM CITY WHERE COUNTRYCODE = 'JPN'`
- `SELECT * FROM Student WHERE FirstName NOT LIKE 'A%'` – The following query returns the rows in which FirstName do not start with A.
- `SELECT * FROM Student WHERE FirstName NOT LIKE '%d'` – The following query returns the rows in which FirstName do not end with d.
- `select distinct(city) from station where city not REGEXP '^[AEIOU]' AND city not REGEXP '[aeiou]$';` : Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.
- `select distinct(city) from station where city not REGEXP '^[AEIOU]' OR city not REGEXP '[aeiou]$';` : Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.
- `SELECT NAME FROM EMPLOYEE ORDER BY NAME ASC;`: Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.
- `SELECT NAME FROM EMPLOYEE WHERE SALARY > 2000 AND MONTHS < 10 ORDER BY EMPLOYEE_ID ASC`: Write a query that prints a list of employee names (i.e. the name attribute) for employees in Employee having a salary greater than 2000 per month who have been employees for less than 10 months. Sort your result by ascending employee-id.
- `ALTER TABLE celebs ADD COLUMN twitter_handle TEXT;`: Add new column
- `DELETE FROM celebs WHERE twitter_handle IS NULL;`: Delete rows where one specific column is empty
- `SELECT SUM(v) FROM ELEMENTS`: Sum Column v from table ELEMENTS
-
```
SELECT
  employee_id,
  last_name,
  first_name,
  salary,
```

```
  RANK() OVER (ORDER BY salary DESC) as ranking
FROM employee
ORDER BY ranking
```

In this example query, we will show a list of all employees ordered by salary (highest salary first). The report will include the position of each employee in the ranking. RANK is a function which arranges salary in DESC as ranking.

- ```
  WITH employee_ranking AS (
    SELECT
      employee_id,
      last_name,
      first_name,
      salary,
      RANK() OVER (ORDER BY salary DESC) as ranking
    FROM employee
  )
  SELECT
    employee_id,
    last_name,
    first_name,
    salary
  FROM employee_ranking
  WHERE ranking <= 5
  ORDER BY ranking
  ```

  Top 5 salary rankings.

-

- `CREATE TABLE`: creates a new table, `INSERT INTO`: adds a new row to a table, `SELECT` queries data from a table, `ALTER TABLE` changes an existing table, `UPDATE`: edits a row in a table, `DELETE FROM` deletes rows from a table.

Section G

Basic programming algorithms

There are different ways to sort a list. Some important ones can be remembered by mneumonic **ISBMQ**. Insertion, Selection, Bubble, Merge, Quick. We denote time and space complexity as $C(\cdot, \cdot)$

- Insertion: Start with second element of the list and place it on left or right of first element. Do this loop over elements similarly. $C(N^2, 1)$
  ```

- Selection: Find minimum of list and place it first (assuming ascending sort). Then focus on remaining and iterate. $C(N^2, 1)$
- Bubble: This works by repeatedly swapping the adjacent elements if they are in the wrong order. $C(N^2, 1)$
- Merge: Divide and conquer. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge(A, p, q, r) is a key process that assumes that A[p..q] and A[q+1..r] are sorted and merges the two sorted sub-arrays into one. $C(N \log N, N)$
- Quicksort: QuickSort is also a divide and conquer algorithm. It picks an element as a pivot (usually rightmost element) and partitions the given array around the picked pivot i.e., less than pivot moved to left, more than pivot to right and pivot at center. Then pick pivots of left and right sets and continue iterating. There are many different versions of quickSort that pick pivot in different ways. $N \log N, \log N$

Python uses Tim sort (which is a combination of both merge sort and insertion sort). $C(N \log N, N)$

Say Tim sort wants to sort `3, 1, 6, 4, 5, 7, 2`. We choose two blocks, run size=4. Left is 3,1,6,4 and right is 5,7,2. 3,1,6,4 -> 1,3,6,4 -> 1,3,4,6

Section H

Codility Exercises Solutions

Question 1: LongestPassword (see here)

Solution 1:

```
def solution(S):

    longest = -1
    n_letters = 0
    n_digits = 0
    n_others = 0

    for char in S:
        if char.isalpha():
            n_letters += 1
        elif char.isdigit():
            n_digits += 1
        elif char == " ":
            # Check whether it's a valid password.
            if n_others == 0 and n_letters % 2 == 0 and n_digits % 2 == 1:
                if longest < n_letters + n_digits:
                    longest = n_letters + n_digits

```

```
            # Reset the counters for the next word.
            n_letters = 0
            n_digits = 0
            n_others = 0

        else:
            n_others += 1

    # Check whether the last word is a valid password.

    if n_others == 0 and n_letters % 2 == 0 and n_digits % 2 == 1:
            if longest < n_letters + n_digits:
                longest = n_letters + n_digits

    return longest
```

Question 2: Binary Gap (see here)

Solution 2:

```
def solution(N):
    return len(max(format(N, 'b').strip('0').split('1')))
```

Section I

Interview questions

1. Q1: How to improve your model performance?
   Answer: Some methods are – use validation methods, add more data, apply feature engineering techniques (normalization, imputation etc.), compare multiple algorithms, hyperparameter tuning

2. Q2: What is difference between standardization and log transformation.
   Answer: Standardization is the process of putting different variables on the same scale. This process allows you to compare scores between different types of variables. Typically, to standardize variables, you calculate the mean and standard deviation for a variable. Log transformation is a data transformation method in which it replaces each variable x with a $log(x)$. Log-transform decreases skew in some distributions, especially with large outliers. But, it may not be useful as well if the original distribution is not skewed. Also, log transform may not be applied to some cases (negative values), but standardisation is mostly always applicable except $\sigma = 0$.

3. Q3: How to deal with unbalanced binary classification?
   Answer: In a classification task, there is a high chance for the algorithm to be biased if the dataset is imbalanced. An imbalanced dataset is one in which the number of samples in

one class is very higher or lesser than the number of samples in the other class. To counter such imbalanced datasets, we use a technique called up-sampling and down-sampling. In up-sampling, we randomly duplicate the observations from the minority class in order to reinforce its signal. The most common way is to resample with replacement. In down-sampling, we randomly remove the observations from the majority class. Thus after up-sampling or down-sampling, the dataset becomes balanced with the same number of observations in each class.

4. Q4: Pruning in Decision Tree?
   ANSWER: Decision Trees are a popular machine-learning model for classification and regression tasks. They build a tree-like decision model based on the data's features to predict the output label of a given input sample. However, if the Decision Tree is allowed to grow too large, it can become overly complex and may start to fit the training data too closely, a phenomenon known as overfitting.

   Pruning is a technique that can be used to reduce the size of a Decision Tree and prevent overfitting. It involves removing nodes from the tree that are not contributing significantly to the model's accuracy. By pruning a Decision Tree, the model's ability to generalize to new data is improved, as the model is less likely to be overly complex and overfit the training set. This is the primary purpose of pruning a Decision Tree.

   In addition to improving the model's ability to generalize to new data, pruning a Decision Tree reduces the complexity of the model and makes it easier to interpret. By removing unnecessary nodes from the Decision Tree, the model becomes simpler and more transparent, making it easier to understand and interpret.

   Pruning also reduces the number of nodes and decision points the model needs to consider during training, which can improve the model's training speed and efficiency.

   ALL QUESTIONS AND ANSWERS BELOW ARE FROM CHAT-GPT:

   - What is overfitting and how can it be avoided? Overfitting is a common problem in machine learning where a model is trained too well on the training data, and as a result, it fits the noise or random fluctuations in the data as well. This leads to poor performance of the model on the test data, where it fails to generalize well. Overfitting can be prevented by using techniques such as regularization, cross-validation, and early stopping.
   - What is the difference between supervised and unsupervised learning? Supervised learning is a type of machine learning where the model is trained on labeled data, where the input and output pairs are known. The goal of supervised learning is to learn a mapping function that can predict the output for new input data. Examples of supervised learning algorithms include linear regression, logistic regression, decision trees, and neural networks. Unsupervised learning is a type of machine learning where the model is trained on unlabeled data, where only the input data is known. The goal of unsuper-

vised learning is to discover patterns or structures in the data. Examples of unsupervised learning algorithms include clustering, dimensionality reduction, and generative models.

- What is the difference between regression and classification?
- What is regularization and why is it important? Regularization is a technique used to prevent overfitting in machine learning models. It involves adding a penalty term to the loss function of the model, which discourages the model from fitting the noise or random fluctuations in the data. The penalty term is typically a function of the model parameters, such as the L1 norm or L2 norm of the weight vectors. Regularization can be used in various machine learning models such as linear regression, logistic regression, and neural networks.
- What are some common machine learning algorithms?
- What is the difference between a decision tree and a random forest?
- What is cross-validation and why is it used? Cross-validation is a technique used to evaluate the performance of a machine learning model on a limited amount of data. It involves splitting the data into k equally sized folds, and then training the model on k-1 folds and evaluating it on the remaining fold. This process is repeated k times, with each fold being used once as the validation set. The performance of the model is then averaged across the k runs. Cross-validation is useful for selecting hyperparameters, comparing different models, and estimating the generalization performance of a model.
- What is the bias-variance tradeoff? The bias-variance tradeoff is a fundamental concept in machine learning that describes the relationship between the complexity of a model and its generalization performance. Bias refers to the error introduced by the model's assumptions or simplifications, while variance refers to the error introduced by the model's sensitivity to small fluctuations in the data. A model with high bias is said to underfit the data, while a model with high variance is said to overfit the data. The goal is to find a model that achieves a balance between bias and variance that minimizes the generalization error.
- What is gradient descent and how does it work?
- What is deep learning and how is it different from traditional machine learning?
- What is a convolutional neural network (CNN) and how is it used in image recognition? A convolutional neural network (CNN) is a type of neural network used in deep learning to process and analyze images. It uses convolutional layers that extract features from the input image, followed by pooling layers that reduce the dimensionality of the output, and finally fully connected layers that classify the image. CNNs are widely used in image recognition tasks such as object detection, face recognition, and image segmentation.
- What is the ROC curve and how is it used to evaluate a classification model? The ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model. It illustrates the trade-off between the

true positive rate (TPR) and false positive rate (FPR) for different threshold values. In a binary classification problem, we classify examples into two classes, such as "positive" and "negative". The ROC curve is obtained by plotting TPR (also called sensitivity or recall) on the y-axis and FPR on the x-axis.

To calculate TPR and FPR, we need to set a threshold that separates the positive and negative examples predicted by the model. A threshold of 0.5 is often used as a default, but this can be adjusted depending on the problem and the goals of the model.

The TPR is the proportion of true positive examples that are correctly classified as positive by the model, while the FPR is the proportion of negative examples that are incorrectly classified as positive. As the threshold is varied, different points on the ROC curve are generated.

A perfect classifier would have a ROC curve that passes through the top-left corner of the plot, with TPR = 1 and FPR = 0. In practice, most classifiers will have a ROC curve that lies somewhere below this ideal line.

The area under the ROC curve (AUC) is a common metric used to evaluate the performance of a binary classifier. AUC ranges from 0.0 to 1.0, with a higher value indicating better classification performance. An AUC of 0.5 indicates random classification performance, while an AUC of 1.0 indicates perfect classification performance.

In summary, the ROC curve provides a useful tool to visualize and evaluate the performance of a binary classification model, and the AUC provides a single scalar value to summarize the overall performance.

- What is natural language processing (NLP) and how is it used in machine learning?
- What is reinforcement learning and how is it used in AI?

Section J

Statistics

Central Limit theorem: If $X_1, X_2, \cdots X_N$ are random samples drawn from a distribution (any) with overall mean μ and variance σ^2 and if $\bar{X}$ is the sample mean of first n samples, then the limiting form of distribution is Gaussian in $n \to \infty$ limit.

Section K

Reading stuff

PinSage is a random-walk-based Graph Convolutional Networks (GCN) which operates on a massive graph with 3 billion nodes and 18 billion edges — a graph that is 10,000 times larger than typical applications of GCNs. GCN were introduced in [**?**]. This is used to create image embedding of billion of pins. It uses visual and textual features with graph information.

When training ItemSage, the PinSage model is used to provide the embeddings of both the feature images of the product and the query images, while the SearchSage model embeds the search query string.

Earlier approaches typically focus on building embeddings from content in a single modality, i.e. text or images. Product information spans both modalities. We introduce a transformer-based architecture capable of combining features from these different modalities which is different from earlier works.

There are three surfaces: Home (past activity), Close-up (similar to current view), Search (recommendations based on query string). They use single-stream of single-layer transformer model.

Argue that first work which uses the transformer idea with multi-modal features to learn for production level recommendations.

References

[1] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," *arXiv e-prints* (July, 2013) arXiv:1307.0411, arXiv:1307.0411 [quant-ph].

[2] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Yuezhen Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. Von Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, "TensorFlow Quantum: A Software Framework for Quantum Machine Learning," *arXiv e-prints* (Mar., 2020) arXiv:2003.02989, arXiv:2003.02989 [quant-ph].