MY JOURNEY INTO DATA SCIENCE AND MACHINE LEARNING

Library of Concepts and useful stuff

Abstract

In this document I compiled all the knowledge that I accumulated and learned through all kinds of content throughout the internet and some practices. I hope it proves useful in your adventure to get into this exciting science.

Jose Luis Ossio Bejarano

Joseluis.bejarano97@gmail.com

https://github.dev/jlob97/DataScience-Scripts

Contents

Fundamentals

1. Extract, Transform and Load (ETL)

Extract, transform, load (ETL) is a data integration process that collects data from various sources, transforms it according to business rules, and loads it into a target data store, such as a data warehouse. ETL provides the foundation for data analytics and machine learning workstreams.

ETL consists of three steps: extract, transform, and load. In the extract step, raw data is copied or exported from source locations to a staging area. The sources can be structured or unstructured data. In the transform step, the data is cleaned, organized, and modified in a specialized engine using staging tables. The transformation can involve filtering, sorting, aggregating, joining, cleaning, deduplicating, and validating data. In the load step, the transformed data is moved to a destination data store, where it can be accessed by users or applications.

ETL is often used by organizations to extract data from legacy systems, improve data quality and consistency, and load data into a centralized database for analysis. ETL can also handle more advanced analytics, such as monthly reporting or machine learning models, by shaping the data to meet specific business intelligence needs.

2. Useful Libraries for Data Analyzing

- Libraries for data analyzing:
- AWS Data Wrangler - Pandas on AWS.
- Blaze - NumPy and Pandas interface to Big Data.
- Open Mining - Business Intelligence (BI) in Pandas interface.
- Optimus - Agile Data Science Workflows made easy with PySpark.
- Orange - Data mining, data visualization, analysis and machine learning through visual programming or scripts.
- Pandas - A library providing high-performance, easy-to-use data structures and data analysis tools.

More libraries in GitHub: https://github.com/vinta/awesome-python

3. Useful Datasets (Awesome Public Datasets)

https://github.com/awesomedata/awesome-public-datasets

4. Principal Component Analysis (PCA)

Principal component analysis (PCA) is a statistical technique that reduces the dimensionality of a data set by transforming it into a new coordinate system. The new coordinates, called principal components, are linear combinations of the original variables that capture the maximum amount of variation in the data. PCA can be used for data compression, noise reduction, feature extraction, visualization, and clustering. PCA has applications in many fields, such as image processing, face recognition, natural language processing, bioinformatics, and finance.

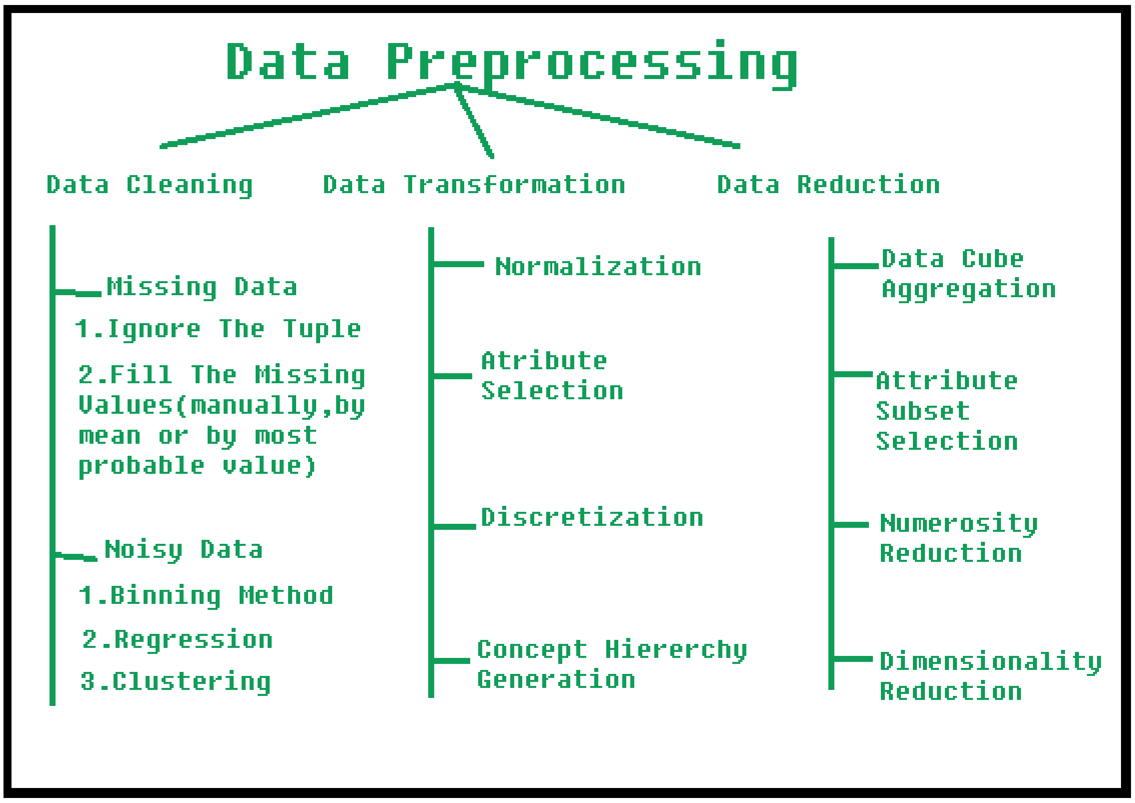
5. Dimensionality Reduction for Machine Learning

- Dimensionality reduction is the process of reducing the number of input variables or features in a dataset, which can improve the performance and efficiency of machine learning algorithms.
- Dimensionality reduction can help overcome the curse of dimensionality, which refers to the problems that arise when working with data in higher dimensions, such as overfitting, sparsity, complexity, and noise.
- Dimensionality reduction can be achieved by feature selection or feature engineering methods. Feature selection is the process of identifying and selecting relevant features for a specific task. Feature engineering is the process of creating new features from existing features by applying some transformation or operation on them.
- Dimensionality reduction techniques can be divided into two categories: linear and nonlinear. Linear techniques assume that the data lies on or near a lower-dimensional linear subspace. Nonlinear techniques can capture the intrinsic structure of the data that may not be linearly separable.
- Some common linear dimensionality reduction techniques are Principal Component Analysis (PCA, listed as point 4), Linear Discriminant Analysis (LDA), and Factor Analysis (FA). Some common nonlinear dimensionality reduction techniques are t-distributed Stochastic Neighbor Embedding (t-SNE), Isomap, and Locally Linear Embedding (LLE).

6. Data Preprocessing

- Data preprocessing is an essential step in data mining, as it prepares the raw data for analysis. Data preprocessing involves four main tasks: data cleaning, data integration, data transformation, and data reduction.
- Data cleaning is the process of identifying and removing missing, inconsistent, or irrelevant data from the dataset. This can improve the accuracy and efficiency of the data mining algorithms. Data cleaning techniques include filling in missing values, removing duplicate records, and handling outliers.
- Data integration is the process of combining data from multiple sources into a single, consistent view. This can provide a more comprehensive and diverse perspective on the data mining problem. Data integration techniques include schema integration, entity identification, and data fusion.
- Data transformation is the process of converting the data into a format that is more suitable for the data mining task. This can enhance the performance and interpretability of the data mining results. Data transformation techniques include normalizing numerical data, encoding categorical data, and creating dummy variables.
- Data reduction is the process of selecting a subset of the data that is relevant and representative for the data mining task. This can reduce the complexity and dimensionality of the data, and speed up the data mining process. Data reduction techniques include feature selection, feature extraction, and data discretization.

More info here: https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/



7. **Numerosity Reduction**

Numerosity reduction is a technique used in data mining to reduce the number of data points in a dataset while still preserving the most important information. It can help to improve the efficiency and performance of machine learning algorithms by reducing the size and complexity of the data. There are two main types of numerosity reduction methods: parametric and non-parametric. Parametric methods assume that the data fits a certain model, such as a linear regression or a log-linear model, and store only the parameters of the model instead of the actual data. Non-parametric methods do not assume any model and store a reduced representation of the data, such as a sample, a cluster, or a histogram. Both types of methods have advantages and disadvantages, depending on the characteristics and goals of the data analysis.

More info here: https://www.geeksforgeeks.org/numerosity-reduction-in-data-mining/

8. **Normalization**

Normalization (statistics) is a process of adjusting values measured on different scales to a common scale, often before averaging or comparing them. Normalization can also refer to more complex adjustments where the goal is to make the probability distributions of different variables more similar or aligned. Normalization can help to eliminate the effects of some gross influences and make the data more suitable for statistical analysis.

There are different types of normalization techniques in statistics, such as:

- Standard score: This is a way of normalizing errors when population parameters are known. It is calculated by subtracting the mean from the value and dividing by the standard deviation. This works well for populations that are normally distributed.

- Student's t-statistic: This is a way of measuring how far an estimated parameter is from its hypothesized value, normalized by its standard error. It is used for hypothesis testing and confidence intervals.

- Studentized residual: This is a way of normalizing residuals when parameters are estimated, especially in regression analysis. It is calculated by dividing the residual by an estimate of its standard deviation.

- Feature scaling: This is a way of rescaling data to have values between 0 and 1, or between -1 and 1. It can be done by dividing by the maximum value, or by subtracting the minimum value and dividing by the range.

- Quantile normalization: This is a way of making the quantiles of different variables match. It can be done by sorting the values, averaging across variables, and replacing the original values with the averages.

Normalization can improve the performance and stability of machine learning models, as well as make the data more interpretable and comparable.

9. "Ultimate" Guide to Cleaning

Complete guide here: https://towardsdatascience.com/the-ultimate-guide-to-data-cleaning-3969843991d4

A guide made by a Data Cleaning Specialist with 4+ years of experience in applying data analysis and cleaning techniques to various datasets. Skilled in using Python, SQL, Excel and other tools to perform data cleaning tasks such as handling missing values, outliers, duplicates, formatting errors and inconsistencies.

10. Based vs Unbiased

I just better citate this: Biased vs Unbiased: Debunking Statistical Myths - DataScienceCentral.com

11. Feature Engineering – Data Science

Read this: https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b

12. Feature Engineering – Machine Learning

Same but for ML: https://medium.com/@mehulved1503/feature-selection-and-feature-extraction-in-machine-learning-an-overview-57891c595e96

13. Dealing with Noisy Data

The resource I used was deleted years ago, I was able to get a copy so I am leaving it on my GitHub, hope you can read it.

To deal with noisy data in data science. Noisy data refers to data that contains errors, outliers, missing values, or inconsistencies that can affect the quality and reliability of the analysis. The author suggests some methods to identify and handle noisy data, such as:

- Exploratory data analysis: using descriptive statistics and visualization techniques to understand the distribution and characteristics of the data.

- Data cleaning: applying techniques such as imputation, interpolation, filtering, or transformation to correct or remove noisy data.

- Data validation: using rules or constraints to check the validity and accuracy of the data.

- Data augmentation: using techniques such as oversampling, undersampling, or synthetic data generation to increase the diversity and balance of the data.

Don’t underestimate importance of domain knowledge and business understanding in dealing with noisy data, as different scenarios may require different approaches and solutions. The author concludes by stating that noisy data is inevitable in real-world data science projects, and that it is essential to have a systematic and robust process to handle it effectively.

14. Sampling (Statistics not ML)

Sampling is a statistical procedure that involves selecting a subset of individuals or cases from a population to estimate its characteristics. Sampling is used when it is impractical or infeasible to measure the entire population. Sampling can be based on probability theory or other methods. Sampling can provide insights, estimates, and inferences about the population distribution.

There are two primary types of sampling methods: probability sampling and non-probability sampling. Probability sampling involves random selection, allowing for strong statistical inferences about the whole population. Non-probability sampling involves non-random selection based on convenience or other criteria, allowing for easy data collection. However, non-probability sampling may introduce bias and limit generalizability.

Some common examples of probability sampling methods are simple random sampling, stratified random sampling, cluster random sampling, and systematic random sampling. Some common examples of non-probability sampling methods are convenience sampling, voluntary response sampling, quota sampling, and snowball sampling.

The choice of sampling method depends on various factors, such as the size and variability of the population, the research design, the research question, the available resources, and the desired level of precision and accuracy.

15. Theory of Probability:

General concept: https://seeing-theory.brown.edu/index.html

Concepts to Google:

- Randomness, random variable, random sample.
- Probability Distribution
- Conditional Probability and Bayes Theorem
- Statistical Independence

- Iid (https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables)
- Cumulative distribution function (cdf)
- Probability Density function (pdf)[Continuous distributions]

Normal/Gaussian, Uniform(Continuous), Beta distribution, Dirichet, Exponential, X^2(Chi Squared)

- Probability mass function (pmf)[Discrete distributions]

Uniform (Discrete), Binomial, Multinomial, Hypergeometric, Poisson, Geometric

- Summary Statistics

Expectation and mean, variance and standar deviation (sd), Covariance and Correlation, median and quartile, interquartile range, Percentile/Quantile, Mode

- Important Laws to Know

LLN or Law of large numbers and CLT Central Limit Theorem

- Estimation. - MLE Maximum Likelihood Estimation and KDE Kernel Density Estimation
- Hypothesis testing: P-Value, Chi2 Test, F-test and t-test
  Also note for the ANOVA, MANOVA, ANVOCA AND MANCOVA. Not a test but also useful in this topic.
- Confidence Interval (CI)
- THE MONTE CARLO METHOD!

16. For visualizations:

Starter Kit to choose. - https://extremepresentation.typepad.com/files/choosing-a-good-chart-09.pdf

Python. - Matplotlib, Plotnine, Bokeh (not as used), seaborn, ipyvolume(3D)

Reference for Plotnine: https://medium.com/@gscheithauer/data-visualization-in-python-like-in-rs-ggplot2-bc62f8debbf5 (I like R)

Dashboards and BI. - Tableau and PowerBI also Dash (I need more info about Dash)

If you have free time, check this GitHub: https://github.com/streamlit/streamlit

Machine Learning

Useful Sources:

https://github.com/EthicalML/awesome-production-machine-learning

Don't mislead with Deep Learning

17. **Types of Variables:**

- Categorical Variables, Ordinal Variables, Numerical Variables

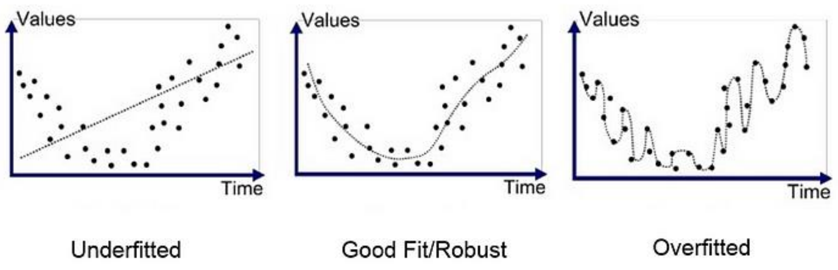
This is Basic, should be easily understandable.

18. **Cost functions and gradient descent (In linear regression)**

Cost functions are mathematical functions that measure how well a model fits the data. They quantify the error or loss of a model by comparing the predicted values with the actual values. One example of cost function is the mean squared error (MSE) cost function, which is commonly used for regression problems. The MSE calculates the average of the squared differences between the predicted values and the actual values.

Gradient descent is an optimization algorithm that finds the optimal values of the model parameters that minimize the cost function. Gradient descent iteratively updates the parameters by moving in the opposite direction of the gradient (the slope) of the cost function at each point. It is easily understandable how gradient descent works visually using a contour plot and a surface plot of a cost function.

19. **Overfitting / Underfitting**



Overfitting occurs when a model learns too much from the training data and fails to generalize well to new data. Underfitting occurs when a model learns too little from the training data and performs poorly on both training and testing data. It is all about finding the right balance between model complexity and data size.

Some concepts:

Overfitting: too much reliance on the training data

Underfitting: a failure to learn the relationships in the training data

High Variance: model changes significantly based on training data

High Bias: assumptions about model lead to ignoring training data

Overfitting and underfitting cause poor generalization on the test set

A validation set for model tuning can prevent under and overfitting

20. Training, Validation and test data

Training, validation, and test sets are different subsets of data that are used to build and evaluate machine learning models. The training set is used to fit the model parameters, such as the weights of a neural network. The validation set is used to tune the model hyperparameters, such as the learning rate or the number of hidden units. The test set is used to measure the generalization performance of the final model on unseen data. The data sets should be representative of the problem domain and independent of each other. Using separate data sets for training, validation, and testing helps to avoid overfitting and underfitting, and to ensure that the model can make accurate predictions on new data.

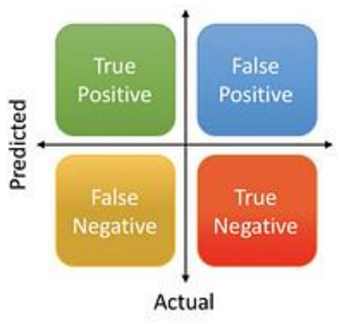
21. Precision, Recall, Accuracy and F1 score

I will introduce first this graph to:

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



These are metrics that are going to have different importance in regard of the topics or the importance we give to each one of the 4 situations. Per example, if you take the situation of classification of spam messages, is harmful to calcsilicate a non-spam message as spam… but is as harmful that a spam just passed the filter. In this case, you need to think which metrics will help us determine the effectivity of our model.

The issue…

Recall and precision are important metrics to evaluate machine learning models, especially for classification problems. They measure how well a model can identify relevant and correct results from a given data set. However, there is often a trade-off between recall and precision, meaning that improving one may lower the other. For example, in a digital world where customers have limited attention and space, showing too many irrelevant results (high recall, low precision) may cause them to lose interest and switch to other websites. Therefore, a model needs to balance recall and precision to achieve optimal performance.

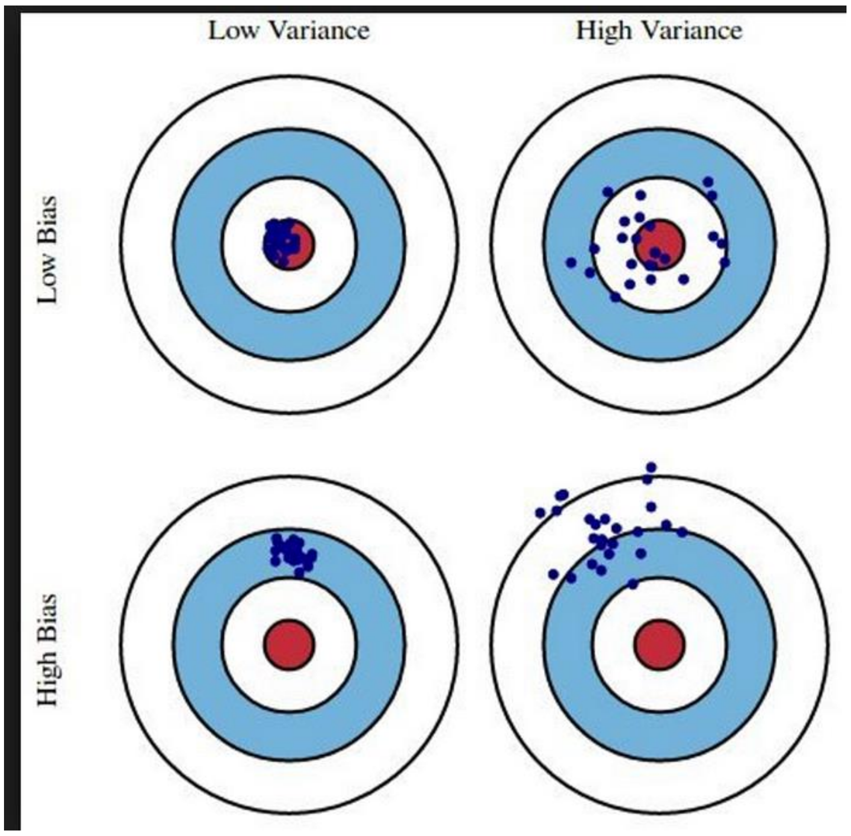
The "solution" …

It introduces F1-score as a harmonic mean of the two, and argue that it is easier and more convenient to work with. It also possible to mention other possible ways to combine precision and recall, such as a geometric mean but that is something that you need to take in consideration.

$$\textbf{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

22. Bias vs Variance

Source: https://becominghuman.ai/machine-learning-bias-vs-variance-641f924e6c57



Bias and variance are two key concepts in machine learning that affect the performance and accuracy of a model. In this post, we will explain what bias and variance are, how they relate to each other, and how to reduce them.

Bias is the difference between the average prediction of a model and the true value of the data. A model with high bias makes strong assumptions about the data and fails to capture its complexity. This leads to underfitting, which means the model performs poorly on both training and testing data.

Variance is the variation in the predictions of a model for different data sets. A model with high variance is sensitive to noise and overfits to the training data. This means the model performs well on training data but poorly on new or unseen data.

Bias and variance are inversely related, meaning that reducing one usually increases the other. This is known as the bias-variance tradeoff, which is a fundamental challenge in machine learning. The goal is to find a balance between bias and variance that minimizes the total error of a model.

There are various techniques to reduce bias and variance in machine learning, such as:

- Choosing an appropriate model complexity that matches the data
- Using regularization methods that penalize complex models
- Using cross-validation methods that evaluate model performance on different subsets of data
- Using ensemble methods that combine multiple models to reduce variance
- Using feature selection methods that reduce irrelevant or redundant features
- Using feature engineering methods that create new or transform existing features

23. Lift (data mining)

Lift is a concept in data mining that measures how well a targeting model can identify cases with a positive response, compared to a random selection of cases. Lift is calculated as the ratio of the response rate within a target group to the average response rate for the whole population. For example, if a model predicts that 20% of customers who receive a certain offer will buy a product, and the average response rate for all customers is 5%, then the lift is 20%/5% = 4. This means that the model is four times more effective than random targeting. Lift can be used to evaluate and compare different models or rules, and to rank segments of customers by their profitability. Lift can also be visualized by a lift curve, which shows the cumulative lift achieved by selecting different proportions of the population.

Let’s talk about Methods in ML

Supervised, Unsupervised, Ensemble and Reinforcement Learning.

24. Supervised Learning

Here we have Regression models like Linear and Poisson Regression.

Linear Regression

Linear regression is a statistical method that allows us to model the relationship between a dependent variable and one or more independent variables. It is based on the assumption that there is a linear correlation between both variables. In this post, we will explain how linear regression works, what are its main applications, and how to perform it using Python.

Linear regression can be used to analyze how a change in one variable affects another variable. For example, we can use linear regression to study how the price of a product influences its sales, or how the amount of rainfall affects the crop yield. Linear regression can also be used to make predictions based on existing data. For example, we can use linear regression to estimate the future sales of a product based on its past performance, or to forecast the weather based on historical data.

To perform linear regression, we need to find the equation of the line that best fits our data. This line is called the regression line, and it has the form $y = mx + b$, where y is the dependent variable, x is the independent variable, m is the slope of the line, and b is the intercept of the line. The slope m tells us how much y changes for every unit change in x, and the intercept b tells us the value of y when x is zero.

There are different methods to find the best-fitting line for our data, such as ordinary least squares (OLS), maximum likelihood estimation (MLE), or gradient descent. These methods aim to minimize the sum of squared errors (SSE) between the observed values of y and the predicted values of y by the regression line. The smaller the SSE, the better the fit of the line.

Source: https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

Poisson Regression

Poisson regression is a type of statistical analysis that can be used to model count data, such as the number of events that occur in a given time or space. Poisson regression assumes that the response variable follows a Poisson distribution, which means that the mean and the variance are equal. Poisson regression also assumes that the logarithm of the expected value of the response variable can be expressed as a linear function of the predictor variables.

One example of count data that can be modeled using Poisson regression is the number of traffic accidents at a particular intersection based on weather conditions and whether or not a special event is taking place in the city. To perform Poisson regression on this data, we would use the following model:

$\log(E(Y)) = \beta 0 + \beta 1X1 + \beta 2X2 + \beta 3X3$

where Y is the number of traffic accidents, X1 is a dummy variable for weather conditions (0 for sunny, 1 for cloudy, 2 for rainy), X2 is a dummy variable for special event (0 for no, 1 for yes), and X3 is an offset term that accounts for the exposure time (the number of hours that the intersection is observed). The coefficients β0, β1, β2, and β3 are estimated using maximum likelihood methods.

The interpretation of the coefficients in Poisson regression is similar to that of logistic regression, except that we use exponentiation instead of odds ratios. For example, exp(β1) is the multiplicative effect of weather conditions on the expected number of traffic accidents, holding other variables constant. If exp(β1) = 1.2, it means that cloudy weather increases the expected number of traffic accidents by 20% compared to sunny weather, and rainy weather increases it by 44% compared to sunny weather.

Poisson regression has some limitations and assumptions that need to be checked before applying it to count data. One of them is the equidispersion assumption, which means that the mean and the variance of the response variable are equal. If this assumption is violated, it means that there is overdispersion or underdispersion in the data, which can lead to biased or inefficient estimates. One way to deal with overdispersion or underdispersion is to use a negative binomial regression model, which allows for a different mean and variance for the response variable.

Classification models

There is a PDF in my GitHub, feel free to read it!

Classification Rate

- WIP

Decision Trees

A decision tree is a type of supervised learning algorithm that can be used for both classification and regression problems in machine learning. It is a graphical representation of the possible outcomes of a decision based on certain conditions. A decision tree consists of a root node, which represents the entire dataset, internal nodes, which represent the features or attributes of the dataset, and leaf nodes, which represent the final outcomes or classes. The internal nodes are connected by branches, which represent the decision rules or criteria for splitting the data. The decision tree algorithm works by recursively partitioning the data into smaller and more homogeneous subsets based on the values of the features. The algorithm stops when there are no more features to split on, or when a predefined criterion is met.

Source: https://hackernoon.com/what-is-a-decision-tree-in-machine-learning-15ce51dc445d

Logistic Regression

Logistic regression is a statistical model that estimates the probability of an event occurring based on one or more independent variables. It is often used for classification and predictive analytics in various fields, such as machine learning, medicine, and social sciences.

To understand logistic regression, we need to know some basic concepts:

- The dependent variable is the outcome we want to predict, such as whether a customer will default on a loan or not. It is usually binary (0 or 1) or categorical (e.g., cat, dog, lion).

- The independent variables are the features or predictors that influence the outcome, such as credit score and bank balance. They can be binary, categorical, or continuous (any real value).

- The odds are the ratio of the probability of success to the probability of failure. For example, if the probability of defaulting on a loan is 0.2, then the odds are 0.2 / (1 - 0.2) = 0.25.

- The logit is the natural logarithm of the odds. For example, if the odds are 0.25, then the logit is ln(0.25) = -1.386.

- The logistic function is the inverse of the logit function. It converts a logit value to a probability value between 0 and 1. For example, if the logit is -1.386, then the logistic function is exp(-1.386) / (1 + exp(-1.386)) = 0.2.

The logistic regression model assumes that there is a linear relationship between the logit of the dependent variable and the independent variables. That is,

logit(p) = b0 + b1x1 + b2x2 + ... + bkxk

where p is the probability of success, b0 is the intercept, b1 to bk are the coefficients, and x1 to xk are the independent variables.

The coefficients are estimated using a method called maximum likelihood estimation (MLE), which tries to find the values that maximize the likelihood of observing the data.

Once we have the coefficients, we can plug in any values of the independent variables and calculate the predicted probability of success using the logistic function.

For example, suppose we have a logistic regression model that predicts whether a customer will default on a loan based on their credit score and bank balance. The coefficients are:

b0 = -3

b1 = 0.01

b2 = -0.001

If a customer has a credit score of 600 and a bank balance of $10,000, then their predicted probability of defaulting is:

logit(p) = -3 + 0.01 * 600 - 0.001 * 10000 = -2

p = exp(-2) / (1 + exp(-2)) = 0.119

This means that there is a 11.9% chance that this customer will default on their loan.

To evaluate how well the logistic regression model fits the data, we can use various measures such as accuracy, precision, recall, ROC curve, AUC, etc.

We can also interpret the coefficients using odds ratios (OR), which are obtained by exponentiating the coefficients. The odds ratio tells us how much the odds of success change for every unit increase in an independent variable.

For example, in our model,

OR(credit score) = exp(0.01) = 1.01

OR(bank balance) = exp(-0.001) = 0.999

This means that for every one point increase in credit score, the odds of defaulting increase by 1%, holding bank balance constant.

Similarly, for every one dollar increase in bank balance, the odds of defaulting decrease by 0.1%, holding credit score constant.

Source: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Naïve Bayes Classifiers

Naive Bayes is a supervised machine learning algorithm that uses Bayes' theorem to perform classification tasks. It is based on the assumption that the features of the data are independent of each other given the class label. This simplifies the computation and makes the algorithm fast and scalable.

One of the advantages of Naive Bayes is that it can handle both continuous and discrete data. For example, it can be used to classify text documents based on the frequency of words or phrases. It can also be used to classify images based on the presence or absence of certain pixels or colors.

Naive Bayes works by calculating the posterior probability of each class given the features, and then choosing the class with the highest probability. The posterior probability is proportional to the product of the prior probability of the class and the likelihood of the features given the class. The prior probability represents our initial belief about the class distribution, and can be estimated from the training data or specified by some domain knowledge. The likelihood represents how likely the features are to occur in each class, and can be estimated using different methods depending on the type of data.

One of the challenges of Naive Bayes is that it may not perform well when some features are correlated or have different importance for different classes. It may also suffer from zero-frequency problem, which occurs when a feature value does not appear in the training data for a certain class, resulting in zero likelihood and zero posterior probability. This can be solved by using smoothing techniques, such as adding a small constant to the frequency counts.

Naive Bayes is a simple but powerful algorithm that can be applied to many real-world problems, such as spam filtering, sentiment analysis, medical diagnosis, and face recognition. It is also easy to implement and interpret, making it a good choice for beginners and experts alike.

K-Nearest Neighbors

The k-nearest neighbors' algorithm (KNN) is a simple and powerful technique for supervised learning, which can be used for both classification and regression problems. The main idea of KNN is to use the proximity or similarity of data points to assign labels or values to new observations. KNN does not require any training or parameter tuning, but only stores the entire training dataset in memory.

To make a prediction for a new data point, KNN finds the k closest or most similar points in the training dataset, where k is a positive integer that can be specified by the user. The similarity or distance between points can be measured by different metrics, such as Euclidean distance, Manhattan distance, or cosine similarity. Depending on the type of problem, KNN uses different strategies to make a prediction:

- For classification problems, KNN assigns the new data point to the most common class among its k nearest neighbors. This is also known as majority voting or plurality voting. For example, if k = 3 and the three nearest neighbors of a new data point are red, blue, and red, then KNN will classify the new data point as red.

- For regression problems, KNN assigns the new data point to the average value of its k nearest neighbors. For example, if k = 3 and the three nearest neighbors of a new data point have values 10, 12, and 14, then KNN will assign the new data point a value of 12.

KNN is a flexible and intuitive algorithm that can achieve high accuracy on many datasets. However, it also has some limitations and challenges, such as:

- Choosing an appropriate value of k that balances between overfitting and underfitting. A small value of k may lead to high variance and noise sensitivity, while a large value of k may lead to high bias and loss of local information.

- Choosing an appropriate distance metric that captures the relevant features and similarities of the data points. Different distance metrics may have different effects on the performance and interpretation of KNN.

- Dealing with high-dimensional and sparse data, which may cause the curse of dimensionality and make the distance metric less meaningful. Dimensionality reduction and feature extraction techniques may help to overcome this issue.

- Handling imbalanced and skewed data, which may affect the majority voting or averaging process of KNN. Weighted KNN or other methods may be used to give more importance to rare or minority classes or values.

- Managing computational complexity and memory requirements, which may become prohibitive for large datasets. Data reduction and indexing techniques may help to speed up the search for nearest neighbors and reduce the storage space.

Source: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

SVM

- WIP (I need to read more about this)

Gaussian Mixture Models

- WIP (I need to read more about this)

25. Unsupervised Learning

We have Clustering here, there is lot of methods that I will just make a citation.

They are: Hierarchical, K-Means, DBSCAN, HDBSCAN, Fuzzy C-Means, Mean Shift, Agglomerative, OPTICS

Hierarchical Clustering

Hierarchical clustering is a technique of cluster analysis that aims to create a hierarchy of clusters based on the similarity or dissimilarity of data points. It can be divided into two types: agglomerative and divisive. Agglomerative clustering starts with each data point as a separate cluster and then merges them based on a distance measure until a single cluster remains. Divisive clustering does the opposite: it starts with all data points in one cluster and then splits them recursively based on a distance measure until each data point is its own cluster.

One way to visualize hierarchical clustering is by using a dendrogram, which is a tree-like diagram that shows the nested structure of clusters and their distances. The height of each branch indicates how far apart the clusters are, and the horizontal line can be used to cut the tree at a desired level of granularity. The resulting clusters are the ones that are below the cut line.

Source: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec

K-Means

K-Means clustering is a popular unsupervised machine learning technique that aims to partition a set of observations into a number of clusters based on their similarity. In this post, we will explain the main steps of the K-Means algorithm, its applications, evaluation methods, and drawbacks.

The K-Means algorithm works as follows:

1. Specify the number of clusters K.

2. Randomly initialize K cluster centroids.

3. Assign each observation to the nearest cluster centroid.

4. Recompute the cluster centroids based on the assigned observations.

5. Repeat steps 3 and 4 until the cluster assignments do not change or a maximum number of iterations is reached.

The K-Means algorithm tries to minimize the within-cluster sum of squares (WCSS), which is the sum of squared distances between each observation and its cluster centroid. The lower the WCSS, the more homogeneous the clusters are.

Some of the applications of K-Means clustering are:

- Market segmentation: grouping customers based on their demographics, preferences, behaviors, etc.

- Image segmentation: grouping pixels based on their color, intensity, texture, etc.

- Document clustering: grouping documents based on their topics, keywords, sentiments, etc.

Some of the evaluation methods for K-Means clustering are:

- Elbow method: plotting the WCSS against different values of K and choosing the value where the WCSS curve has an elbow-like shape, indicating a trade-off between the number of clusters and the WCSS.

- Silhouette score: measuring how similar an observation is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1, where a higher value indicates a better clustering quality.

- Gap statistic: comparing the WCSS of the actual data to the WCSS of randomly generated data and choosing the value of K where the gap between them is largest.

Some of the drawbacks of K-Means clustering are:

- It requires specifying the number of clusters beforehand, which may not be easy or optimal.

- It is sensitive to outliers and noise, which can affect the cluster centroids and assignments.

- It is not suitable for clustering data that are not spherical or have different sizes and densities.

- It may converge to a local optimum depending on the initial cluster centroids.

Source: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

26. Association Rule Learning

Here we have Apriori, ECLAT and FP Trees

Apriori Algorithm

The Apriori Algorithm is a technique for finding frequent patterns or associations among a collection of items. It was proposed by Agrawal and Srikant in 1994 and is based on the idea that a subset of a frequent itemset must also be a frequent itemset. The algorithm works by iteratively generating candidate itemsets of increasing size and testing them against a minimum support threshold. The algorithm stops when no more candidates can be generated or all the candidates are infrequent.

The Apriori Algorithm can be applied to various domains, such as market basket analysis, web mining, text mining, etc. It can help discover interesting relationships or rules among items that occur together frequently.

Source: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec

ECLAT Algorithm

The ECLAT algorithm is a technique for finding frequent itemsets in a transactional database. It is based on the idea of equivalence classes and a depth-first search strategy. The ECLAT algorithm differs from the Apriori algorithm, which is another popular method for association rule mining, in several aspects.

First, the ECLAT algorithm uses a vertical data format, where each item is associated with a set of transaction identifiers (tids) that contain the item. For example, if item A appears in transactions 1, 3 and 5, its tidset is {1, 3, 5}. The Apriori algorithm, on the other hand, uses a horizontal data format, where each transaction is associated with a set of items that it contains.

Second, the ECLAT algorithm avoids scanning the entire database repeatedly to compute the support of each itemset, which can be costly for large databases. Instead, it computes the support of an itemset by intersecting the tidsets of its subsets. For example, if we want to compute the support of {A, B}, we can simply intersect the tidsets of A and B, i.e., {1, 3, 5} ∩ {1, 2, 4} = {1}. The size of this intersection is the support of {A, B}, which is 1 in this case.

Third, the ECLAT algorithm uses a depth-first search to explore the space of possible itemsets. It starts from single items and extends them by adding one item at a time. It also uses a prefix tree to store and share the intermediate results of frequent itemsets. The prefix tree is a compact data structure that avoids generating duplicated or invalid itemsets.

The ECLAT algorithm can be summarized as follows:

1. Initialize all single items as frequent itemsets.

2. For each frequent itemset I:

   - Find all items that can be appended to I to form a larger itemset.

   - For each such item J:

      - Generate a new candidate itemset by joining I and J.

      - Compute the support of the new itemset by intersecting the tidsets of I and J.

      - If the support is greater than or equal to the minimum support threshold, add the new itemset to the set of frequent itemsets.

      - Insert the new itemset and its support into the prefix tree as a child node of I.

   - Recursively repeat step 2 for each child node of I in the prefix tree until no more nodes can be generated.

The ECLAT algorithm can be used to discover interesting associations between items in a database. For example, it can help identify products that are frequently bought together by customers or topics that are frequently discussed together by users.

Source: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

FP Trees

FP Trees are a data structure that can be used to mine frequent patterns from a database efficiently. They are based on the idea of storing frequent itemsets in a compact tree form, where each node represents an item and each path represents a transaction. The FP Tree algorithm can find all frequent itemsets without generating any candidate sets, unlike the Apriori algorithm. This makes it faster and more scalable for large databases.

The FP Tree algorithm consists of two steps: building the FP Tree and extracting the frequent itemsets. To build the FP Tree, the algorithm first scans the database and counts the frequency of each item. Then, it sorts the items in descending order of frequency and creates a header table that stores the item names and their links. Next, it scans the database again and inserts each transaction into the tree, following the sorted order of items and creating new nodes as needed. The nodes also store a count value that indicates how many transactions contain that item. The header table links each item to its first occurrence in the tree, and each node links to its next occurrence of the same item. This way, the FP Tree preserves the information of the database in a compressed form.

To extract the frequent itemsets, the algorithm uses a recursive approach called FP Growth. It starts from the bottom of the header table and selects one item at a time. For each item, it constructs a conditional pattern base, which is a sub-database that contains only the transactions that include that item. Then, it constructs a conditional FP Tree from the conditional pattern base, and mines it for frequent itemsets that end with that item. This process is repeated until all items are processed or the conditional FP Tree is empty. The final result is the union of all frequent itemsets found in each step.

FP Trees are a powerful tool for data mining, as they can discover frequent patterns without scanning the database multiple times or generating large candidate sets. They can also handle sparse data and high-dimensional data effectively. However, they also have some limitations, such as requiring more memory space than Apriori and being sensitive to the order of items.

Source:
http://www.hypertextbookshop.com/dataminingbook/public_version/contents/chapters/chapter002/section006/blue/page001.html

27. Dimensionality Reduction

Here we have PCA (Principal Component Analysis), Random Projection, NMF, T-SNE and UMAP.

I Don't know about the last ones, but I know about PCA.

PCA (Principal Component Analysis)

Principal component analysis (PCA) is a technique that allows you to reduce the dimensionality of a large dataset by transforming it into a smaller one that preserves most of the information. PCA can help you to analyze and visualize data that has many features or variables, and to identify patterns and trends in the data.

To perform PCA, you need to follow these steps:

1. Standardize the data so that each feature has a mean of zero and a standard deviation of one. This ensures that each feature contributes equally to the analysis and avoids bias due to different scales or units.

2. Compute the covariance matrix of the standardized data. This matrix shows how much each feature varies with respect to each other, and how correlated they are.

3. Compute the eigenvectors and eigenvalues of the covariance matrix. These are the directions and magnitudes of the principal components, which are the new features that capture the most variation in the data. The eigenvectors are orthogonal to each other, meaning they are uncorrelated.

4. Create a feature vector by selecting the eigenvectors that correspond to the largest eigenvalues. This vector determines how many principal components you want to keep, and how much information you want to retain from the original data.

5. Recast the data along the principal components axes by multiplying the standardized data by the feature vector. This gives you the reduced dataset with fewer dimensions, which can be easier to explore and visualize.

For example, if you have a dataset of 100 observations with 10 features each, you can use PCA to reduce it to a dataset of 100 observations with 2 features each, by choosing the first two principal components that explain most of the variation in the data. You can then plot these two features on a scatter plot and see if there are any clusters or outliers in your data.

PCA is a useful tool for data analysis and machine learning, as it can help you to simplify complex data, identify important features, and reduce noise and redundancy.

Let's talk about Ensemble Learning

28. Ensemble Learning

Ensemble learning is a meta-approach to machine learning that combines the predictions of multiple learning models to achieve better performance than any single model alone. Ensemble learning can reduce the errors caused by noise, variance, and bias in the data and improve the generalization and robustness of the models. There are three main types of ensemble learning methods: bagging, stacking, and boosting. Bagging involves training multiple models, usually decision trees, on different subsets of the data and averaging their predictions. Stacking involves training multiple models of different types on the same data and using another model to learn how to best combine their predictions. Boosting involves training multiple models sequentially, each one trying to correct the errors of the previous ones, and weighting their predictions according to their accuracy.

There are three main methods that I know: Boosting, Bagging and Stacking.

Boosting

Boosting is a technique that improves the accuracy and performance of machine learning models by combining multiple weak learners into a single strong learner. A weak learner is a model that has low prediction accuracy, while a strong learner is a model that has high prediction accuracy. Boosting works by training weak learners sequentially on different subsets of the data, and then assigning weights to their outputs based on their errors. The final output is a weighted combination of the weak learners' outputs, which results in a more accurate and robust prediction.

One of the most popular boosting algorithms is AdaBoost, which stands for Adaptive Boosting. AdaBoost works by initially assigning equal weights to all the data points in the training set. Then, it trains a weak learner on the data and calculates its error rate. Based on the error rate, it updates the weights of the data points, increasing the weights of the misclassified points and decreasing the weights of the correctly classified points. This way, it forces the next weak learner to focus more on the difficult points. This process is repeated until a predefined number of weak learners are trained or the error rate reaches zero. The final output is a weighted majority vote of the weak learners' outputs.

Boosting is a powerful method that can improve the performance of any machine learning model, especially decision trees. Decision trees are simple models that split the data based on some criteria at each node, forming a tree-like structure. However, decision trees tend to overfit the data, meaning they cannot generalize well to new data that differs from their training data. By using boosting, decision trees can overcome this problem and become more accurate and stable. Some examples of boosting algorithms that use decision trees are Gradient Boosting, XGBoost, LightGBM, and CatBoost.

Boosting is widely used in various domains and applications, such as computer vision, natural language processing, recommender systems, fraud detection, and more. It is one of the most popular and effective machine learning techniques that can enhance any model's performance with minimal effort.

Source: https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5

Bagging

Bagging is a machine learning technique that combines multiple base models to produce a more robust and accurate ensemble model. Bagging stands for bootstrap aggregating, which means that each base model is trained on a random subset of the training data, with replacement. This reduces the variance of the individual models and makes them less prone to overfitting. Bagging can be applied to any type of machine learning algorithm, such as decision trees, neural networks, or linear regression. The final prediction of the ensemble model is obtained by averaging the predictions of the base models for regression problems, or by voting for classification problems. Bagging is especially useful when the base models have high variance and low bias, such as complex decision trees.

Some of the python libraries that can be used to implement bagging are scikit-learn, which provides a BaggingClassifier and a BaggingRegressor class, and mlxtend, which provides a Bagging meta-estimator that can be used with any base estimator.

Stacking

Stacking in machine learning is a technique that combines the predictions of multiple models to produce a new model with better accuracy and performance. It is also known as stacked generalization. The idea behind stacking is to explore different types of models that can learn different aspects of the problem and then use their predictions as features for a second-level model that learns the final output.

Stacking can be done in two steps:

- First, we split the training data into K folds and train multiple base models on K-1 folds. Then we use the base models to make predictions on the remaining fold. We repeat this process for each fold and collect all the predictions as a new feature matrix. We also use the base models to make predictions on the test set and store them for later use.

- Second, we train a second-level model (also called a meta-learner) on the new feature matrix and use it to make final predictions on the test set using the stored predictions as features.

Stacking can be applied to both classification and regression problems. It can also be extended to use more than two levels of models, but this may increase the complexity and risk of overfitting.

Some examples of python libraries that implement stacking are:

- mlxtend: A library that provides a StackingClassifier and a StackingRegressor class that can stack arbitrary scikit-learn estimators.

- vecstack: A library that provides a stacking function that can stack any models that follow the scikit-learn API.

- h2o: A library that provides an H2OStackedEnsembleEstimator class that can stack H2O models using cross-validation or blending.

29. Reinforcement Learning

Reinforcement learning is a branch of machine learning that focuses on learning from trial and error. It is based on the idea that an agent can learn to perform a task by interacting with an environment and receiving rewards or penalties for its actions. Reinforcement learning can be used to solve complex problems that require sequential decision making, such as game playing, robotics, self-driving cars, and natural language processing.

One of the main challenges of reinforcement learning is to balance exploration and exploitation. Exploration means trying out new actions that may lead to better outcomes in the future, while exploitation means choosing the best action based on the current knowledge. A common way to achieve this balance is to use an epsilon-greedy policy, which randomly chooses an exploratory action with a small probability epsilon, and otherwise chooses the best action according to a value function.

There are many python libraries that can help with implementing reinforcement learning algorithms. Some of the most popular ones are:

- TensorFlow: A library for creating and training neural networks, which can be used as function approximators in reinforcement learning.

- PyTorch: Another library for creating and training neural networks, with a more dynamic and flexible approach than TensorFlow.

--I haven't even touched this library but I going to include them.

- Gym: A library that provides a collection of environments for testing and benchmarking reinforcement learning algorithms.

- Stable Baselines: A library that provides implementations of state-of-the-art reinforcement learning algorithms, such as DQN, A2C, PPO, and SAC.

- Ray: A library that provides distributed execution and scalability for reinforcement learning applications.

Q-Learning

Q-Learning is a model-free reinforcement learning algorithm that learns the value of an action in a particular state without requiring a model of the environment. It can handle problems with stochastic transitions and rewards by learning from its own experience. Q-Learning aims to find the optimal policy that maximizes the expected value of the total reward over any and all successive steps, starting from the current state. The Q function represents the quality of an action, which is the expected future reward for taking that action in a given state. Q-Learning uses a Q table to store the Q values for each state-action pair. The Q table is updated iteratively using the Bellman equation, which incorporates the immediate reward and the discounted future reward. Q-Learning is an off-policy algorithm, which means it evaluates and improves a different policy than the one used to explore the environment. Some popular Python libraries for implementing Q-Learning are OpenAI Gym, TensorFlow, PyTorch, and Keras-RL.

30. Other Cases Learning

Here we have Sentiment Analysis, Collaborative Filtering, Tagging and Prediction.

I can only talk about Tagging in this case so…

Tagging

Tagging is the process of assigning keywords or labels to online content, such as news articles, videos, or social media posts, to make them more easily searchable and accessible. Tagging can help content marketers to understand the performance and impact of their content across different channels and platforms, as well as to provide personalized and relevant experiences for their audiences. However, tagging can also be challenging and time-consuming, especially when dealing with large and complex datasets that require multilabel classification, i.e., assigning multiple tags to each content unit.

Machine learning is a branch of artificial intelligence that uses algorithms and data to learn from patterns and make predictions. Machine learning can be applied to automate the tagging process and improve its accuracy and efficiency. Machine learning can also enable intelligent content, which is content that is dynamic, adaptable, and responsive to user needs and preferences. Intelligent content can enhance user engagement, satisfaction, and loyalty, as well as optimize content marketing outcomes.

There are different machine learning techniques that can be used for auto-tagging online content, such as random forest, k-nearest neighbor, and neural network. These techniques vary in their complexity, performance, and applicability. For example, neural network is a technique that mimics the structure and function of the human brain, and can achieve high accuracy and generalizability for multilabel classification of online content. However, neural network also requires more computational resources and data than other techniques.

Source: https://collaboro.com/tagging-machine-learning-and-intelligent-content-why-you-should-give-a-dam/

31. Random Forest (Supervised Machine Learning)

Random Forest is a supervised machine learning algorithm that can be used for both classification and regression problems. It is based on the idea of creating multiple decision trees from a subset of features and data, and then combining their predictions using a voting or averaging scheme. Random Forest has several advantages over other algorithms, such as:

- It can handle high-dimensional and nonlinear data with ease.

- It can reduce overfitting and variance by introducing randomness and diversity in the tree construction process.

- It can provide estimates of feature importance and variable interactions.

- It can deal with missing values and outliers without requiring preprocessing.

However, Random Forest also has some drawbacks, such as:

- It can be computationally expensive and slow to train and test, especially with large datasets and many trees.

- It can be difficult to interpret and explain the individual trees and their predictions.

- It can suffer from bias if the data is imbalanced or noisy.

Therefore, Random Forest is a powerful and versatile machine learning algorithm that can be applied to many problems, but it also requires careful tuning and evaluation to achieve optimal performance.

32. Libraries

Python is one of the most popular programming languages for machine learning, thanks to its simplicity, readability, and versatility. Python also has a rich ecosystem of libraries that provide various tools and functionalities for machine learning tasks. Some of the most important libraries for machine learning in Python are:

- NumPy: NumPy stands for Numerical Python and is the core library for scientific computing in Python. It provides high-performance multidimensional arrays and various mathematical operations on them. NumPy is essential for working with numerical data and implementing algorithms.

- Pandas: Pandas is a library for data analysis and manipulation in Python. It offers data structures such as Series and DataFrame that make it easy to handle tabular and time-series data. Pandas also provides various methods for data cleaning, filtering, grouping, aggregating, merging, and reshaping.

- Scikit-learn: Scikit-learn is a library for machine learning in Python that provides a consistent and user-friendly interface for various algorithms and models. Scikit-learn supports supervised learning (such as regression, classification, and clustering), unsupervised learning (such as dimensionality reduction, feature extraction, and anomaly detection), and model selection and evaluation (such as cross-validation, grid search, and metrics).

- TensorFlow: TensorFlow is a library for deep learning in Python that allows users to create and train neural networks using tensors (multidimensional arrays) and graphs (computational models). TensorFlow supports various types of neural networks, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and generative adversarial networks (GANs). TensorFlow also offers high-level APIs such as Keras and Estimators that simplify the development process.

- PyTorch: PyTorch is another library for deep learning in Python that is based on Torch, a scientific computing framework. PyTorch provides tensors and dynamic computation graphs that enable users to define and modify neural networks on the fly. PyTorch also supports various types of neural networks, such as CNNs, RNNs, and GANs. PyTorch is known for its flexibility, speed, and ease of use.

Source: https://github.com/vinta/awesome-python

Here we have also spacy (NLP), however I never used it.

Deep Learning

At this point I am still learning a lot about Deep Learning, my background in Engineering helped me a lot with the mathematics around.

Here is the roadmap in working on: https://github.com/floodsung/Deep-Learning-Papers-Reading-Roadmap

A useful place to find code: https://paperswithcode.com/ and https://paperswithcode.com/sota

33. Neural Networks

Neural networks are a type of machine learning algorithm that can learn complex patterns from data. They are inspired by the structure and function of the human brain, which consists of billions of interconnected neurons that process information. Neural networks are composed of layers of artificial neurons that receive inputs, perform calculations, and produce outputs. The inputs can be any kind of data, such as images, text, or numbers. The outputs can be predictions, classifications, or decisions.

A neural network has three main components: an input layer, one or more hidden layers, and an output layer. The input layer has one neuron for each feature in the input data. The hidden layers have a variable number of neurons that perform computations on the inputs and pass them to the next layer. The output layer has one neuron for each possible outcome or label. Each neuron in a layer is connected to every neuron in the next layer by a weight, which represents the strength of the connection. Each neuron also has a bias, which is a constant value that shifts the output of the neuron.

The process of neural network learning involves two steps: forward pass and backward pass. In the forward pass, the input data is fed into the input layer and propagated through the hidden layers until it reaches the output layer. Each neuron calculates a weighted sum of its inputs and adds its bias. Then, it applies an activation function, which determines whether the neuron should fire or not. The activation function can be different for each layer or neuron, but some common ones are sigmoid, tanh, ReLU, and softmax. The output of each neuron is then passed to the next layer until it reaches the output layer, where the final prediction or classification is made.

In the backward pass, also known as backpropagation, the neural network compares its output with the actual target or label and calculates an error or loss function. The loss function measures how well the neural network performed on a given input. The goal of neural network learning is to minimize this loss function by adjusting the weights and biases of each neuron. To do this, the neural network uses a technique called gradient descent, which calculates how much each weight and bias contributed to the error and updates them accordingly. This process is repeated for many iterations or epochs until the neural network converges to a satisfactory level of accuracy.

Neural networks are powerful and versatile tools that can be used for many applications, such as computer vision, natural language processing, speech recognition, and more. They can learn from large amounts of data and generalize to new situations. However, they also have some limitations and challenges, such as overfitting, underfitting, vanishing gradients, exploding gradients, and interpretability. These issues require careful design choices and optimization techniques to overcome them. These problems are not anything new and require a work similar to what we were used, but applied to this new science.

34. Loss Function

A loss function is a function that measures the error between a prediction and the actual value. It is used to determine how far off the computed output is from the target value. Loss functions are not fixed and change depending on the task and goal. In mathematical optimization and decision theory, a loss function maps an event or values of one or more variables onto a real number representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.

There are different types of loss functions for different problems, such as classification or regression. For example, in classification problems, where the goal is to assign a label to an input, a common loss function is the zero-one loss function, which takes the value of 0 if the predicted label matches the true label, or 1 otherwise. Another example is the quadratic loss function, which is used in regression problems, where the goal is to predict a continuous value. The quadratic loss function is the squared difference between the predicted value and the true value.

Loss functions are important because they affect the performance and accuracy of the learning algorithm. Different loss functions may lead to different optimal solutions and different trade-offs between bias and variance. Choosing a suitable loss function depends on the nature of the data, the objective of the analysis, and the assumptions of the model.

35. Activation functions

An activation function is a mathematical function that determines the output of a neuron in an artificial neural network, based on its input. Activation functions are essential for neural networks to learn complex patterns and perform nonlinear operations. There are different types of activation functions, such as linear, nonlinear, threshold-based, and continuous. One of the sources that explains activation functions is https://en.wikipedia.org/wiki/Activation_function.

Some examples of activation functions are:

- Linear function: This is a simple function that returns a scaled version of its input, such as f(x) = ax + b. A linear function has the advantage of being easy to compute and train, but it cannot capture nonlinear relationships between the input and output.

- Sigmoid function: This is a nonlinear function that maps any real value to a value between 0 and 1, such as f(x) = 1 / (1 + exp(-x)). A sigmoid function has the advantage of being smooth and differentiable, but it suffers from problems such as vanishing gradients and saturation.

- ReLU function: This is a nonlinear function that returns 0 for negative inputs and the input itself for positive inputs, such as f(x) = max (0, x). A ReLU function has the advantage of being fast to compute and train, but it suffers from problems such as dying neurons and unbounded output.

36. Vanishing Gradient Problem

The vanishing gradient problem is a challenge that arises when training deep neural networks with gradient-based methods and backpropagation. It occurs when the gradient values become very small or zero as they propagate from the output layer to the input layer, preventing the network from learning effectively. This problem can affect both feedforward and recurrent neural networks, especially when they have many layers. The vanishing gradient problem can be solved by using different activation functions, such as rectified linear units (ReLUs), that do not saturate easily, or by introducing skip connections, such as in residual networks (ResNets), that allow the gradient to flow faster through the network.

37. Weight Initialization

Weight initialization is a procedure to set the weights of a neural network to small random values that define the starting point for the optimization (learning or training) of the neural network model. The aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. Different weight initialization techniques have been proposed for different types of activation functions and network architectures. **One of the most popular techniques** is based on the idea of maintaining the variance of the inputs and outputs of each layer. **Xavier initialization** uses a uniform or normal distribution with zero mean and a specific variance that depends on the number of inputs and outputs of the layer. Another technique is called **He initialization**, which is designed for layers that use the ReLU activation function. He initialization uses a normal distribution with zero mean and a variance that is proportional to the number of inputs of the layer. Both Xavier and He initialization aim to make the learning process faster and more stable by avoiding gradient problems.

Source: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79

These are all the concepts.

Let's talk about the Training

38. Optimizers

Here we have a lot of optimizers like: SGD, Momentum, Adam, AdaGrad, AdaDelta, Nadam, RMSProp, etc.

I will talk only about three of them:

39. Adagrad

Adagrad is a gradient-based optimization algorithm that adapts the learning rate for each parameter based on the history of gradients. It is especially useful for sparse data and problems with different scales of features. Adagrad works by accumulating the squared gradients for each parameter in a diagonal matrix and dividing the current gradient by the square root of this matrix. This effectively reduces the learning rate for parameters that have large gradients and increases it for parameters that have small gradients. Adagrad can be seen as an approximation of second-order methods that use the Hessian matrix of the objective function.

Adagrad has theoretical guarantees for convex and non-convex optimization problems, and that it performs well on various machine learning tasks, such as natural language processing, computer vision and speech recognition. Adagrad is also easy to implement and does not require tuning of hyperparameters, such as the global learning rate.

One of the drawbacks of Adagrad is that it can become too aggressive in reducing the learning rate, especially for parameters that are updated frequently. This can lead to premature convergence or slow progress. To overcome this issue, variants of Adagrad have been developed, such as AdaDelta, RMSProp and Adam, which use different ways of accumulating and decaying the gradient history.

40. AdaDelta

Adadelta is a stochastic optimization technique that allows for per-dimension learning rate method for stochastic gradient descent (SGD). It is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to a fixed size w. It uses a decaying average of all past squared gradients to adjust the learning rate for each parameter. Adadelta also maintains a decaying average of all past parameter updates, which is used to scale the current parameter update. Adadelta does not require a default learning rate, as it adapts based on the history of parameter updates. More investigation is required, so… WIP

41. Nadam

NADAM is an optimization algorithm for deep learning that combines two techniques: Nesterov momentum and Adam. Nesterov momentum is a variant of gradient descent that uses the gradient of the future position instead of the current position to update the parameters. Adam is an adaptive learning rate method that computes individual learning rates for different parameters based on the first and second moments of the gradients. NADAM combines the advantages of both methods and can achieve faster convergence and better performance than standard gradient descent, momentum, or Adam.

Note: There is a pdf in my GitHub for source. There are many Gradient decent algorithms there, like Adagrad, AdaDelta, RMSProp, Adam, etc. Check it!

END of Optimizers Topic!

42. Learning Rate Schedule

Learning rate schedule is a predefined framework that adjusts the learning rate between epochs or iterations as the training progresses. It is a technique to help the optimization algorithm converge quickly and avoid overshooting or getting stuck in local minima. There are different types of learning rate schedules, such as constant, step, exponential, cosine, and cyclic. Each schedule has its own advantages and disadvantages depending on the problem and the model.

The Wikipedia page describes four learning rate schedules: time-based decay, step decay, exponential decay, and adaptive learning rate. These schedules are based on reducing the learning rate as the training progresses, following a certain formula or rule. The idea is to start with a high learning rate to explore the parameter space quickly, and then gradually decrease it to fine-tune the model near the optimal solution. The main difference between these schedules is how fast and when they reduce the learning rate.

Time-based decay reduces the learning rate by a constant factor every epoch. The formula is:

lr = lr0 / (1 + kt)

where lr0 is the initial learning rate, k is a hyperparameter, and t is the current epoch. This schedule is easy to implement and has a smooth decay curve. However, it may not be flexible enough to adapt to different stages of training.

Step decay reduces the learning rate by a constant factor every few epochs. The formula is:

lr = lr0 * drop^floor(t / epochs_drop)

where lr0 is the initial learning rate, drop is a hyperparameter between 0 and 1, epochs_drop is the number of epochs after which the learning rate is reduced, and t is the current epoch. This schedule allows more control over how often and how much to reduce the learning rate. However, it may introduce abrupt changes in the learning rate that can affect the model performance.

Exponential decay reduces the learning rate by an exponential factor every epoch. The formula is:

lr = lr0 * exp(-kt)

where lr0 is the initial learning rate, k is a hyperparameter, and t is the current epoch. This schedule has a fast decay at the beginning and a slow decay at the end. It can help to escape from poor local minima and reach a good solution quickly. However, it may also reduce the learning rate too much and prevent further improvement.

Adaptive learning rate adjusts the learning rate based on a performance metric, such as validation loss or accuracy. The idea is to increase the learning rate when the metric improves and decrease it when it worsens. There are different algorithms that implement this idea, such as AdaGrad, RMSProp, Adam, and Nadam. These algorithms use different formulas to update the learning rate for each parameter based on its gradient history. They can adapt to different scenarios and achieve good results with less tuning. However, they may also have some drawbacks, such as being sensitive to noise or requiring more computation.

43. Batch Normalization

Batch normalization is a technique that makes the training of artificial neural networks faster and more stable. It normalizes the inputs of each layer by re-centering and re-scaling them, so that they have a mean of zero and a standard deviation of one. This helps to avoid the problem of internal covariate shift, which is the change in the distribution of the inputs of each layer due to the randomness in the parameter initialization and the input data.

Some of the advantages of batch normalization are:

- It allows higher learning rates, since it prevents the gradients from exploding or vanishing.

- It reduces overfitting, since it acts as a regularizer by adding noise to the inputs of each layer.

- It enables each layer to learn independently of the others, since it reduces the dependency on the initialization and the previous layers.

Batch normalization works by applying the following formula to each input x_i of a layer:

y_i = gamma * (x_i - mu) / sigma + beta

where mu and sigma are the mean and standard deviation of x_i over a mini-batch, gamma and beta are learnable parameters that control the scale and shift of the normalized output y_i. During training, batch normalization also keeps track of the running mean and variance of each layer, which are used during inference instead of the mini-batch statistics. This ensures that the outputs of each layer have consistent distributions during both training and testing phases.

44. Batch Size Effects

The effect of batch size on training dynamics is an important topic in deep learning. Batch size refers to the number of training examples used to estimate the error gradient in each iteration of gradient descent. Different batch sizes can have different impacts on the speed, stability, and accuracy of the learning process.

There are three main characteristics in gradient descent based on the batch size: batch, stochastic, and minibatch. Batch gradient descent uses the entire training set to compute the error gradient, which guarantees convergence to the global optimum, but is slow and computationally expensive. Stochastic gradient descent uses a single training example to compute the error gradient, which is fast and can escape local optima, but is noisy and unstable. Minibatch gradient descent uses a small subset of the training set to compute the error gradient, which balances the trade-offs between batch and stochastic methods.

The optimal batch size depends on various factors, such as the dataset, the model, the optimization algorithm, and the hardware. In general, larger batch sizes can result in faster training, but may not converge as well or generalize as well as smaller batch sizes. Smaller batch sizes can result in slower training, but may converge faster or generalize better. Therefore, it is advisable to experiment with different batch sizes and monitor the training and validation metrics to find the best one for a given problem.

Source: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

45. Regularization (Maths)

Regularization is a technique that makes a mathematical model more generalizable and transferable by adding a penalty for complexity or extra information to the model. It is often used in machine learning and inverse problems to prevent overfitting or to obtain results for ill-posed problems. Regularization introduces a parameter called the regulator that controls the strength of the penalty or the amount of extra information. There are different types of regularization, such as L1, L2, early stopping, sparsity, and multitask learning. Each type has its own advantages and disadvantages depending on the problem and the data.

With this explained we can have 5 techniques here that are: Early Stopping, Dropout, Parameter Penalties, Data Augmentation, Adversarial Training. I'll do as much as I can to get into all those five.

46. Early Stopping

Early stopping is a technique to prevent overfitting when training a machine learning model with an iterative method, such as gradient descent. Overfitting occurs when a model learns the training data too well and fails to generalize to new data. Early stopping stops the training process before the model reaches the point of overfitting.

One way to implement early stopping is to use a validation set, which is a subset of the training data that is not used for learning the model parameters, but only for evaluating its performance. The idea is to monitor the validation error (the error on the validation set) during the training process and stop the training when the validation error starts to increase or stops decreasing. This indicates that the model is no longer improving on the unseen data and may be overfitting the training data.

Another way to implement early stopping is to use analytical results from statistical learning theory or boosting algorithms. These results provide bounds or estimates on the generalization error of the model based on the number of iterations, the complexity of the model, and the size of the training data. By choosing an appropriate number of iterations that minimizes the bound or estimate, one can achieve early stopping without using a validation set.

Early stopping is a simple and effective regularization technique for machine learning models. It can reduce the computational cost and time of training, as well as improve the generalization performance of the model.

47. Dropout

Dropout is a technique for regularizing deep neural networks by randomly dropping out some of the neurons during training. This prevents co-adaptation of neurons and reduces overfitting. Dropout can be applied to any layer of a neural network, including the input layer. The dropout rate is the fraction of neurons that are zeroed out in each layer.

Dilution is a related concept that refers to the effect of dropout on the weights of a neural network. When dropout is applied, the weights of the remaining neurons are scaled up by a factor of 1/(1-p), where p is the dropout rate. This ensures that the expected value of the output of each layer remains unchanged. Dilution can be seen as a way of regularizing the weights by shrinking their squared norm.

Note: WIP

48. Parameter Penalties

Parameter penalties are a way of regularizing neural networks by adding a term to the objective function that depends on the norm of the parameters. The idea is to prevent overfitting by penalizing complex models with large parameter values. There are different types of parameter penalties, such as L1, L2, and elastic net, which use different norms to measure the complexity of the model. L1 penalty uses the absolute value of the parameters, L2 penalty uses the square of the parameters, and elastic net penalty uses a combination of both. Parameter penalties can be applied to different layers of a neural network, such as the kernel, the bias, or the output. For example, in Keras, one can specify the kernel_regularizer, bias_regularizer, or activity_regularizer arguments when creating a layer. Parameter penalties are also known as weight decay or shrinkage methods in some contexts.

Note: WIP

49. Data Augmentation

Data augmentation is a technique that can improve the performance and generalization of machine learning models by creating modified copies of the existing data. It can help to prevent overfitting, increase the size and diversity of the training set, and reduce the cost of data collection and labeling. Data augmentation can be applied to different types of data, such as audio, text, image, and signal. Depending on the type of data, different methods of augmentation can be used, such as noise injection, word replacement, geometric transformations, random erasing, or generative adversarial networks (GANs). Data augmentation can also be done using TensorFlow, a popular framework for deep learning. TensorFlow provides various tools and functions for data augmentation, such as tf.image for image augmentation, tf.data for dataset manipulation, and tf.keras.layers.experimental.preprocessing (Just a reminder for me) for preprocessing layers. Data augmentation can be done asynchronously on the CPU while the model is training on the GPU, using Dataset.prefetch. This can improve the efficiency and speed of the training process.

50. Adversarial Training

Adversarial training is a technique in deep learning that aims to improve the robustness and generalization of neural networks by exposing them to adversarial examples. Adversarial examples are inputs that are slightly perturbed in a way that causes the network to make incorrect predictions, but are imperceptible to human eyes. Adversarial training involves adding adversarial examples to the training data and updating the network parameters to minimize the loss on both the original and the adversarial inputs. Adversarial training can help the network learn more diverse and invariant features, and prevent overfitting to the training distribution.

Note: There is a paper in my GitHub, I recommend reading it!

End of this topic

51. Transfer Learning

Transfer learning is a research problem in machine learning that focuses on applying knowledge gained while solving one task to a related task. For example, knowledge gained while learning to recognize cars could be applied when trying to recognize trucks. Transfer learning is useful in deep learning because it can train deep neural networks with comparatively little data. Instead of starting the learning process from scratch, we start with patterns learned from solving a related task.

Transfer learning works by reusing the weights that a network has learned at one task for a new task. The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data. In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. By using a pre-trained model on a large and general dataset, such as ImageNet, we can take advantage of these learned feature maps without having to start from scratch. This is a common practice to save time.

There are different approaches to transfer learning, such as training a model to reuse it, using a pre-trained model, or feature extraction. Depending on the size and similarity of the datasets and tasks, we can choose different strategies for fine-tuning the pre-trained model or freezing some of its layers. Transfer learning can help us achieve better performance and faster convergence on new tasks that are related to the ones we have already solved.

52. Multi-task Learning

Multi-task learning (MTL) is a subfield of machine learning that aims to solve multiple related tasks at the same time, by exploiting the commonalities and differences among them. MTL can improve the learning efficiency and prediction accuracy of the task-specific models, compared to training them separately. MTL can also act as a regularizer, preventing overfitting by penalizing complexity uniformly across tasks.

One example of MTL is to train a neural network to perform multiple natural language processing tasks, such as part-of-speech tagging, named entity recognition, and sentiment analysis. By sharing some of the network's layers and parameters across tasks, the network can learn a more general and robust representation of the language, which can benefit each individual task.

MTL can be implemented in different ways, depending on how the tasks are related and how much information is shared among them. Two common methods for MTL in deep learning are hard parameter sharing and soft parameter sharing. Hard parameter sharing means that some of the network's layers are shared across all tasks, while each task has its own output layer. This reduces the number of parameters and the risk of overfitting. Soft parameter sharing means that each task has its own network, but the parameters of corresponding layers are regularized to be similar. This allows more flexibility and task-specificity, but also increases the computational cost.

53. Advanced Model Optimization

Here we have methods like Distillation, Quantization and NAS (Neural Architecture Search).

Distillation

Knowledge distillation is a machine learning technique that transfers knowledge from a large model (called the teacher) to a smaller model (called the student) without losing much accuracy. The idea is that the teacher model has a higher knowledge capacity than the student model, but this capacity might not be fully utilized. Therefore, the student model can learn a more concise and efficient representation of the knowledge by mimicking the teacher model's output distribution. This way, the student model can be deployed on less powerful devices or achieve faster inference speed than the teacher model.

One way to perform knowledge distillation is to train the teacher model on a large dataset and then use its predictions as soft labels for training the student model. The soft labels contain more information than the hard labels (the ground truth class labels) because they reflect the teacher model's confidence and uncertainty about each class. The student model learns to match the soft labels by minimizing a loss function that compares its output distribution with the teacher model's output distribution. This loss function can be combined with another loss function that compares the student model's output with the hard labels, to balance between distillation and generalization.

Knowledge distillation has been applied to various domains of machine learning, such as object detection, natural language processing, acoustic models, and graph neural networks. It can improve the performance of small models or reduce the computational cost of large models. It can also help transfer knowledge across different modalities, tasks, or domains.

Quantization

Quantization is a technique that reduces the precision of the numerical values used in deep learning models, such as weights, activations, and gradients. By using lower-precision values, such as 8-bit integers instead of 32-bit floats, quantization can reduce the memory footprint and computational cost of deep learning models, while maintaining acceptable accuracy. Quantization can also enable the deployment of deep learning models on hardware platforms that do not support floating-point operations, such as microcontrollers and edge devices. In this article, we will explore the different types of quantization, the trade-offs between performance and accuracy, and some practical examples of quantization in TensorFlow and PyTorch.

You can google about the Facebook Matrix multiplication library and learn more about Quantization.

NAS

Neural Architecture Search (NAS) is a technique for automating the design of artificial neural networks (ANN), a widely used model in the field of machine learning. NAS has been used to design networks that are on par or outperform hand-designed architectures. NAS can be categorized according to the search space, search strategy and performance estimation strategy used:

- The search space defines the type (s) of ANN that can be designed and optimized.

- The search strategy defines the approach used to explore the search space.

- The performance estimation strategy evaluates the performance of a possible ANN from its design (without constructing and training it).

NAS is closely related to hyperparameter optimization and meta-learning and is a subfield of automated machine learning (AutoML). It can use different methods for the search strategy, such as reinforcement learning, evolutionary algorithms, gradient-based methods, or Bayesian optimization. NAS can target different tasks and datasets, such as image classification, natural language processing, speech recognition, or reinforcement learning.

NAS is a promising technique for advancing machine learning research and applications by automating one of the most challenging and time-consuming aspects of deep learning: the design of neural architectures.

54. Libraries for Deep Learning

In this section we have already talked about the most common and there is a high chance that you already heard about libraries like TensorFlow and Pytorch.

However, this GitHub's are pretty god.

1. Awesome Deep Learning

There is so much to learn here, from books, courses, papers, just check the Table of Contents!

https://github.com/ChristosChristofidis/awesome-deep-learning

2. Transformers

Lots of Demos here, good to learn too!

https://github.com/huggingface/transformers

55. This is the end for now,

WIP