Data Cleaning

Manipulation

Large Scale Data A.

Lecture 1

Big Data Intro

Teachers — Part of DASYA at ITU ☺

What is Big Data

Volume - there is a LOT of data ☺ (a few TB)
Velocity - data is generated QUICKLY
Variety

Scaleability — How good is a system at adapting to more/less data, more/less ressources etc.

Project 1 — Implement sklearn-Pipeline for wind power prediction!

Project 2 — Large Scale experiments with mlflow

Project 3 — Query & analyze yelp dataset using apache Spark ☺

Machine Learning!

Proces

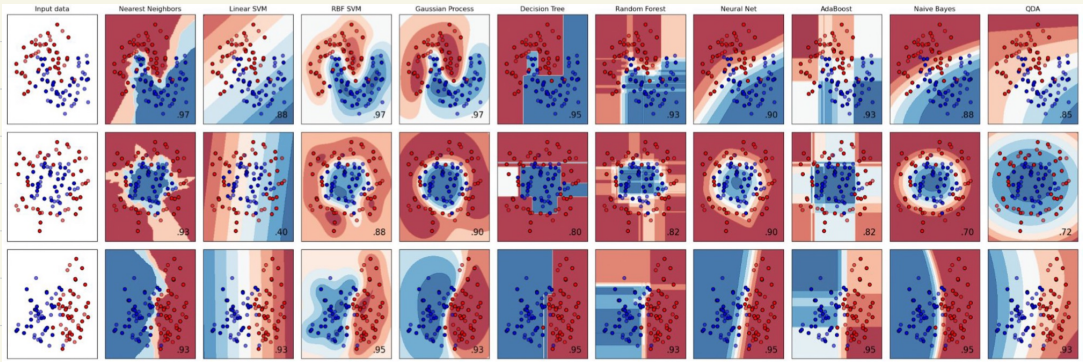
(1) choose model

(2) get data in new environment!
+ inspection & train-test split

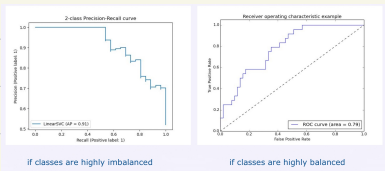
(3) visualize and analyze data in-depth

(4) data cleaning (merge data + dropcolumns (avoid cor. features)
+ handle missing values + rescale
+ encode categorical features

(5) model selection



(6) Fine-tuning model
+ evaluation metrics for hyperparameter
R^2, MSE, Cross-Entropy, misclassification



if classes are highly imbalanced | if classes are highly balanced

(7) Present Solution

(8) Launch, Monitor & Maintain

Pd for missing values

sklearn.impute.SimpleImputer or .KNNImputer
pandas.DataFrame.dropna or .fillna or .interpolate

Machine Learning! (2)!

Hyper Parameter Tuning

Manual we manually evaluate what p should be
Grid Search we check all possible subsets of features
Randomized Search When p-space is too large

Model Deployment

We can deploy models to PRD, or to a public Ip-address... f.ex. Azure Cloud ☺
We need to serialize before we publish!

Pipeline is: Batch

Periodic predictions using historical data. Might have a trigger or time interval or similar

Online

Live predictions based on streaming data. Predictions are updated frequently /near Live ☺

Computation in: Cloud

Everything is done "in the cloud"

Edge

Computations are done on the device itself, but it is still connected to the cloud. This type of model will work well for unstable connections/avoids network latency/no data transfer

Monitoring

We monitor the model to ensure performance
Operational metrics (latency, #requests, CPU utilization)
ML metrics (user feedback, preds, features, inputs)

Distribution Shift

Covariate shift $P(X)$ changes; people are poorer
Label shift $P(Y)$ changes; Covid cases rise in fall
Concept drift $P(Y|X)$ changes; Input is the same
Buy=False rises in crisis

Big Data Challenges

ML challenges

Dependencies of model can be hard to maintain - think online prediction / streaming
We can freeze the system (including OS), and this is called virtualization of the model
The hypervisor runs all of the virtual Machines, the hypervisor needs to be installed - BUT hard to scale!
OS virtualization is similar, but OS in container needs the same kernel as computer OS... Instead of hypervisor, we have docker engine (most popular)

Tools

Tracking experiments in terms of code, dependencies, parameters & dataset!
mlflow (open source) W&B
MLflow tracking every time we run the code, it is logged along with versioning and parametcrs / metric
MLflow projects for organizing and describing the project and code ;
MLflow model standard format for packaging models

Example of model deployment with MLFlow and VMs

1. Push a locally developed MLflow project to Github.
2. Create a VM on Azure and ssh into it.
3. Install mini/conda on the VM.
5. Install MLflow. mlflow
6. Download your model to the VM using the GitHub repository.
7. Open port 5000 (mlflow default) and serve the model on that port.
9. Query the served model from your local machine.

Data Challenges

High-Dimensional Data

Data today have hundreds+ dimensions (features)

Dimensionality Reduction How can we approximate our features in a lower dimensional space? Get rid of redundant features?

Principal Component Analysis (PCA)

Use eigenvectors of covariance matrix to do transformation into most variant directions. Then remove low variance features!

Randomized Models

We lose some accuracy, but win in terms of scalability ♥

Clustering Grouping similar entities together

K-means (Lloyd's algorithm)

Initialise k random cluster means, assign points to closest center, then re-center!

K-means (Bradley-Fayyad-Reina (BFR))

Assumption: clusters are ellipses along axes

Classes: Discard set (summarized in centroid)
Compressed set (summarized, no centroid)
Retained set (NOT summarized)

Summary Statistics: (1) #points
(2) coordinate-wise sum
(3) coordinate-wise squared sum

Locality-Sensitive Hashing (LSH) How do we find similar items fast?

Idea: high probability of $h(q)=h(k)$ given similar items, low probability for dissimilar items ☺

Note: Most distance measures are computationally expensive!

Distributed ML

Ensemble Models

In order to have better generalisation of our final model, we can do:

Bagging: Train models on n bootstrap sets (with replacement)
Reduces variance!

Boosting: Train models, weighing error preds higher
Reduces bias!

XGBoost

$$\text{score} = \frac{\left(\sum_{i=0}^{n} r_i\right)^2}{n + \lambda}$$

(r_i: residuals; λ: regularisation)

eXtreme Gradient Boosting

(1) Decide baseline model (avg.), calculate residuals + similarity score

(2) Build tree, using similarity score (gain)

Remember: high similarity score implies that residuals have same pre-sign!

Predictions: $\underbrace{\text{base} + lr \cdot \text{tree}_1}_{\text{input to tree}_2!} + lr \cdot \underbrace{\text{tree}_2}_{\text{residual trees}}$

Large Data: instead of every split, we do quantiles

Light GBM

Improves XGBoost for high-dimensional data

Gradient-based One-Side Sampling GOSS

(1) Sort and grab top a gradients

(2) Random sample b gradients from rest

(3) amplify small gradients by $1 - \frac{a}{b}$

Exclusive Feature Bundling EFB

a	0	1	0	0	1	
b	1	0	1	0	0	

← (1) establish mutually exclusive features (i.e. 1-hot)

(2) bundle

Reduces Sparse high-dimensional data!

Learning at the Edge

Distributed Systems

- For medium datasets we processed data in batches from disk...
- Distributed systems are used for scalability, fault tolerance & latency
- They are clusters of interconnected machines

Edge Computing

To improve responsetime and save bandwidth
Data is processed closer to the source!

Federated Learning

each device has the model, processes local data, updates parameters and sends parameter updates back to source that will update and redistribute!
But! highly unreliable computing ressource

Workflow:

1. Participant Selection
   which participants will be good in this round?
2. Broadcast – global model is sent out
3. Local Computation – compute on local data
4. Aggregation – aggregate received updates
5. Model Update – updat global model

Local Models

We use a global as a starting point, and fine-tune the model to fit better if local data is non-IID (from different distributions)

CPU, GPU, FPGA, Accelerator

What is Computing gain knowledge/insights, transforming I→O, complete a goal-oriented task!

Processors Makes computing possible. Sends signals with electrons! the brain of a computer - orchestrader

CPU is a sequential processor, i.e. it executes one thing at a time → latency

GPU is a parallel processor, i.e it will execute with higher throughput

Field Programmable Gate Array!
the idea is to use low-level languages in order to avoid translation using computation...

Accelerators only do I→O for the processor.
they are good at matrix multiplication... but!
they are not that flexible...

Big Data Storage

Multi-machine Storage

We store our data on multiple machines... why?

- Scalability - we want our model to sustain variable loads!
- Fault Tolerance - non-volatile system → 1 machine can fail, no prob!
- Latency - more stable processing time → more users = more resources

Coping with load

Scale up: upgrade hardware capabilities of a single machine → vertical

Scale out: we add more cheap machines to our processing → horizontal

shared-nothing architecture (most popular)

Coping with Data

We can replicate the data on each machine or partition it... when partitioning we want to avoid hotspots → one machine taking higher load.

One solution is hash partitioning where we use a hash function to choose machine, but range queries are not efficient.

Effective Rebalancing

Fixed # partitions way larger than number of computers → new computer just gets a partition from current computers

Dynamic partitions

Routing

node knows at which node each key is!

Routing Tier knows and redirects to node!

Client itself knows at which node to access!

Big Data Processing

New tools

we need specialized tools for storage (Distributed File System) and parallel processing

DFS

Google File System has three replicates of each chunk in diff. chunk servers, there is a master containing meta data, and a number of clients, doing the actual processing!

Hadoop Distributed File System

same structure as GFS, but chunks are called blocks!

MapReduce

a programming model; hadoop is an implementation of Map Reduce!

(word1,1)|(word2,1)(word1,1)
(word1,1)|(word1,1),(word2,1)
(word1,1),(word1,1)|(word2,1)
(word1,[1,1])|(word2,[1])
(word1,2)|(word2,1)

1) mapping face → (key, value)
1.2 sort within node
2) shuffling/Merging; all equal keys together
3) reduce list of values into single value

← word count example

Snowflake Hands-On

What is it?

It is a data management architecture that is cloud-based!
Supports users on-demand with varying ressources; reduce "waste" (unused) ressources & reduce congestion/waiting time for employees

MPP

Massive Parallel Processing Architecture
all resources are shared amongst users!

Snowflake

Computing resources in several virtual warehouses with a shared data storage layer...
Resources are dynamic and follow the actual observed workloads; easily scale up/down! We can also ask for faster processing and the architecture can fetch more ressources

Architecture Deep Dive

Cloud: serves the user + store metadata & result cache
Compute: query processing: "the heavy lifting" compute
Database:

Spaaaark! (& Cross-Platform)

Lazy execution Means that we only do minimum amount of work, and only compute the actual data, upon being asked to display it!

MapReduce Is slow due to all the read/write acessing... very time-consoming and ressource-intensive :(

Spark Data is retrieved only once ~10x faster because relevant processed data/results are cached ♡ we can specify .cache()

Cross-Platform We can do traditional coupled processing but sometimes we want to utilize several different systems in an uncoupled set-up!

Multiple Platform Motivation:

- Opportunistic we can decrease costs on platform
- Mandatory we can only run it on specific platforms
- Polystore we have data stored on specific platform

Apache Wayang Wayang is a system that will automatically choose the best platform to execute any given job. Main thing for this functionality is Cross-Platform Optimizer which is the "decision-maker", but also takes care of planning and monitoring the entire proces!

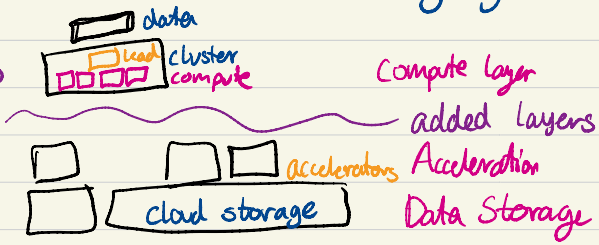
Amazon Guest Lecture ♥

From Germany!

Amazon Redshift

Cloud Warehouse by amazon!
Best performance out there (they say...)

How it started



Performance

- compute node to data node connections implement affinity to have better cache performance... but it has the downside that a partition of data can only be accessed by the connected node!
- Often queries are the same! Thus keep the C++ compilation code in compilation cache ♥
- The customer can define the cluster size to fit size of data! (#CN, Accelerators etc.!)

Autonomics
ML for Performance

How is machine learning used in redshift?

- ML can be used to detect grey errors... i.e. a disk stopped running...
- ML can be used for ressource management by forecasting the expected workload ♥
- ML can be used for warehouse tuning → how do we help people optimize the cluster choices ☺
- ML can be used for workload management !!

Summary

What is big data → how do we implement scalability to allow optimal performance

The ML Lifecycle → preprocessing, models and how to deploy them into pipelines ♥

Model Deployment & Monitoring are relevant... continual learning is a good alternative to monitoring Deploying models to the cloud / on the edge offer each their own pros/cons ☺

Big Data Challenges ML → dendency management and reproducibility are hard to come by on large scale productions Data → privacy, bias and environmental impact for storage

Dimensionality Reduction 3 methods

Learning at scale tree-based methods work well on large tabular Gradient boosting frameworks are optimized for large datasets

Learning at the edge local device train on local data and might tweak main model in the cloud

Hardware some are hard to code, performance varies with type of code / hardware

NNs on GPU NNs can be trained in parallel → GPUs are way more effective than CPUs

Big Data Storage we can Scale up (↓ increase RAM etc.) or Scale out → increase #machines

Big Data Processing HDFS
MapReduce
Hadoop
Sparle

Cross-Platform dataprocessing making different platforms communicate can be hard... Apache Wayang is a cross-platform manager → the first freely available ♥