

Large Scale Data Analysis Exam

BSLASDA1KU

IT UNIVERSITY OF COPENHAGEN

Ludek Cizinsky (luci@itu.dk)

May 20, 2022

1 Pipelines

1.1 Pre-processing pipeline

Figure 1 gives a high level perspective of my preprocessing pipeline.

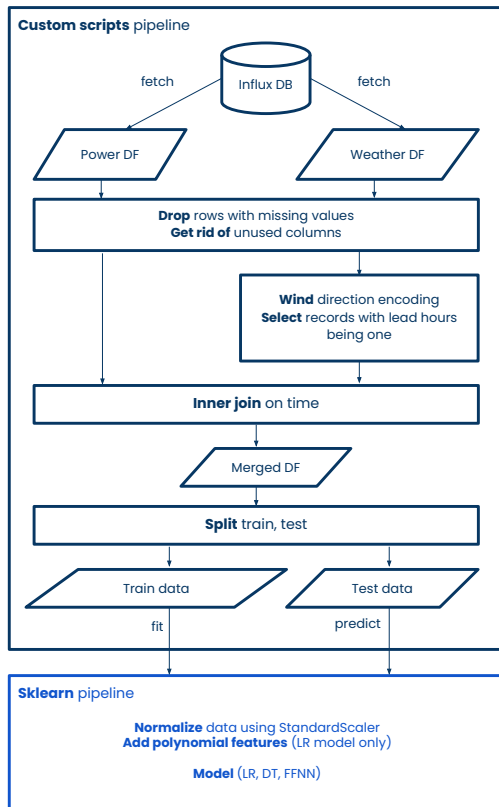


Figure 1: Architecture of the preprocessing pipeline

I splitted the architecture into two main parts: custom scripts, sklearn Pipeline. The goal of custom scripts is to output a single aligned dataset which can then be used for training and prediction using sklearn Pipeline. This architecture proved to be problematic when I simply needed to reuse this pipeline in 3rd assignment since a lot of functionality was implemented in the custom scripts part. Therefore, I would improve the following:

1. Implement all transformations in such a way that they can be part of sklearn Pipeline.
2. Add automatic feature selector to the pipeline to select best features automatically and not based on empirical evidence

A clear benefit of these changes would be easier reproducibility. In addition, some transformers in the pipeline might have certain hyper-parameters, for instance how to align the two datasets. Using sklearn's Grid search, these hyper-parameters could be then considered as part of the search. To summarize, robust pipeline architecture enables the following:

- Easy reproducibility across different environments
- Easier hyper-parameter tuning
- Prevention against leak of testing data into training procedure [doc]

Further, I would like to now zoom into most important details of the implemented pipeline. (note: wind direction is discussed in the section 1.2) First, I have done basic data cleaning: making sure there are now rows with missing values, dropping unused columns. In addition, I used weather forecast made only an hour ahead to have most precise data.

Second, after preprocessing both datasets, the pipeline aligns them using **inner join** on time. More specifically, power dataset had samples from every minute whereas weather data had samples from every 3 hours. The motivation for this alignment is that I want the model to know what is the precise power production given some weather data for that particular time. In addition, this approach performed better based on empirical evidence compare to the other possible approach: group power records into 3 hour groups and then take a mean of power. As a possible improvement, external weather API could be used to get real time data about weather in Orkney. This way, we would not loose significant number of records due to the alignment.

Finally, I normalized selected features such that each feature had zero mean and unit variance - to make the features units independent. In addition, for linear regression, I used polynomial features to increase model's flexibility.

1.2 Wind direction encoding

The information about wind direction was initially encoded as a string where each of the possible values referred to a possible direction. For example 'N' would refer to 'North'. Figure 2 shows possible encodings and their corresponding informativeness.

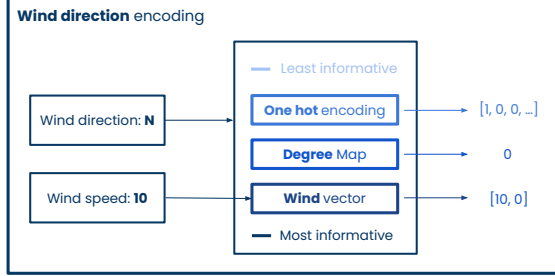


Figure 2: Overview of possible wind direction encodings

The most naive way is to use one hot encoding which simply indicates presence of given direction. Using this approach, we would increase feature space disproportionately to number of samples and as such our model might be over-fitting due to curse of dimensionality. In addition, information about similar and opposite directions is completely lost. Alternatively, we might map each direction to a corresponding degree. The downside of this approach is that for instance degrees 0 and 360 are pointing in the same direction, but from a model perspective the direction is significantly different. To avoid all these drawbacks, I decided to use two dimensional wind vector where each dimension is scaled by corresponding wind speed. Interestingly, thanks to using feature selector as part of my pipeline in third assignment, I found out that in most cases, trained models preferred to select speed and **only one part** of the wind vector.

2 Scalable processing

2.1 Geographic use of authentic language

The goal was to explore whether the use of authentic language at given area is **significantly** higher from our expectation. More specifically, according to my methodology, authentic language is defined as:

A review contains authentic language if it contains either word *authentic* or word *legitimate*.

Note that only reviews associated with businesses tagged as **Restaurant** were considered. Further, I decided to use two definitions of an area:

- state (26 in total)
- city (782 in total)

To account for different number of reviews at each area, I normalized the absolute counts of use of authentic language by the total number of reviews in the given area.

Finally, I set the expected proportion p_0 of authentic reviews to $p_0 = 0.025$. This expectation is based on the sample proportion of authentic reviews relative to all reviews. Therefore, I defined my hypothesis as follows:

- $H_0 : p = p_0$
- $H_1 : p > p_0$

To test this hypothesis, I first used **cube** function on the restaurant reviews ($\approx 4M$ reviews) which yielded sums for all possible combinations of values from the following columns: state, city, isReviewAuthentic. The resulting spark dataframe was saved locally and final computations were done using pandas. Figure 3 shows results when area is defined as city.

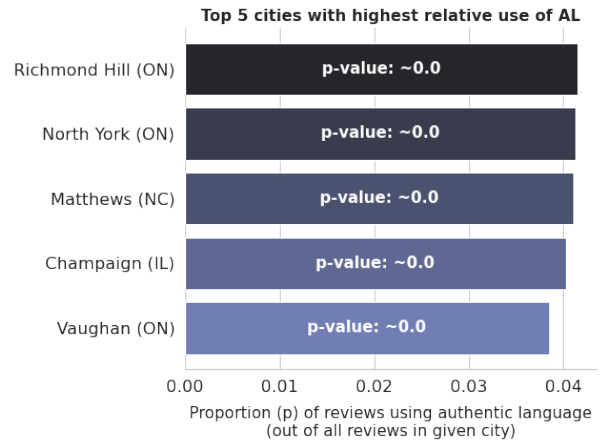


Figure 3: Use of authenticity language at a city level

We can see that for all shown cities, we can reject the null hypothesis in favor of the alternative ones given confidence threshold $\alpha = 0.025$. Therefore, these locations show significant deviation from our expectation and as such, further analysis could be focused on reviews from these locations. Similar conclusions can be made when looking at the results from state level (figure 4).

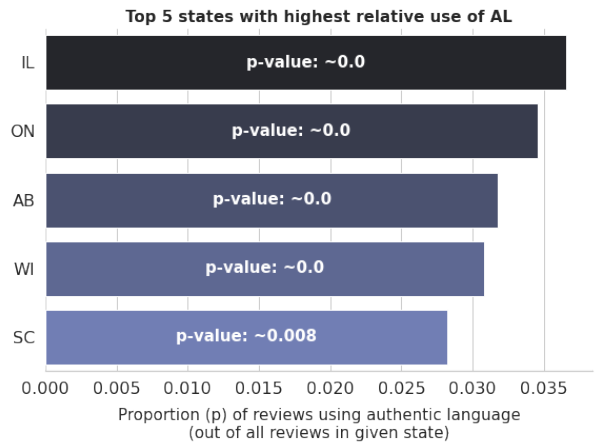


Figure 4: Use of authenticity language at a state level

2.2 Automating hypothesis testing

The goal of the study can be defined as follows:

Test hypothesis that authentic language is used in the reviews more in negative context when it comes to non-western restaurant types.

Automation of such hypothesis testing has several clear advantages:

- Time it takes to reach some viable results
- Amount of reviews that can be processed per time unit
- Re-usability of the hypothesis testing pipeline on new reviews

In summary, the main advantages of automation are speed and scale. On the opposite site, it is important to consider possible drawbacks of such automation. Therefore, I consider all sub-tasks of hypothesis testing and discuss performance of a designed system against human:

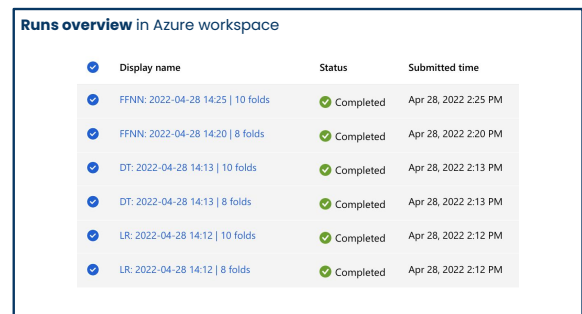
- **Review filtering:** Without a doubt system is superior compare to human assuming we have all the necessary knowledge about reviews: business type, location etc.
- **Authentic language detection:** This is a challenging task even for a human. If we were to use machine learning based system, then we would have to first train the system on examples of authentic and non-authentic reviews. This could be for example using transfer learning, i.e., use a pre-trained language model such as Bert and fine tune it on our specific task. Since the model would be trained on human labeled data, it can be assumed that it would inherit biases embedded by annotators. Therefore, it seems reasonable to claim that the quality of hypothesis testing would not be lowered due to automation since the system would have same biases as the human.
- **Detection in which context is authentic language used:** This is perhaps even harder tasks than the previous one. It would require the model to first understand range within which the authentic language is used and then classify whether this subset of words is positive or negative. From the available reports about state of the art (SOTA) performances of models in the above mentioned tasks [AI], we can assume that human processing would be more accurate.

In the above section, I assumed that we use SOTA methods. In my own study, I used less robust approaches which used rule-based methods to solve above defined NLP tasks. These method's language understanding does not come close to human's language understanding and as such, it can be assumed that we would have to sacrifice quality of the hypothesis testing results in the favor of faster processing.

3 ML lifecycle

3.1 MLflow: experiment tracking

The MLflow framework is designed around the concepts of experiments and runs. Following the documentation, experiment groups runs with a similar objective. [MLfd] Therefore, I decided to have a single experiment - Orkney - power production prediction. Further, I used sklearn's [GridSearch](#) to find optimal hyper-parameters for the given model. Yet, [GridSearch](#) itself has an hyper-parameter which is how many splits should be used during cross-validation. For this reason, each of my runs represents a run of [GridSearch](#) with given number of splits for cross validation. This can be seen on the figure 5.



Display name	Status	Submitted time
FFNN: 2022-04-28 14:25 10 folds	Completed	Apr 28, 2022 2:25 PM
FFNN: 2022-04-28 14:20 8 folds	Completed	Apr 28, 2022 2:20 PM
DT: 2022-04-28 14:13 10 folds	Completed	Apr 28, 2022 2:13 PM
DT: 2022-04-28 14:13 8 folds	Completed	Apr 28, 2022 2:13 PM
LR: 2022-04-28 14:12 10 folds	Completed	Apr 28, 2022 2:12 PM
LR: 2022-04-28 14:12 8 folds	Completed	Apr 28, 2022 2:12 PM

Figure 5: Azure workspace view of my runs

Finally, for each run I first logged info about my preprocessing and EDA steps including one figure. Once my code reached the grid searching part, I used MLflow's auto-logging feature developed specifically for sklearn. I chose this option due to its simplicity as well as its functionality which logged all the information I needed including the model with best parameters found by the grid search and best model's performance on chosen metrics: r-squared (R^2), mean absolute error (MAE), mean squared error (MSE), maximum error (ME).

In addition, I could also inspect in detail performance for each possible combination of hyper-parameters. These were logged as a csv file which I then loaded as a pandas dataframe and analysed it further in a jupyter notebook. Figure 6 shows my analysis of the performance (R^2) of the best grid searched models.

3.2 MLflow's support of reproducibility and scaling

First, I would like to define my understanding of a reproducible ML system:

A reproducible ML system is a system which I can run in any environment and I will be able to obtain the same results independently of the given environment.

In particular, MLflow supports this in the following ways:

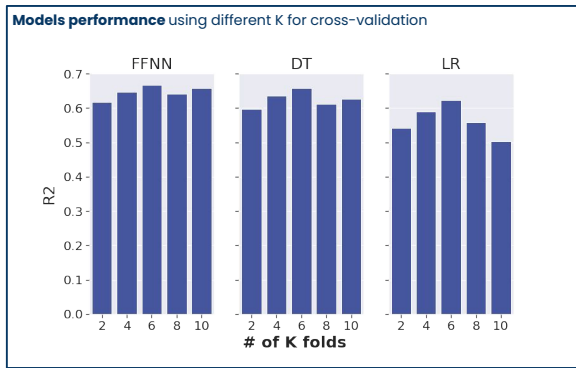


Figure 6: Cross validated performance (R^2) of my models, each bar represents different run with given K

- **Tracking API:** enables to track every detail of design of the given ML system. For instance, during inspection of the code which I want to reproduce, I might have accidentally changed some of the hyper-parameters. Thanks to the logs of previous run, I can retrieve these parameters and set them back.
- **Projects:** provides standardized packaging format. [MLfc] For example, it might be used in academia where it is critical for other researchers to reproduce given paper's results.
- **Models:** provides standardized model format. The core idea behind this format is to enable easier deployment of the model in different deployment environments. For instance, I built my model locally in sklearn, then I can package it into a Docker image using MLflow. [MLfb] Finally, I can use for example Digital Ocean's platform as a service to easily deploy this image. Deployed model's results can then be reproduced by anyone who has access to the REST API of the model.

Based on the MLflow's docs [MLfa], scalability is supported in the following ways:

- One can execute the given ML pipeline within a **cluster**. In addition, within each node, computations can run in **parallel**. For example when running grid search, this feature can speed up the whole execution significantly.
- The above described execution power can be used on big data since MLflow enables its users to read and write to distributed file systems.
- Finally, MLflow offers **Model registry** which is supposed to serve as a central hub for model storage and their querying across the company. This becomes especially useful as the company and number of model grows.

4 Lecture material

4.1 Serialisability and linearisability

Serialisability is a guarantee that concurrent execution of transactions (groups of operations) will return the same DB state as if they had been executed using serial schedule. Consequently, serialisability ensures isolation property of transactions, i.e., changes of one transaction do not impact the other transaction. ([LBB18], [Kle21])

Linearisability provides a guarantee that any operation on any single object within the system will take an immediate effect. [Bai] Note that object can be viewed as a single object accessible by multiple threads within one node or a single object replicated across several nodes in a distributed system.

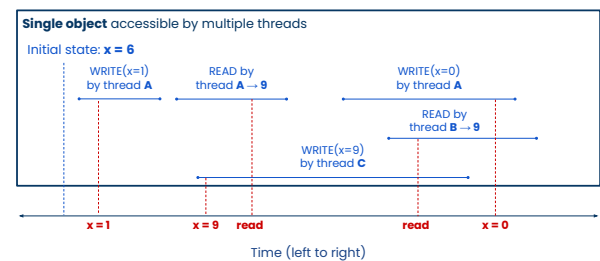


Figure 7: Example of linearisable schedule inspired from [Kle21]

Figure 7 shows linearisability in a practical single node scenario. Despite read by thread A started earlier than write by thread C, the read value is 9 and not 1 since the actual write happened earlier than the read. Similarly, thread B reads also value 9 since at the particular point of reading this is the most recent value.

To summarize, serialisability is applicable to group of transactions which can operate on a single or multiple objects whereas linearisability is **only applicable to a single object**. [Bai] As described in the original paper ([HW90]), linearisability is a local property - if each object in the system satisfies the linearisability, then the system as a whole is linearisable. In contrast, serialisability is rather a global property. Further, linearisability is a non-blocking property - same object can be accessed by multiple invoked operations. Whereas serialisability is due to its definition blocking property. Finally, serialisability is more suitable in the context of DBMS, in contrast linearisability is better suited for multi-processor systems.

4.2 XGboost

4.2.1 Framework summary

In general, XGboost is a machine learning framework which can be used to make predictions for all kinds of tasks. Its main power compare to other machine learning frameworks such as scikit-learn is concealed in its usage of available computational resources within given environment allowing it to process even very large datasets signif-

icantly faster compare to other frameworks. In addition, the framework can be used both on a single node as well as in a distributed setting. [CG16]

XGboost stands for extreme gradient boosting. Gradient boosting is an ensemble machine learning technique which was already introduced in 2001 by Friedman. [Fri01] The core idea of the paper is to combine predictions of several weak learners in such a way that as a whole they achieve great performance. Weak learner in this context can be any supervised machine learning model, the original XGboost paper uses decision trees, this is also a default setting in their library. [CG16] Each decision tree in the ensemble is trained based on the mistakes of the previous tree. For instance, in regression, these mistakes can be quantified as residuals, i.e., difference between given tree's prediction and true value.

4.2.2 What makes XGboost scalable

First, for small datasets it is viable to use greedy algorithm to find an optimal split for each node:

1. iterate over all M features
2. for each feature iterate over all possible splits

However, for larger datasets, it becomes quickly infeasible to inspect all possible splits. Therefore, XGboost only uses **quantiles** as potential split candidates. In addition, one can choose whether these quantiles should be determined globally, i.e. once at the start and then used throughout the training, or locally where these quantiles are re-computed after each split. Further, computing quantiles for large datasets is also an expensive operation. Therefore, XGboost uses an approximation algorithm called **weighted quantile sketch**. ([CG16], [GK01])

Second, XGboost's general philosophy is to avoid unnecessary reads from disk since these are expensive. For this reason, it utilizes **cache** in order to make common computations such as node splitting fast. [Ast22a] Yet, in many cases, it is not feasible to fit the entire dataset into memory. For this reason, XGboost also focuses on optimizing **out-of-core computation**. Data is first divided into blocks, these blocks are then compressed by columns and saved to disk. Thanks to the **block compression**, reading time of data from the disk is reduced, yet certain time needs to be spent on the decompression. This is an acceptable cost since overall the reading time is reduced. In a distributed setting, when multiple disks are available, **block sharding** can be used. This increases the overall throughput of the system. [CG16]

Third, gradient boosting is by definition a sequential algorithm, therefore the parallelization can happen only when training each tree in the ensemble. More specifically, it happens when finding optimal split of a node and considering M possible features, i.e., M threads can run in parallel to yield the optimal split for each feature. [Fan15]

Last but not the least, in real world settings, data are quite often sparse. This can be due to many factors, for example due to missing values or one hot encoding. To

solve this issue, a default direction is learnt during building a tree. Therefore, when the decision needs to be made for a record which is missing given feature, the algorithm progresses through the tree based on the learnt default direction. [CG16]

4.3 Discussion of SaaS, PaaS, IaaS

4.3.1 SaaS

SaaS refers to software as a service. This concept became very popular in recent years since it allows organisations of all sizes to outsource certain software needs to external companies which specialize in the given domain. For example, as a startup, I might need a communication platform which has to have certain features. One clear option is to build my own communication platform from scratch. The advantage is that my startup does not depend on any other company, therefore it can for example prioritise development of new features according to its own needs. On the opposite, it will cost my startup certain engineering resources which could otherwise be dedicated to the development of the startup's own product. Therefore, from a finance perspective, this option becomes overtime expensive since the platform needs to be not only developed but also maintained. On the opposite site, paying monthly or yearly subscription fee to a company which in return provides me with their **software as a service** (e.g. Slack) might be way better option not only from a finance perspective, but also from the engineering resources point of view - no need to develop or maintain anything.

4.3.2 PaaS

PaaS stands for platform as a service. We can view it as going by one abstraction layer lower relative to the SaaS. In other words, we want to focus on developing our own software and spend minimal effort deploying it. This can be for many reasons, for example not having the necessary engineering skills or simply because the time spent on maintaining and setting up the deployment pipeline could be better utilized by engineers by working on developing new product's features instead. An example of such platforms are Heroku or Google Cloud's service called Run. These platforms provide its users with well structured documentation and examples on how to deploy all kinds of applications within minutes. Last, but not the least, to make the whole deploying even smoother, it is often possible to trigger the deploy procedure by a new commit to the main branch of the given GitHub repository. PaaS is a good option for development agencies which then hand the application to their clients whose technical team might be small and as such does not have resources to deal with DevOps.

4.3.3 IaaS

IaaS stands for infrastructure as a service. Again, we step one abstraction layer lower relative to PaaS. In this particular scenario, the service provider gives us access to the

infrastructure, which usually refers to compute, storage, networking etc., but their setup is up to us. IaaS can be used by company of all sizes, which shows another great feature of cloud computing in general - scale as you go. A great example is Slack which uses AWS infrastructure services such as EC2 (compute) since 2009 when it was a small startup until today when it is a publicly listed company. [PR15] This also shows how valuable IaaS provided by AWS is for Slack since despite its growth it has not switched to its own infrastructure. (on-premise)

4.4 Big data challenges

4.4.1 Need of custom ML algorithms

One of the common challenges across ML algorithms is that they were not designed to be ran neither in parallel nor distributed settings. One of the clear advantages of embedding these settings requirements into the design process of modern ML algorithms is clearly the reduced processing time, but not only that. Article from 2019 by MIT review [Hao20] also shows impact of training deep learning neural networks on production of CO₂ - training these large models requires substantial amount of energy and as a result significant amount of CO₂ is produced. A possible solution is redesign of already known algorithms or their customization. In this paper, I have already discussed in detail XGboost which extends the classical gradient boosting in such a way that the algorithm can be ran in both parallel and distributed settings. Another great example is HOGWILD! [Niu+11] which proposes a way how to make Stochastic gradient descent run in parallel. This is an impactful paper in a sense that many nowadays ML algorithms are using SGD as an underlying optimization technique to tune model's parameters.

4.4.2 Privacy and Federated learning

Privacy of user data started to be a concern in the past several years mainly due to the growth of possible usages of data. This increases system requirements and adds an additional complexity to their design. Further, the more sensitive data given company has access to, the more critical it becomes for it to handle it correctly. Google is a good example of a company that has access to a lot of sensitive data ranging from personal health data to private messages. Up until to 2017, all Google's customer data were stored centrally across their data centers. Therefore, also model re-training happened there. For certain use cases, such as data from mobile interactions, this was a sub-optimal solution. In addition, also the central model re-training was an expensive operation. Therefore, they introduced a novel approach called **federated learning** (FL) . [17] A core idea of this approach is that the data does not need to leave users' devices. More specifically, model re-training happens locally on user devices and the newly found parameters are then send to the central data storage where they are merged into a global model. This model is then distributed back to all devices. While FL

is a novel approach, it introduces many new challenges. [Che22] As an example, one of the hyper-parameters in FL is how many participants, i.e. edge devices, should participate in each round of the central model update. In the context of Google with a global user-base, this can easily lead to model biases if for instance only devices from the US would be used.

4.4.3 Data streams

In many real world scenarios, data can be viewed as a continuous stream of information. One of the challenges of working with stream data is that quite often they can not be stored to a DB and analyzed at some point later. Instead, it is necessary to do certain preprocessing steps right after the data reach the system. In addition, there is no constraint on how many streams of data there can be and of what nature they are. For instance, one stream might produce image data and the other textual data. Therefore, this means that the system has to be capable of alignment of the data. To address these and many other problems related to stream data, frameworks such Apache Kafka or Apache Flink were introduced. [Ast22b] The latter mentioned framework was for example used recently in a paper introducing a novel way of text polarity labelling of massive data streams. [Wu+22]

5 Favorite concept: Apache Spark

Apache Spark is framework for large scale data analysis in a distributed setting, yet it can be also used locally. As described by [Kal21], It was created as an improvement of framework called Hadoop, more specifically, it provides high level APIs in several languages including Python and R. The core concept around which Spark is built is called resilient distributed dataset (RDD). In general, when you load dataset into memory, it is stored across several nodes using this data structure. Further, Spark is using lazy evaluation. More specifically, no computations are ran until it is necessary. This is enabled by saving all the actions and transformations performed onto the given RDD into a computational graph (directed acyclic graph). The advantage of lazy evaluation is that every time a sequence of transformations is performed on a given dataset, Spark creates a new copy, i.e. changes are not made in place. Therefore, lazy evaluation significantly reduces the space needed. [fou] Last but not the least, Spark abstracts away the work with RDDs via its high level API. Therefore in practice, a client is interacting with a dataframe.

Spark is important in the context of large data analysis because of its ease of use. Therefore, as a data scientist, I can focus on the actual data science work. This is also confirmed by use cases of Spark in the industry and academia: eBay, NASA, Stanford and many others. [Apa]

References

- [HW90] Maurice P Herlihy and Jeannette M Wing. “Linearizability: A correctness condition for concurrent objects”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12.3 (1990), pp. 463–492.
- [Fri01] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: [10.1214/aos/1013203451](https://doi.org/10.1214/aos/1013203451). URL: <https://doi.org/10.1214/aos/1013203451>.
- [GK01] Michael Greenwald and Sanjeev Khanna. “Space-Efficient Online Computation of Quantile Summaries”. In: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’01. Santa Barbara, California, USA: Association for Computing Machinery, 2001, pp. 58–66. ISBN: 1581133324. DOI: [10.1145/375663.375670](https://doi.org/10.1145/375663.375670). URL: <https://doi.org/10.1145/375663.375670>.
- [Niu+11] Feng Niu et al. *HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent*. 2011. DOI: [10.48550/ARXIV.1106.5730](https://arxiv.org/abs/1106.5730). URL: <https://arxiv.org/abs/1106.5730>.
- [Fan15] Zhanpeng Fang. May 2015. URL: <http://zhanpengfang.github.io/418home.html>.
- [PR15] AWS PR. *Slack Case Study*. 2015. URL: <https://aws.amazon.com/solutions/case-studies/slack/>.
- [CG16] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785). URL: <http://doi.acm.org/10.1145/2939672.2939785>.
- [17] *Federated learning: Collaborative machine learning without centralized training data*. Apr. 2017. URL: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>.
- [LBB18] Wilfried Lemahieu, Seppe Vanden Broucke, and Bart Baesens. *Principles of database management: The Practical Guide to storing, managing and analyzing big and small data*. Cambridge University Press, 2018.
- [Hao20] Karen Hao. *Training a single AI model can emit as much carbon as five cars in their lifetimes*. Dec. 2020. URL: <https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/>.
- [Kal21] Amir Kalron. *Hadoop vs. spark: A head-to-head comparison*. Nov. 2021. URL: <https://logz.io/blog/hadoop-vs-spark/>.
- [Kle21] Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O’Reilly, 2021.
- [Ast22a] Maria Sinziiana Astefanoaei. *Algorithms for large scale data: XGBoost, Locality sensitive hashing*. May 2022.
- [Ast22b] Maria Sinziiana Astefanoaei. *Streaming data*. May 2022.
- [Che22] Niels Ørbæk Chemnitz. *Federated learning*. May 2022.
- [Wu+22] Huilin Wu et al. *A Framework for Fast Polarity Labelling of Massive Data Streams*. Mar. 2022.
- [AI] Meta AI. *Natural language processing*. URL: <https://paperswithcode.com/area/natural-language-processing>.
- [Apa] Apache-foundation. *Powered by spark: Apache spark*. URL: <https://spark.apache.org/powered-by.html>.
- [Bai] Peter Bailis. *Peter Bailis :: Highly available, seldom consistent*. URL: <http://www.bailis.org/blog/linearizability-versus-serializability/#fn:implementation>.
- [doc] sklearn docs. *6.1. pipelines and composite estimators*. URL: <https://scikit-learn.org/stable/modules/compose.html#pipeline>.
- [fou] Apache foundation. *RDD Programming Guide*. URL: <https://spark.apache.org/docs/latest/rdd-programming-guide.html>.
- [MLfa] MLflow. *Concepts*. URL: <https://www.mlflow.org/docs/latest/concepts.html#scalability-and-big-data>.
- [MLfb] MLflow. *MLflow models*. URL: <https://www.mlflow.org/docs/latest/models.html#built-in-deployment-tools>.
- [MLfc] MLflow. *MLflow projects*. URL: <https://www.mlflow.org/docs/latest/projects.html#projects>.
- [MLfd] docs MLflow. *MLflow tracking*. URL: <https://www.mlflow.org/docs/latest/tracking.html#concepts>.