Exam Assignment

Large Scale Data Analysis - Spring 2023

Mie Jonasson - miejo@itu.dk

June 2, 2023

B.Sc. Data Science - Spring 2023

Large Scale Data Analysis - BSLASDA1KU

IT UNIVERSITY OF COPENHAGEN

1 Pipelines

1.1 Describe the models you included in the pipeline and compare their performance. (10%)

I chose to do the same set of preprocessing steps for all my models. First I encoded the wind direction and speed into a two-coordinate vector. Then I applied standard scaling to my 3 features; speed, wind x & wind y. I chose to still include the speed on its own, as I figured it might offer some insights on its own. Of these steps, the standard scaling is particularly important, as some of the regression models we will try out are sensitive to features being on different scales. I reserve the last 20 days out of 90 days total as validation data.

The first regression model I used was a simple least squares linear regression. I use this model as a baseline model to be compared to models of higher complexity. It was found during exploratory data analysis that the relationship between a feature and power output had at least a polynomial relationship of degree 2, thus we know that a linear regression might not bee complex enough. This model is solely implemented for comparison purposes, as well as to establish basic notions of feature importance.

Ridge Kernel is a linear regression model with a kernel, that is fit using regularization. For this particular data set I chose to use a polynomial kernel, and tweaked the hyper parameter describing the degree using the R^2 on validation data. In the end, a polynomial kernel of degree 3 was chosen as it was the one with highest R^2-score on validation data. The model is fit through least squares regression on the kernel features with l2-regularisation, which adds a squared penalty term for the coefficients, in order to make them as small as possible. In practice, we are adding a term to the loss so we will not only minimize squared error, but also the model parameters themselves.

Decision trees are mostly used for classification, but can be defined for regression tasks as well. The main difference between the two types of the decision trees, is the measure defining which split is 'best'; for classification tasks we use an impurity measure, while we use the greatest loss-decrease in regression problems. I tuned the value of the hyper parameter `'min_samples_leaf'` which describes the number of samples in a leaf. After tuning using R^2-score, I ended up with a value of 320.

A decision tree is a high variance and low bias model, that tends to overfit to the exact patterns of the training data, if nothing is done to avoid this. Linear Regression is, on the other hand, a low variance and high bias model. It is an oversimplified model for our data. Ridge Kernel regression can be tuned to fall somewhere between the oversimplified and complex model. For all these models it depends largely on regularisation parameters and hyper parameters whether they overfit or underfit. This is why hyper parameter tuning is an important step in model selection, and regularisation parameters are implemented for many complex models.

Once i had all 3 trained models, i compare both their R^2 scores and their residuals. The scores for the three models were 0.75, 0.80 & 0.81 on validation data. All of these scores are relatively high, and were similar to those obtained from the training data, indicating that we have not overfit to the data. The decision tree scores the best but is only slightly better than the ridge kernel regression. We see that our baseline linear regression classifier has a lower R^2-score, but does not have as low a score as one might expect.

On the other hand, when we view the residuals in Figure 1, we see that there are some patterns in the residuals indicating that the true relationship we are trying to model is of a higher complexity (higher degree/level of detail) than our models. Particularly we see this pattern be most clear for the linear regression and less clear as the models become more complex.

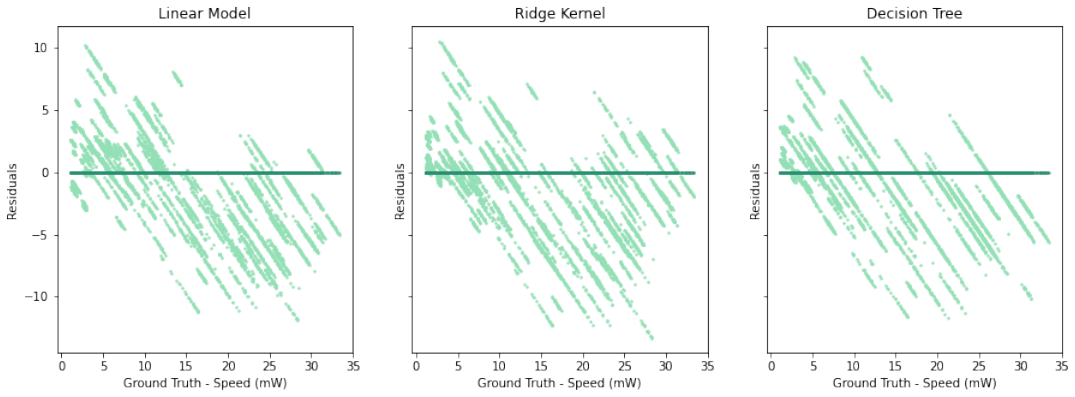

Figure 1: Scatter Plots of Predicted Value against True Value

1.2 Explain why using pipelines is useful in this context - you can also refer to work done for other assignments. (10%)

A pipeline is an object that contains a series of steps which includes, for example, transformations and scaling. All pipeline steps in an sklearn pipeline [1] need to have a `'fit'` and a `'transform'` method that can be applied to the data, and might optionally end with a predictor that implements a `'predict'` method instead of `'transform'`. The advantages of pipelines are manifold, but for this context i will focus on the advantages in terms of reproducibility, distribution and retraining.

Pipelines are particularly useful in the setting of creating reproducible results, since we are able to pickle (read: 'save') the fitted pipeline into a file that can later be loaded and used. This eases both the distribution of models and reduces human error when setting up code and including all the steps in the exact same order when testing.

Particularly for distribution, it is relevant to have an object that can simply take the data in raw form and produce predictions without any additional commands or preprocessing steps before passing the data. This is relevant for the case of packaging & serving models, as was part of the ML Lifecycle assignment. In the assignment we used MLFlow to package and serve our models, mimicing a real life case of training a model to be used by the public. In this case, packaging the model as a whole, from taking in raw data to producing a prediction, was an imminent part of creating a model to be used by the wider public, as the model should be able to take in a single line of input (features in their raw form) and produce a prediction for the user.

When doing retraining and testing of models, using pipelines can ensure that train and test data are put through the exact same steps in preprocessing and prediction. This both makes the results comparable as well as ensures experimental integrity of the produced outputs. It also ensures that transformers and scalers are fitted to the training data and thus only *applied* to test data, as this might in some cases be non-trivial for users to do. It is an important point, as the integrity of the experiment relies on test data being unknown to the model during training / fitting.

Since we can save an entire pipeline, it also comes in handy during retraining, as scalers and transformers are also fitted with a set of parameters that might change over time due to data shift. If data shifts are present, we will be able to automatically compare an old fitted pipeline to a new fitted pipeline on the most recent data, and save the new one if it out-performs the previously saved one on test data.

2 ML Lifecycle

2.1 Justify what you logged during your experiments and show the results. (15%)

In order to perform model selection, when trying to model a relationship between features and a target feature, it is often relevant to examine quantitative loss and performance measures. There exists an abundance of measures to pick from, but the most used for regression problems are Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 score. I chose to focus on R^2 due to this measure being less volatile towards outliers, but I still logged MSE and MAE to see if these measures show some of the same trends.

I chose to examine the three models Ridge Kernel (polynomial kernel), Support Vector Regression (polynomial kernel) and a decision tree. The hyper parameters I decided to tune in my experiments where `'degree'` for the polynomial kernels and `'min_samples_leaf'` for the decision tree. As I wanted to keep track of these experiments within one place, I log both the model name along with the hyper parameter I am tuning using `mlflow.log\_param(<parameter_name>, <parameter_value)`. Further, after fitting the pipeline on training data, I log the three metrics described in the above paragraph, evaluated on test data.

To perform model selection i created a function that would take a model class, a parameter input and the data for training and testing. The parameter input should consist of a parameter name and list of values to test. Defining a function automated the majority of the process, and I just call the function once for each model and hyper parameter tuning combination (i.e. once per classifier, since we are tuning one parameter). I named all of my runs, to make them easy to find in the log, following the convention; *'<model_name> with <parameter_name> = <parameter_value>'*.

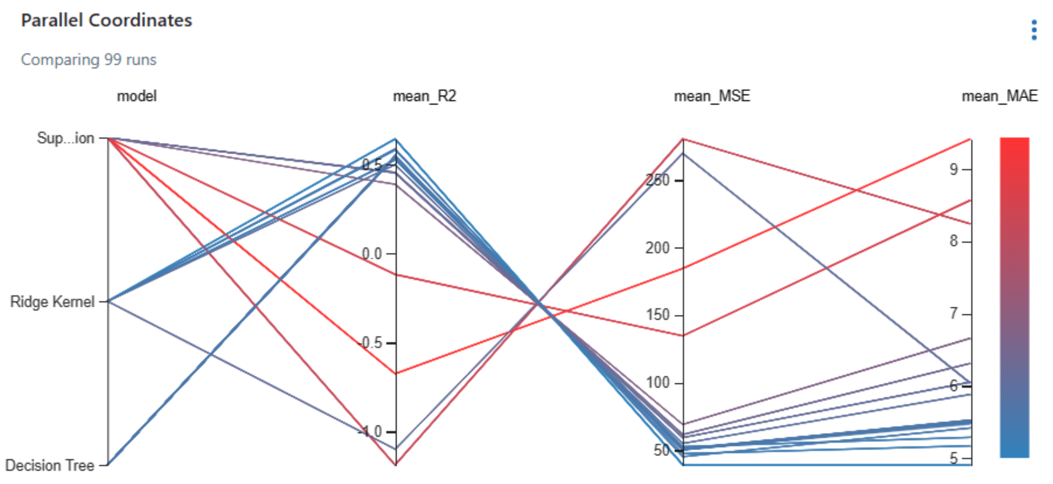

Figure 2: Logged Metrics of Different ML Models on Test Data

The results of the experiments can be seen in Figure 2. It is clear that the best performing model is Ridge Kernel with a polynomial kernel of degree 3, as this is the model that scores best on all three evaluation metrics. It is closely followed by the same model with degree 2 and decision trees. An interesting observation that might argue against choosing the Ridge Kernel of degree 3 is that the Support Vector Regression scores a negative R^2 for the same degree, meaning the the true relationship of the data might not follow a polynomial of degree 3. We also see that there is one experiment with a high MSE but low MAE. This is due to MSE squaring terms, and thus being more sensitive to outliers than MAE.

2.2 Describe possible causes of data/model drift in the context described in the assignment; explain how using MLflow can help identify this. (10%)

When we model a regression problem, we model the distribution of a target value given a set of features. Several things might change over time - both the features and target variable follow a distribution that might encounter changes. It is hard to predict whether a particular model will encounter a data / model drift, and the main remedies are monitoring, retraining or adding more training data. We want to find whether the model performs well in the real life application and update it if performance decreases.

When we look at possible causes in the context of our wind power data, we can consider both changes in the input or output. If we talk about changes to the input, some seasons might inherently have more windy periods, and this would require an update to the model to take into account the different distribution of the features. Further, one might consider that improvements might be made to the way the produced energy is sourced, stored or transported, in turn creating a greater power output from the same wind input. This can be considered a change in the relationship between feature and target value, and will require training a new model. Each of these cases would require our model to be updated in order not to lose it's predictive power and good performance. First, we would need to discover that a data drift is present.

The setup using ML Flow to track our experiments when retraining makes it possible to discover data drifts. When we monitor the performance of models it is a passive approach to combat data drifts, as we are not actively doing something to prevent it. On the other hand we consider continuously retraining and updating the model an active approach, as we are actively counteracting the possibility of having a decrease in performance due to model / data drift by updating the model.

Through our logs we can see whether models that used to be established as the best ones perform worse on new data, indicating that the hyper parameters of the true relationship has shifted. An example could be that it might shift from a kernel of degree 3 to one of degree 2 (a simpler and less complex model) over time, as discussed shortly in Section 2.1. Another example is that the distribution of features might change, with a need for updating the coefficients for scaling of the features.

In practice, ML Flow is particularly helpful in terms of monitoring and updating models, as we can keep all of our training results in one place along with packaged models. In our project we logged our results to the ITU remote server, to mimic having a non-local server for tracking training / retraining results. This automates a lot of the process to ensure that we maintain a log of all previous results for future analysis.

3 Scalable Data Processing

3.1 Explain how you counted how many reviews have a variant of the word "authentic".(10%)

In order to analyze authenticity language, we needed to find a way to establish whether this type of language is present or not. Essentially, we wanted to examine whether the text associated with reviews contained some form of authenticity language.

I went about it by using a Spark DataFrame containing all the review data, and creating a temporary spark view from it. This allowed me to query it in SQL, and I formulated a query following the below syntax:

```sql
SELECT count(*) as total, sum(auth) as w_auth, sum(auth) / count(*) * 100 as pct
FROM ( SELECT CASE WHEN {auth_string} THEN 1 ELSE 0 END as auth
    FROM reviews)
```

my query is formulated as an f-string with `auth_string` being defined in a variable in the Jupyter Notebook. The variable is defined as a boolean, built using a list of words - returning true if any of the words from the list were present in the text (non-case-sensitive, as I made sure to lowercase everything). This made it feasible and more automated to keep track of the set of words / subwords I was testing for, since I could add another word to the list and run the notebook, and all results would be updated.

In the end i had the following list; `['authentic', 'legit', 'credible', 'original']`. This list could easily be optimized or have additional subwords added. For example, i used `'legit'` as it is the shortest form of the word *legitimate* and would also capture slang and other word forms. This notion might also be relevant to apply to other words currently in the list, along with more investigation into synonyms that we might want to consider in the context as well.

I found that approximately 1.78% of the review texts contained one of these words / subwords. This is not a high percentage rate, but considering that there are almost 7 million reviews in the dataset, it amounts to more than 100 000 reviews. Another thing to mention is that the dataset is not solely for restaurants, but also consider, for example, hotels and bars. When doing filtering of businesses based on their categories (as described in Section 3.2), we ended up with per-cuisine percentages ranging 3-6%.

3.2 How did you test the hypothesis? Include your results. (10%)

We want to test the proposed hypothesis, that authenticity language is being used to alternately describe a good or bad experience in different types of cuisines. I started by performing some data cleaning and transformation, since I wanted to solely obtain data on restaurants and also to enable grouping businesses by cuisine.

First I did filtering using where-clauses to remove non-restaurant businesses. For establishing cuisines I had a list of keywords / categories for each cuisine and had a case with five different booleans mapping to each their own cuisine. After cleaning and tranforming the data, I had only restaurant businesses and an added cuisine attribute with the following categories: `['Asian Cuisine', 'South American Cuisine', 'Middle Eastern / African Cuisine', 'European Cuisine', 'American Cuisine', 'Unknown']`

I saved the filtered businesses and their cuisines into a new view, and started examining the reviews for these businesses. I chose to count for each cuisine in each state, how many reviews contained authenticity language, how many contained negative words as well as the number of reviews that had both types of language (using methodology similar to code-snippet in Section 3.1).

Once i had the counts i examined different fractions that would enable me to create a reasonable ranking of the cuisines. I settled on the fraction $\frac{\#\ authentic \cap negative\ reviews}{\#\ negative\ reviews}$, which described the fraction of negative reviews also containing authenticity language. I used the below syntax for obtaining a ranking of cuisines for each state based on the fraction:

```sql
RANK() OVER (PARTITION BY state ORDER BY fraction DESC)
```

I used rank instead of the fractions themselves to avoid bias of some states using more authenticity language and/or more negative language than others. Using the rank has the down-side of losing information about the difference in percentage points, as these differences might be slight or huge, but I settled on using rank as its benefits out-weighted the apparent down-sides. Using the above code a lower rank means more authenticity language in negative reviews while a higher rank indicates less.

In order to examine whether authenticity language is more prevalent in negative reviews of non-western restaurants as opposed to western restaurants, I decided to do a two-sample t-test with a one-sided alternative hypothesis. A two-sample t-test tests for the means of two distributions being equal, meaning that the hypothesis we are testing follows:

$$H_0 : \mu_{n\_west} = \mu_{west}$$

$$H_A : \mu_{n\_west} < \mu_{west}$$

We decide on a threshhold of 0.05, meaning we will reject the null-hypothesis (H_0) if the p-value produced by the t-test is below this. I use `statsmodels.stats.weightstats.ttest_ind` with a one-sided alternative hypothesis, and determine that the variance of the two populations are unequal. The test produced a p-value of 0.0042. This is below our threshhold, and thus we reject the null-hypothesis and accept the alternative hypothesis. In practice, this means that we accept and corroborate that negative reviews for non-western cuisines contains authenticity language more often than western cuisines.

3.3 Describe the steps you took to extract features for the rating prediction model you implemented in the last part of the assignment. (10%)

For the last part of the assignment we wanted to find a way to model the number of stars given in a review based on features of the review itself. The first thing I did was split the data into a train and a test set. I did a split with 90% in the training set using `randomSplit([0.9, 0.1], seed=12345)`. I specified a seed to ensure that my results would be the same every time i run the code.

The first features I chose to use were statistics of the business being reviewed and the user writing the review. I added review count and avg. stars for each of the two (as provided in their related tables). I also wanted to test whether the use of authenticity language or negative words had an impact on the model parameters, and thus added the 2 boolean columns 'auth' and 'bad_review' as were used and defined in the previous tasks (review subquery in code-snippet in Section 3.1). This will allow me to test whether use of authenticity language and / or negative words impact the prediction of stars given for a review. The last thing I included for testing were the state for which the business is located. Since state is a categorical variable, I had to encode it by using a one-hot-encoder, which creates a zero-vector in the length of the number of unique categories and encodes a 1 at the index of the desired category.

When we input a Spark DataFrame to one of it's MLLIB regression models, the model requires the input to be in a particular format. Features should be concatenated in a column under the name `'features'` and the target variable should be contained in a column named `'label'`. For this purpose i created a pipeline concatenating all the preprocessing steps. Due to the nature of Spark's modules, it is only possible to use a one-hot-encoder on indexed (integer) columns, and thus my pipeline for preprocessing had three steps: `Pipeline(stages=[ColumnIndexer, OneHotEncoder, VectorAssembler])`. The first two steps are used to encode the categorical variable `'state'`, while the last step takes all 7 feature columns and gather them in a single column following the naming for input to MLLIB models.

If there had been more time and ressources dedicated to the task, further work might have benefitted from applying language modelling on the textual data. An example of this might be applying a BERT based pre-trained language model to extract embeddings on the text of the review, and use the produced embeddings as features to the regression model. Previous work have shown great performance increases when using BERT for sentiment analysis tasks, which models positive and negative sentiments, specifically in the review domain [2].

4 Lecture Material

4.1 Write a MapReduce program in pseudocode that computes the average salaries per department. Explain each of the steps in your pseudocode and demonstrate an example run. (8%)

MapReduce is a Framework for Big Data Processing, applying a user-defined map and reduce function to large chunks of data in parallel. The implementation of both the map and reduce ensures that items can be processed in parallel, as they are only applied to a single data item at a time. For the mapping we get a data record at a time and output a single key-value pair. For reducing we get a key at a time, with all associated values in a list and output the key and some type of aggregate / summary of the given values. Our example dataset contains records with the following attributes: `id, name, salary, department`.

```
Map(DataFrame data_record):
    key = data_record['department']
    value = data_record['salary']
    emit(key, value)

Reduce(String Key, Iterator <Integer> values):
    tot_salary, salary_cnt = 0, 0
    foreach value in values:
        tot_salary += value
        salary_cnt += 1
    average_salary = tot_salary / salary_cnt
    emit(key, average_salary)
```

In the above pseudo code, I have defined a map and reduce function for the task of determining average salary by department. The map function takes a data record and extracts department as the key and salary as the value. Then, for the reduce function, we get a key (department) and a list of values (salaries). We sum over the list of values and count the number of items. This allows us to calculate the average value (i.e. the average salary) for the given key (department), which we return along with the key.

ID	NAME	SALARY	DEPARTMENT
1234	John Doe	21350	CS
1235	Jane Doe	21450	CS
1236	Johnny Doe	20650	HR

Table 1: Table of example-run data points

We use the example data point along with two self-invented data points to do an example run. The data points are shown in Table 1. After passing each of the records through the map function, we have three data points on the form: `{CS, 21350}, {CS, 21450}, {HR, 20650}`

The next step in the framework is shuffling the intermediate results. In order to produce a single entry for every key we merge data points with equal keys. After shuffling and merging we have the following data points: `{CS, [21350, 21450]}, {HR, [20650]}`

At last the data is passed through the reduce function to produce average salaries per department as requested. The output for our example run is: `{CS, 21400}, {HR, 20650}`.

4.2 Similarly to the example shown in class, build the XGboost tree. You only need to find the first 2 levels of the tree (as in the example from class). Explain your work. Explain why XGBoost is suited for big data. (7%)

When we initialise an XG Boost tree, we start by predicting to an arbitrary value. I chose to do the initial prediction at the mean of our data points. The approximate data points from the assignment description can be seen in Figure 3. Our data points are $[(15,7.5),(20,9),(30,-10),(35,-7.5)]$, and the mean of the second coordinates of each point (and thereby our initial prediction) is -0.25. This gives us the residuals: $[7.75,9.25,-9.75,-7.25]$. In order to determine where to split the data points when building our tree, we use the below similarity score.

$$score = \frac{(Sum\ of\ Residuals)^2}{\#\ Residuals\ +\ \lambda}$$

We will ignore the regularisation parameter λ in our simple example. Our root model gets a score of $\frac{0^2}{4} = 0$. We determine the splits based on the gain in similarity score, meaning we test every possible split and compare them on $gain = SIM_{left} + SIM_{right} - SIM_{root}$. We want to test three different thresholds for the first coordinate of the data points. We choose these thresholds to be in the middle of two adjacent points, and thus we get; $(17.5, 25, 32.5)$. For each of these splits we use the residuals to calculate the gain, and we get the following results: $(60+20-0, 144.5+144.5-0, 17.5+52.6-0) = (80, 289, 70.1)$. From this we clearly see that using the split at 25 yields the biggest gain in similarity score, and thus we will use this threshold to split our points.

To establish another level in the tree, we look at each child-node containing more than one residual. This means we will test for each of the subsets: $[7.75, 9.25], [-9.75, -7.25]$. We note that there is only one possible split left for each, 17.5 & 32.5 respectively. For the left sub tree we get a similarity gain of $60+85.6-144.5 = 1.1$, and for the right sub tree we get $95+52.6-144.5 = 3.1$. Both of these are positive numbers, and are at the only possible splits in each leaf, thus we will split both leaves. The final XG Boost tree for the two-level first iteration is shown in Figure 3.

Building XG Boost trees is done over several rounds, where the model, for which we find the residuals and build a tree, is the output of the initial prediction and previously built trees multiplied by a learning rate. Essentially, we are learning by modelling the errors of the initial model, in order to learn the patterns of the data. XG Boost is well-suited for big data, since it has an application for which it builds trees in parallel on partitions of the data. Once it has computed the XG Boost trees on each partition, the partitions are then used to approximate a histogram for the totality of the data. XG Boost still has limitations, as it is not efficient on high-dimensional data, especially lacking options for performance on sparse data (f.ex. One-Hot-Encoded features).

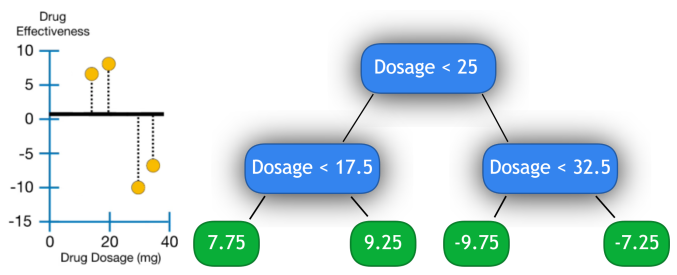

Figure 3: The data points and the corresponding XG Boost Tree

4.3 Summarize the findings of the articles discussed in class in relation to privacy in the context of big data. (5%)

Privacy is an important topic to consider whenever we deal with any type of data, and big data can offer some different challenges than other types of data. In class we particularly discussed the concept of 'anonymising' data, considering whether a collection of data points can ever be truly anonymous, without the opportunity to specifically identify an individual in a crowd of data points.

We discussed the article *Temporal and cultural limits of privacy in smartphone app usage* [3]. The article attempts to establish whether random sampling of used apps can be used to uniquely identify an individual, and found that it was possible to re-identify an individual in 1 of 3 cases by sampling just 5 apps. Additionally, if popular apps were excluded from the random sampling, a massive 90% of individuals could be re-identified with just 3 sampled apps!

We further discussed how this extends to more than just app useage, as discussed in *Unique in the Crowd: The privacy bounds of human mobility* [4]. The dataset contains mobility patterns of 1.5 million users, obtained by viewing which antenna their phone would ping during useage at any given point in time. In this setting, random sampling 4 spacio-temporal data points from an individual allowed re-identification of 95% of subjects.

The main take-aways when it comes to 'anonymous' data is that truly anonymising data is a near impossible task, even for datasets containing millions of entries. If anonymity of data subjects is a desired quality, measures should be taken to keep the amount of data on each individual to a minimum, along with avoiding to share any volatile/identifiable information without due consent and/or reason.

It should be noted that claiming data subjects to be anonymised upon sharing data, is not as easily obtainable a quality as often phrased by corporate officers and data retention companies. Further considerations in terms of data retention periods and keeping storage private and safe from hacking are likely to make an impact on privacy in terms of data only being used for the scope consented to by the data subject.

4.4 Explain the hash partitioning technique used in distributed systems and illustrate it with an example. Discuss its benefits and drawbacks. (5%)

When storing datasets that are too large to fit or perform well on a single drive, we want to find a way to partition the data in a way that ensures that all drives are utilized equally. It is important for performance and storage space that the partitions contain similar amounts of data, even as new data flows in. One technique used for this is hash partitioning, where each data item is hashed using a key and distributed into partitions. Here, the even distribution of keys relies on deciding on a good hash function. Having the power to establish a hashing function amounts a possibility to evenly distribute skewed data as well.

If we consider a simple dataset with 4 data points whose ids are $(0,1,2,3)$, and we want to divide them equally into 2 partitions. The simplest method for this type of sequential numbering is to calculate $id\%n$ as the hash key, where n is the number of partition nodes. Using modulo-2 for the partitions p_0 & p_1 over the sequence of ids will generate the hashing keys $(0,1,0,1)$, which is the equal distribution we wanted to obtain from hashing. Following this partitioning, p_0 would contain the entries with keys $(0,2)$ and p_1 the keys $(1,3)$. A visual run-through of this example can be seen in Figure 4.

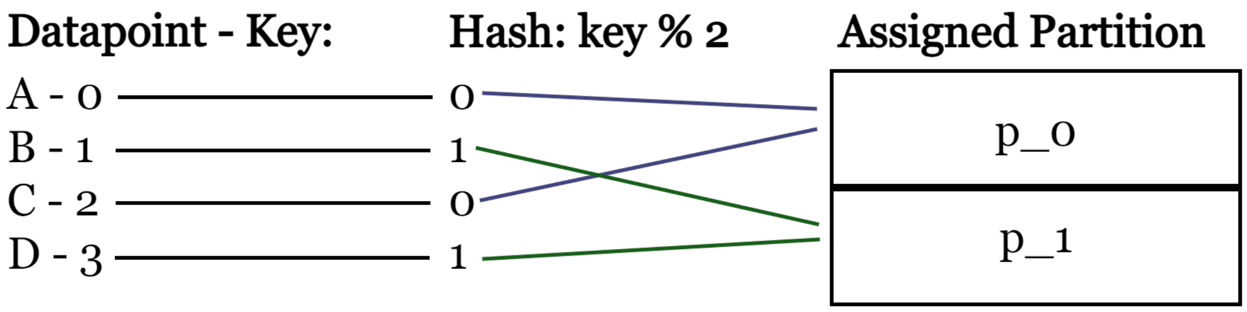

Figure 4: Visual overview of small hash partitioning example

Partitioning in general is beneficial, as it allows access and storage of data in a way that allows for scaling systems up and down, depending on query workload and amount of data stored. It also makes the setup less volatile to individual machines failing. Hash partitioning is particularly fast for point-queries, as one would not need to search through the entire dataset, but solely the partition for which the hashing key is located. Hash partitioning further has the added benefit of (usually) having evenly distributed data, inherently also distributing query workload to improve average performance.

A big drawback for hash partitioning is that it is inefficient on range queries. In order to obtain every data point in the range, one would need to hash every key in the range to find the partitions containing the queried data. Often data will be scattered throughout partitions, as per the goal of equally distributing data across partitions, and thus access to many different machines would be needed.

5 Bonus Question (5%)

In one lecture we were taught about cross-platform frameworks, allowing users to navigate the vast amount of big data / ML tools, in a field that is constantly evolving. Several topics throughout the semester have been about particular applications and optimizations, and I was amazed upon learning about a framework allowing us to utilize different platforms in a single application. Often platforms are specialized in one or few tasks, e.g. a data storage platform with fast fetching of data might not be optimized for fast transformations / training a ML model.

Generally there exists three main reasons for doing cross-platform data processing, all of which have their validity. The opportunistic approach is focused on utilizing different platforms to get the best price and/or performance, whilst there are also more practical concerns. The practical concerns are either of type mandatory, where a single tool does not have the needed capabilities, or polystore, where data is stored on several platforms that need to be communicate. When we have large scale data, these motivations might arise more often, as performance and/or price of processing vast amounts of data easily worsens if no particular optimizations are used.

Cross-platform frameworks solve the problem of users having to navigate how to make systems interact, using systems that cannot simply be plugged in together. It is important to help users get the most out of their money and allow them to obtain the best performance possible with their set of tools, without needing to maintain complicated pipelines that might need to change as the tools change. Using a framework like Apache Wayang helps the user in planning the execution of tasks, considering both price and performance.

Apache Wayang is the first open-source cross-platform framework available. We will, no doubt, see the cross-platform frameworks evolve further in the future, as data processing capabilities on large scale data is in increasing demand! A 2020 study [5] even shows that companies with good Big Data Analytics Capabilities (BDAC) gain a market advantage over other companies, as they utilize the information they gather in meaningful ways to boost their services and marketing. Companies are becoming increasingly aware of this advantage, and thus are motivated to explore tools for increasing their capabilities, in turn further increasing the demand for cross-platform capabilities in the future.

References

[1] sklearn.pipeline.pipeline. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.pipeline.Pipeline.html

[2] S. Alaparthi and M. Mishra, “BERT: a sentiment analysis odyssey,” vol. 9, no. 2, pp. 118–126. [Online]. Available: https://doi.org/10.1057/s41270-021-00109-8

[3] V. Sekara, L. Alessandretti, E. Mones, and H. Jonsson, “Temporal and cultural limits of privacy in smartphone app usage,” vol. 11, no. 1, p. 3861, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41598-021-82294-1

[4] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, “Unique in the crowd: The privacy bounds of human mobility,” vol. 3, no. 1, p. 1376, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/srep01376

[5] P. Mikalef, J. Krogstie, I. O. Pappas, and P. Pavlou, “Exploring the relationship between big data analytics capability and competitive performance: The mediating roles of dynamic and operational capabilities,” vol. 57, no. 2, p. 103169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378720618301022

6 Declaration

I hereby declare that I have answered these exam questions myself without any outside help.

Mie Jonasson *June 2, 2023*