MASTER IN

DATA ANALYTICS FOR BUSINESS

MASTER'S FINAL WORK

INTERNSHIP REPORT

Towards MLOps in a Startup Company

Emil Henricsson Ene

March-2023

MASTER IN

DATA ANALYTICS FOR BUSINESS

MASTER'S FINAL WORK

INTERNSHIP REPORT

Towards MLOps in a Startup Company

Emil Henricsson Ene

SUPERVISORS:

JOÃO AFONSO BASTOS

VILLE ROTO

March-2023

Abstract

Companies which have developed around a product or service built on top of one or several machine learning applications will reach a point where the complexity of these applications become unfeasible to manage manually. To approach a more mature level of their machine learning applications these companies must start implementing MLOps in their operations as well as orchestration tooling for building durable and scalable machine learning pipelines. This paper presents how a startup company in the marketing- and market research sector has taken their first steps towards machine learning maturity by implementing MLOps practices and orchestration with Apache Airflow.

***Keywords*—** MLOps, Automation, Machine Learning Pipelines, Orchestration, Apache Airflow, DAGs, Parallelization

Contents

1 Introduction

Data is growing. By 2023 this is certainly not an original observation, as data and it´s impacts on businesses has been acknowledged for many years already. Nonetheless, the global market for data is still estimated to grow steadily in the coming years and is reported to reach a value of over $420 billion by 2027 (Boso et al. 2021). The effects of this increased market value has implications in many aspects of our world, such as new academic programs focused on skills needed to leverage this data, or companies which are founded solely on the plan of capitalizing on this data.

As a student enrolled in the Msc in Data Analaytics for Business at ISEG, I have had the opportunity to internship at just such a data-centric company called BrandDelta. As many companies which have founded their business models on leveraging data, BrandDelta is challenging an old idea with a new data-based approach. By building a service that utilizes big data analytics, BrandDelta has taken a disruptive position in the marketing- and market research sector.

While BrandDelta already had a running version of their service at the start of my internship, my responsibilities has been centered around improving- and optimizing the data processing foundations for this service. Specifically, BrandDelta wants to mature and automate their machine learning applications to reduce technical debt and possible breakpoints in their IT-infrastructure. This report will present the tools and processes used for approaching machine learning maturity at BrandDelta.

1.1 BrandDelta

BrandDelta is a small startup originating in the UK, with an office in Porto, Portugal. BrandDelta offers market research and marketing analyzes for consumer-goods companies in various industries such as foods and lifestyle. The work carried out by BrandDelta is typically done for a specific brand in one or several markets, or for a group of brands of an umbrella organization. The services provided by BrandDelta are focused on gathering and capturing consumers attitudes towards the client's brands, with the goal of generating new valuable insights for the client. These clients can range from small domestic brands to large international organizations (BrandDelta).

1.2 Internship

During my time at BrandDelta I worked as a data engineer intern. Data engineering is not a traditional science, but the field has seen a strong growth in recent years as more and more companies try to leverage data in their operations. Tasks and technologies related with this role can vary greatly between different companies, or even between different data engineers. Generally speaking, activities focus on creating infrastructure that allows for processing-, storing- and making data available, such that it can be consumed easily by other users.

BrandDelta suggested that internship tasks would focus on the following list of areas they wanted to improve:

- Create and maintain data pipelines.

- Understand internal data supply need and provide data accordingly.
- Develop quality control procedures.
- Optimize data management processes.
- Deploy current data and machine learning processes into cloud environment.
- Deploy and maintain machine learning models in production reliably and efficient.

The master's in Data Analytics for Business prepared me well for this internship and the tasks it entailed. Although I had to learn many new technologies and ways of working almost every day of the internship, the master program prepared me with the foundations that enabled this continuous learning and development.

One of my biggest tasks at BrandDelta has been the last bullet of the list above. I was given the responsibility to lead the deployment of BrandDeltas current machine learning models to allow them to operate in an automated and predictable manner. This reflects the maturity BrandDelta is working towards, where improving the efficiency and reliability of their machine learning operations is key. This project will also be the focus of this internship report.

1.3 Internal Data Workflow

BrandDelta provides their clients with marketing insights through machine learning- and AI models, this means that the entire business relies heavily on data and data analysis. The structure of the company can be organized into three different divisions.

The first layer is the *data engineering* division, whenever a new or existing client has requested an analysis, the process starts with data engineering extracting the relevant data. Data comes from multiple sources, such as web data available through scraping, data retrieved through APIs of different vendors and data that has been provided by the client themselves. Once all relevant data has been extracted, the data engineering team also transforms and aggregates the different raw datasets into one or several processed datasets, ready to be consumed by the next layer.

The second layer is the *data science* division. For existing clients, this layer already contains models that have been trained for their data, so for these clients pre-trained models only need be run on the new data to produce the wanted outputs. For new clients however, models must be configured to fit their specifics (language etc.), trained and evaluated before any outputs can be analyzed. Once the relevant models have been successfully trained and utilized, the outputs of the models are forwarded on to the third and final layer.

The final layer consists of the *analysis* division. This is where the model outputs get transformed into actual insights that analysts can present and discuss with the clients to make strategic decisions about their future operations. This final layer is much more dependent on a human touch and business acumen than the two previous ones, therefore this layer is also the most difficult to automate.

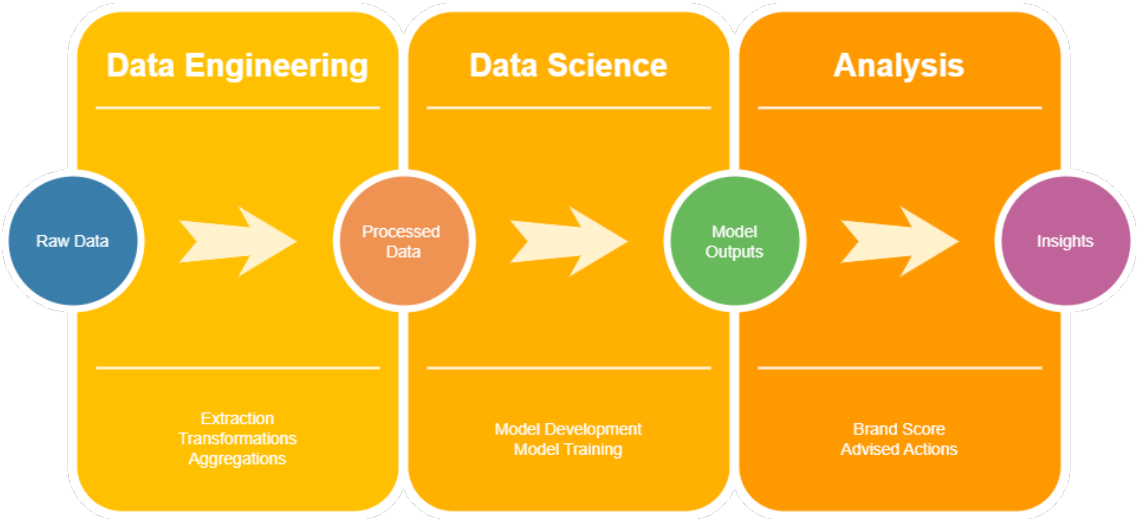

Figure 1: BrandDeltas data flow

The data engineering layer, the division I was assigned to during my internship, and the data science layer contain many tasks that are easily automated. In a perfect world, we could use software tools to completely automate these two divisions' daily tasks so that these employees could focus solely on building new and better services. However, that is not case in the real world as software and workflows will need to be reconfigured or modified eventually. In this paper, we will see how some of the daily data engineering- and data science tasks at BrandDelta can be automated to save engineers and data scientists a lot of work and support quicker deliveries to the client.

1.4 Maturing Machine Learning in BrandDelta

Machine Learning Operations, or MLOps, is a topic that describes the collection of technologies and engineering practices used for building data science- and machine learning applications in a scalable way. As stated by Kreuzberger et al. (2022), MLOps aims to find solutions to issues arising once a machine learning system is taken into a production stage. A machine learning system is often one part of a product with several functionalities, and this system must be fast and robust enough to run in synchronization with the other functionalities of the product. Although machine learning has revealed new opportunities for companies who can leverage it, many ML-projects fail when brought into production, since companies have not yet created the required infrastructure and workflows to facilitate these new complex systems. Therefore, MLOps has been of increasing interest for companies in the last years. However, the field has yet to mature in the scientific world, and there is no real consensus on what "best" MLOps consists of (Kreuzberger et al 2022).

From an enterprise perspective, MLOps is the continuous assessment and effort to maximize the ROI of your data science and machine learning investments. For a young company like BrandDelta, whose whole business-model depends on data, machine learning and data analysis, devel-

oping good MLOps practices at this time while the company is in a stage of growth can help prevent future headaches and provide better services for all their clients. For the rest of this paper, we will examine the latest research in MLOps, the underlying technologies that MLOps utilizes, how a company like BrandDelta can implement these processes and finally we will simulate how a classic machine learning problem can be automated using these processes.

2 MLOps

According to Antonini et al. (2022), research advancements in the field in the latest years has presented MLOps as a seven-step looped framework, see Figure 2. We will present each step briefly and see which steps are relevant for BrandDelta's MLOps-maturity. We start in *Plan*, this is where the data science team explores the data to see what possible insights could be derived from one or several models. In the *Create* step, the data science team builds- and trains the relevant models. In *Verify*, data scientists check model performance and assures that the final proposed model behaves appropriately. At this stage in a large enterprise, an engineering team usually takes responsibility for the following steps. In *Package*, engineers rebuild or adapt the model to be compatible with the environment it will be deployed in. In *Release* and *Configure* the engineering team makes sure that the model works correctly in the deployment environment and configure eventual runtime parameters. Finally, in *Monitor* both engineering and data science take responsibility, the former team monitors system performance while the latter team monitors model performance.

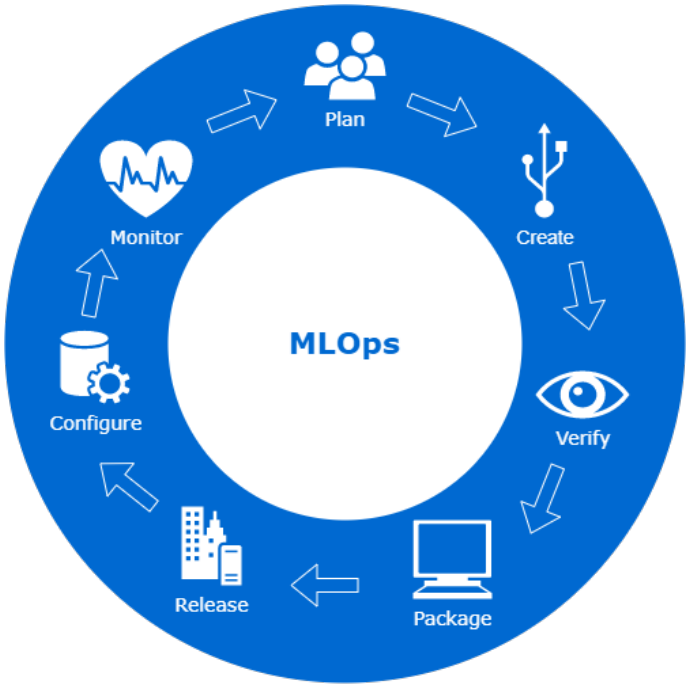

Figure 2: MLOps as a seven-step framework

As BrandDelta has not matured to the stage of hiring machine learning engineers to package, release and configure their models when they go into production, a lot of this work falls jointly on the data science- and data engineering divisions. Moving forward, we will discuss how BrandDelta can implement the MLOps framework. To implement a new framework like this, a company will most likely undergo iterations of integrating parts of the framework, one step at a time. At this time BrandDelta is specifically interested in creating good practices for the steps between Creation/Verification and Monitoring, so this paper will focus on these steps and the technologies needed for automating them.

2.1 Machine Learning Pipelines

One of the most fundamental constructions for controlling and automating large and complex machine learning systems are pipelines. A machine learning pipeline, or ML-pipeline, is an end-to-end infrastructure wherein a ML-model is deployed, maintained, and activated to produce some outputs. As pointed out by Xin et al. (2021) the life-cycle of a ML-model is often perceived in the simplified textbook case where data is used to train a model, validate it through some performance metric and then used to create some predictions or other outputs. However, this is an oversimplification of the complexity that ML-models create once they are deployed in a real application. Data scientists and ML-engineers often perform extensive feature engineering on data before it is passed to the predictive model, ML-pipelines also often consist of chained models, where one or several individual models create features for the final predictive model.

Whether a ML-pipeline or another sort of data centric pipeline, we can think of these pipeline objects in a visual way as graphs of nodes and edges. Specifically directed graphs, where data is passed between nodes according to determined directions. At each node, some operation is performed on the incoming data and the output is forwarded on to the next node, in the final node the output is the model predictions (Xin et al. 2021). An example of this is showed in Figure 3. This is the simple textbook example where the raw data is used to train models, validate them to find the best performing one, and finally produce outputs with the final prediction model.

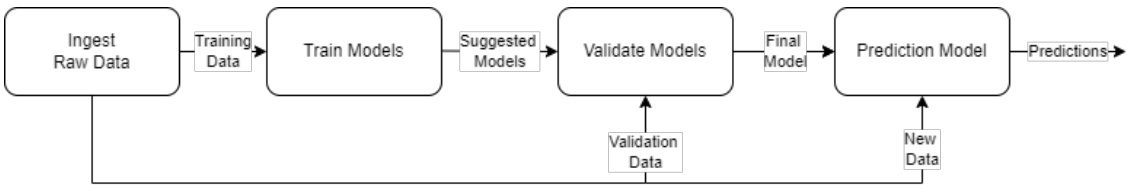

Figure 3: Textbook ML-pipeline example

The ML-pipeline is an essential concept for several steps in the MLOps framework. Mapping out and building pipelines allows for automating both the Creation and Verification of a ML-model, as new data comes in we can both retrain and re-validate the model on the new data by

simply activating the pipeline. Data scientists can also Monitor models by analyzing predictions at the end of the pipeline, made with new incoming data.

2.2 Automation

Once we have a ML-pipeline conceptualized we can start building it, where each node is a separate script or file performing some operation, as we saw above. Since we want this entire pipeline to run when activated, meaning we want tasks to activate sequentially after one another, we have to set the correct edges between the nodes. This is done by setting dependencies, where for example the Train Models-node in Figure 3 is dependent on the Ingest Raw Data-node to finish successfully before starting. This triggers the nodes sequentially and allows for automating the operations contained in the pipeline.

Another important automation aspect for MLOps, as Garg et al. (2021) describes, is implementing CI/CD-processes (Continuous Integration and Continuous Delivery) in these pipelines. This allows the pipelines operators to be updated instantly if their functionalities have to be changed. This is essential for keeping pipelines- and their operators up to date, and for not taking applications offline each time some operator or dependency needs to be reconfigured. Implementing CI/CD-processes is essential for the Package-, Release- and Configure steps of the MLOps framework, and can greatly relive engineers' workloads.

Automating pipelines with dependencies and CI/CD processes also allows for continuous retraining of model parameters. This is an important benefit of pipeline automation as the input data used for making real time predictions might change compared to the data that was used for training the prediction model. This is known as data drift and can cause major performance degradation in the models' predictions (Garg, et al. 2021).

The final step of our MLOps-framework, Monitor, also benefits from being automated. Rukat et al. (2019) states there are many methods for monitoring a deployed models' predictions. Warnings can be generated and sent to the data science team automatically each time some performance threshold is not met. Another more proactive approach is to make some statistical tests of similarity on one or several features of the ingested raw data compared to the old raw data used for training. Again, if certain thresholds of similarity are not met, warnings should be sent automatically to the data science team. Implementing such automated testing processes would further improve the reliability of the machine learning system.

2.3 Parallel Computing

As we have mentioned above, the pipeline in Figure 3 is a simple example, pipelines tend to grow much larger in an actual application. An important tool for efficiently operating increasingly large pipelines and computational tasks is the ability to scale up your compute power. Bekkerman et al. (2011) recounts how several technological advances have facilitated the efficient use of increasingly large datasets for producing more accurate machine learning models. As datasets

grow so does the number of computations needed to train a model or to make predictions with that dataset. Large enough datasets will eventually render a model unfeasible if these computations become too time consuming.

One solution to this issue is parallelizing these computations, this means that repetitive computations of the same sort are carried out in parallel. This works as long as each instance of the computation is independent of the result of any other instance. If this requirement is met, we can run such computations concurrently using independent computing units, these units are often called nodes or workers and represent computing hardware such as CPUs or GPUs . In Figure 4 is is an example of how matrix multiplication can be executed with parallelization. Instead of calculating the $j * n$ elements of the new resulting matrix sequentially in one computing unit, we can instead scale up our computing power to $j * n$ workers who all calculate their element concurrently.

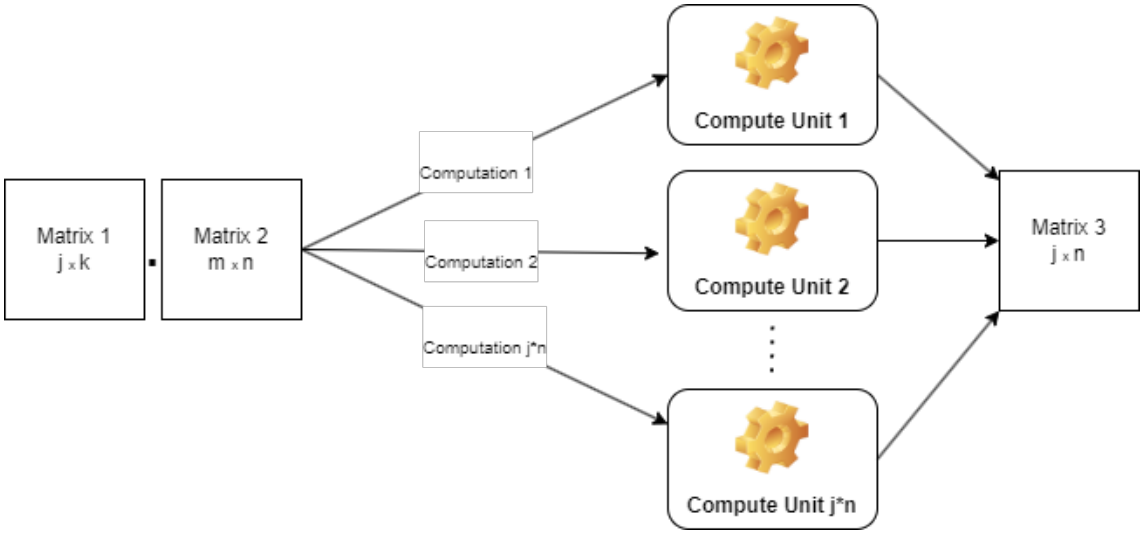

Figure 4: Example of parallelized matrix multiplication

Utilizing parallelization can be applied to larger objects than specific computations as well, such as parallelizing model training or even entire pipelines. A great opportunity to use parallelism in machine learning is in parameter selection and model evaluation. Several different models can be trained and evaluated using the same data, since no information needs to be carried between models they can be trained and evaluated concurrently (Bekkerman et al. 2011). For BrandDelta, especially the parallelization of parameter- and model selection will be of importance, as the ML-pipeline in our application will utilize this for concurrent hyperparameter search.

3 Orchestrating Pipelines with Apache Airflow

To build and automate these pipelines discussed above, there are many separate parts that need to be organized to run together. A technology used for achieving this, with a fitting name, is an *orchestrator*. An orchestrator is, simply put, some software providing the underlying architecture

needed to connect and execute operations according to the dependencies between them, and in so creates the graph-structured pipelines we presented earlier. There are many tools that achieve this, such as Apache Airflow, KubeFlow, Prefect, and many more (Matskin et al. 2021). For BrandDelta, it was decided to go with *Apache Airflow*, due to existing skills in this tool among the team members.

Airflow, described by R. Mitchell et al. (2019) as a workflow management solution, is a free- and open-source tool to build, order and schedule workflows. The workflow of interest for us is the ML-pipeline, where each node in the graph is a computational operation on data, each nodes' operations are defined in python files, one file for each node. In Airflow these graphs are specifically *Directed Acyclic Graphs* or *DAGs*, this ensures there is no possibility of a pipeline infinitely looping over itself. Building these DAGs is done by setting dependencies between the nodes´ operators, building the correct dependencies is also defined in a python file. From here on out the abbreviation DAG will be used interchangeably with the word pipeline.

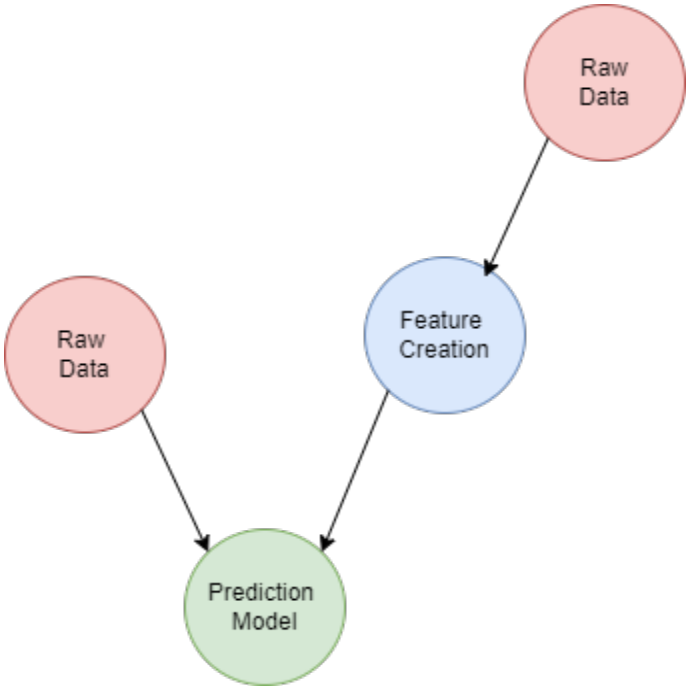

Figure 5: Simple ML-pipeline in DAG representation

The DAG illustrated above is another example of a simple ML-pipeline using a pre-trained model. New data can be passed through a feature creation model and then continue to be used by the pre-trained model, together with some other features of the raw data, to produce outputs. Clearly, following the dependencies in the arrows connecting the nodes, there can be no cycles in this DAG.

Airflow consists of many parts, and a detailed explanation of each one of them is beyond the scope of this paper, but a brief introduction follows here. Airflow has its own meta-database, which contains information on each DAG and its history. Airflow uses a web-hosted user interface where engineers can execute and monitor DAG runs and see where a pipeline failed. There is at least

one worker, which is the compute power used to run the pipeline and its operations, computing power can be increased by scaling up the number of workers. Finally, there is a scheduler and a triggerer, which schedule operations in correct order and trigger them when their dependencies are met. All of these different parts of Airflow are individual pieces of software which need to be built and configured. These parts can be custom built but thankfully the Apache Airflow community provides extensive support in setting up the initial Airflow instance.

One of the most common ways of setting up Airflow is with Docker containers. Docker containers are a way of packaging software and all it's configurations and dependencies in a way that allows it to run on top of any base environment. Figure 6 below is provided by the official Docker website, for our case each App running on top of Docker is instead replaced by each of the Airflow pieces. This means that each piece of Airflow can be packaged in a container and easily be shared between different users (Airflow). This is especially important for the Package- and Release stages of maturing MLOps at BrandDelta, since we want different users to be able to run our ML-pipelines without having to look for- and install hundreds of dependencies when setting up Airflow in their machine.

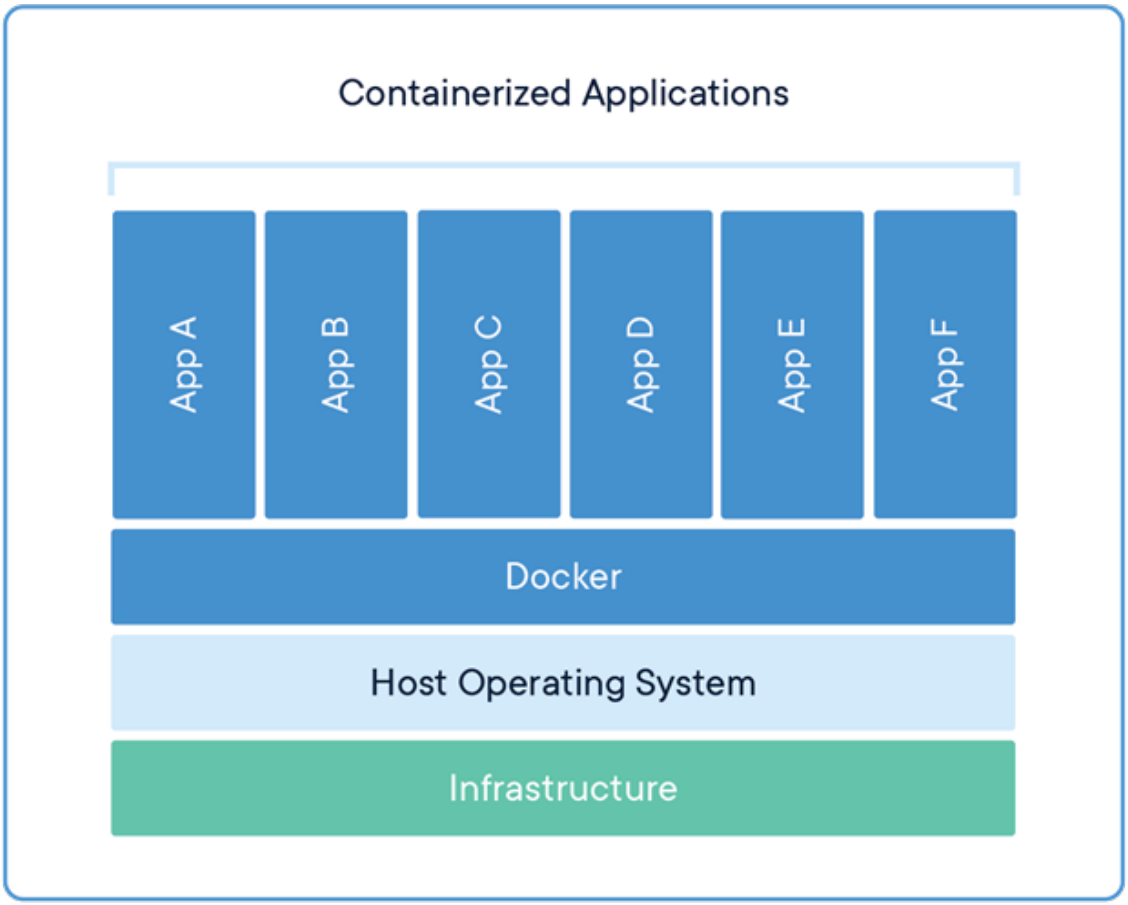

Figure 6: The concept of Docker containers

4 Application of Model Optimization in Pipeline

BrandDelta uses lots of textual data as raw data, which means that many of the ml-pipelines contain one or several models for Natural Language Processing (NLP), some of the NLP models are used for transforming- or creating new features and some calculate the final metrics delivered to clients as well. Training these models involves optimizing several parameters, which in turn is done by separate optimization algorithms. One such optimization algorithm is gradient descent and in the following section we will see an example of how gradient descent is used in an ML-pipeline similar to the ones used in BrandDelta, and how BrandDelta can incorporate more mature MLOps processes to further automate this pipeline and improve their machine learning services.

4.1 Application

As mentioned earlier, ML-pipelines tend to grow more complex in real world applications than compared to the ones presented in academic textbooks. Below is a replicate of a part of the whole pipeline used by the data science division at BrandDelta. The graph represents the first cleaning- and feature creation tasks performed on a new incoming text-dataset. Each of the orange branches extending from the central node in Figure 7 resembles a simple textbook ML-pipeline, some of which the models need to be retrained with the new data. This makes the final insights metrics dependent on hundreds of nodes. Several NLP models are used already in this part of the process for creating features, and the outputted dataset is forwarded onto several succeeding models, the whole pipeline ends with the calculations of the final insight metrics (not visualized here).

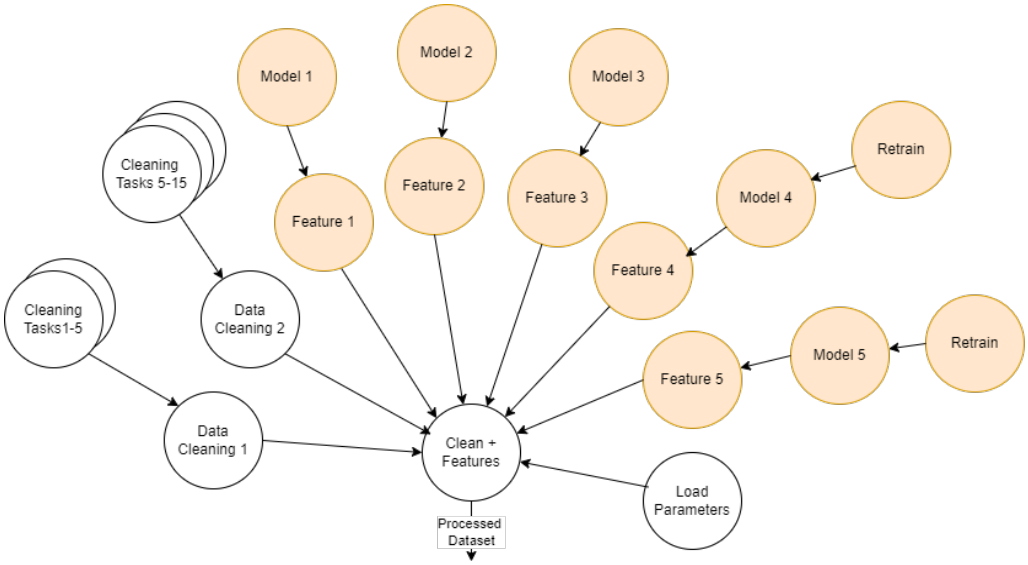

Figure 7: Subgraph of whole data science pipeline

BrandDelta calculates many indicators and measurements of customer satisfaction and brand recognition for their clients. These metrics are calculated with custom models and their specifics cannot be disclosed in this paper. Instead, for our application we will replicate one branch of

the pipeline with some standard models and see how this can be automated and improved with better MLOps practices. The application will be that of opinion mining from text reviews, with the objective of tagging reviews as positive or negative. This resembles some of the initial feature generation tasks of the pipeline at BrandDelta.

4.2 Machine Learning Problem

The objective of our application is to train a model to perform sentiment analysis on text reviews written by customers having tried a product or service. The sentiment scores will be used later as features for training and making predictions with other models, in a larger imaginary pipeline. We should think of our application pipeline as resembling the branch of Model 5 in Figure 7, meaning that it is a model we expect to have to retrain regularly.

As described by Westerski (2017), sentiment analysis is the computational process of extracting opinion from humanly written text, where the opinion is represented by a sentiment score. A simple example is trying to find if the author of a text had a positive attitude towards the subject or a negative one. Here, the sentiment would be a score with range of [-1, 1], where a score of 1 means that the text has a completely positive opinion about the subject, and a score of -1 means a completely negative one. For any score in between, the sentiment analyzer has found a mixed opinion. Sentiment analysis is a classic example of a supervised classification problem in machine learning, where training data is passed as text, specifically reviews in our case. Each training review has been labelled with a true sentiment score and the words in the review form the feature vector. A classifier is trained on this data to adjust parameters and minimize some loss function, once a well performing model is found it can be used to classify new reviews.

There are several algorithms that perform well at sentiment classification, Neethu er al. (2013) explore a machine learning approach to sentiment classification of social media data, some of their proposed classifiers include naïve bayes, support vector machines and ensemble models. Exploring different classification models and motivating their applicability on customer reviews data is outside the scope of this paper, we will instead say that we have knowledge that motivates a linear support vector classifier (SVC) for our use case. The SVC makes predictions using a two-parameter vector W of w_1 and w_2, and is trained by minimizing a cost function like equation 1.

$$L = \arg\min_{W} \sum_{i=1}^{n} \max(0, 1 - y_i(W^T \cdot x_i)) \tag{1}$$

This specific cost function is called Hinge Loss. As Lu and Jin (2017) demonstrates, Hinge Loss can be optimized using stochastic gradient descent. Gradient descent is an iterative optimization algorithm employed by many machine learning classification- and regression models. Before we see the addition of the stochastic part of the algorithm, let us see how normal gradient descent (GD) works and how it will be implemented in our pipeline.

4.3 Gradient Descent for Optimal Parameters

Let Figure 8 below represent the surface of our hinge loss function for $w_1 : -6, 6$ and $w_2 : -6, 6$. With the value of the loss function on the L axis, we can see that this non-convex surface has a global minimum and one or possibly two local minima. Gradient descent, as described by Ruder (2017), approaches the W-vector of a minima of the loss function by updating the parameters of W by the negative directional derivative of the loss function with respect to W, like equation 2.

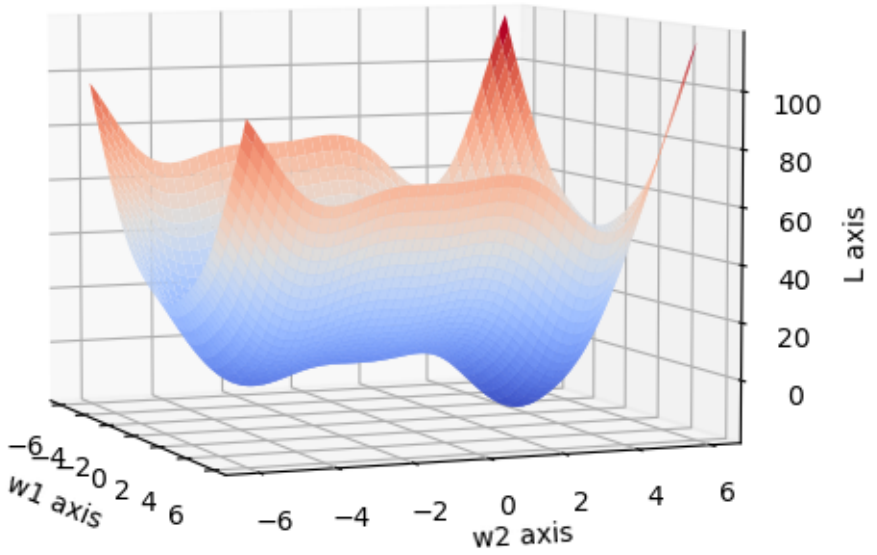

Figure 8: Surface of cost function

$$W_j = W_{j-1} - \alpha \nabla_{W j-1} L(W) \tag{2}$$

The updates of the parameters are made iteratively, each iteration *j* is referred to as an *epoch*. The α is called the learning-rate, a hyperparameter which sets the magnitude of the updates of the parameters in every iteration. The algorithm stops once a maximum number of epochs are reached or when the difference of W_j and W_{j-1} is sufficiently small. Looking again at Figure 8, the algorithm will follow the slopes of the surface until it hits a minimum, but this minimum is not guaranteed to be the global minimum. In Figure 9 below, is illustrated a heatmap of the cost function, here we can see more clearly that the global minimum exists somewhere around $W = [0, 3.5]$, this point is defined by the parameters that minimize the cost function, i.e. the parameters that produce the best SVC.

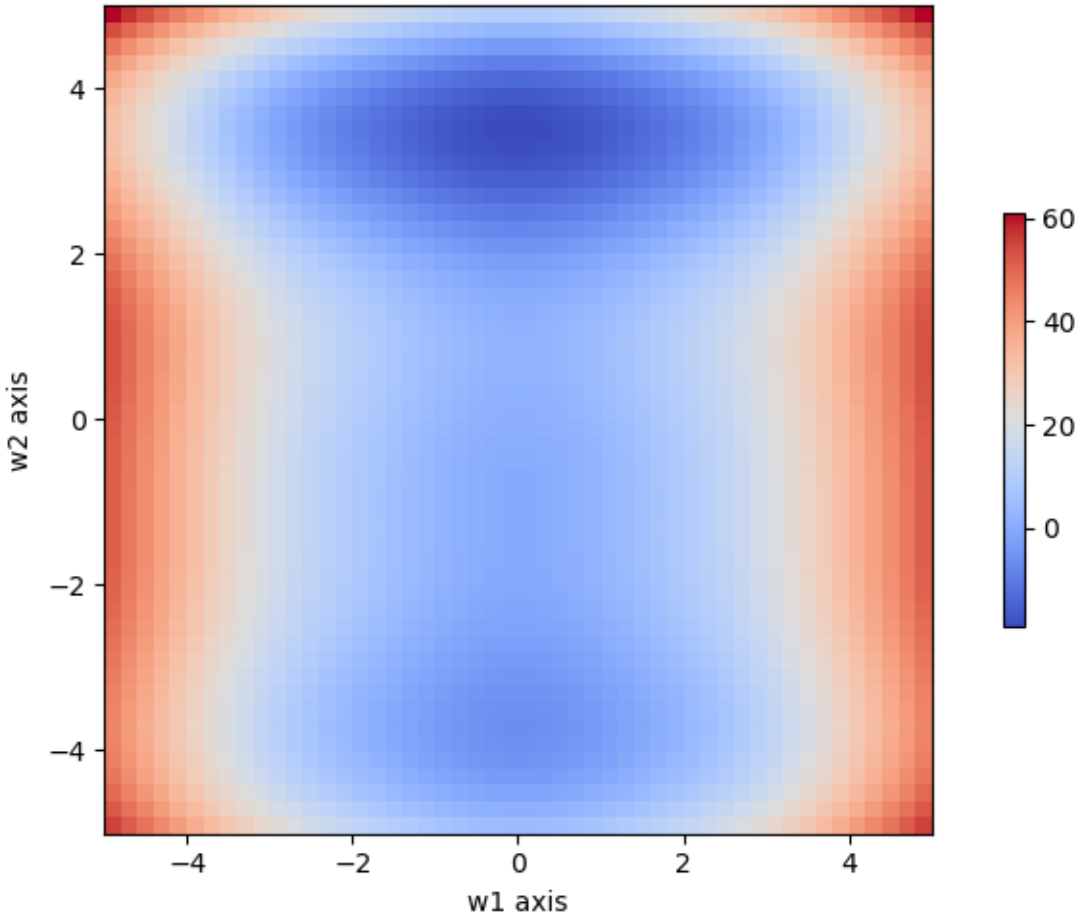

Figure 9: Heatmap of cost function

Stochastic gradient descent (SGD) is a simple but effective tweak on the original algorithm. In normal GD the whole training dataset is used in each epoch to update the parameters, stochastic gradient descent instead uses only one random sample or mini batch of the training data in each epoch. Stochastic gradient descent has been shown to converge to a minima much faster than normal GD and this greatly reduces the time needed to train the SVC (Ruder, 2017). Training a SVC by minimizing Hinge Loss with SGD is also a common practice utilized by standard ML-software like Scikit-learn (Scikit-learn, 2011). Just like in equation 2, SGD still needs the two hyperparameters number of *epochs* and the learning rate α.

There are two caveats we have to mention. The first is the issue of our gradient descent algorithm not finding the global minimum but instead getting stuck in one of the local minima. The second is how to choose the hyperparameters, learning rate and number of epochs, intelligently. There are several techniques for both of these issues, like using momentum-based gradient descent for not getting stuck in local minima, and using grid search for finding the best hyperparameters (Ruder, 2017). As this paper is more concerned with how to build efficient automated pipelines, we will use parallelized training of several models with different hyperparameters.

5 Results

To summarize our application, we want a SVC for classifying sentiments of text reviews, to find a good classifier we will train the SVC using SGD. The SGD may not find the global minimum and which minimum it will find will depend on what hyperparameters we choose. Utilizing parallel computing as described in chapter 2.3, we will evaluate several SGDs with different hyperparameters and choose the W-vector of best performing one. The pipeline, which will be orchestrated with Apache Airflow, should utilize the relevant MLOps practices to improve the robustness, speed, and ease of use for the entire model-life-cycle.

5.1 Application Pipeline

The DAG in Figure 10 represents the DAG built to replace the branch of Model 5 in Figure 7. The pipeline starts by ingesting the relevant data and splits the data into training- and tests sets, the next nodes are the parallelized SGD optimizations of the hinge loss cost function, which are run concurrently. Once all SGD nodes have run successfully, the final W-vectors-, hyperparameters- and cost function values found by each instance are forwarded onto the next node. In this evaluation node, the final model chosen is the one which found the smallest value of the hinge loss function. The final model is saved to a file-format that will be used in the production stage, where it can be called to quickly make new predictions on incoming data when we don't have to retrain the SVC.

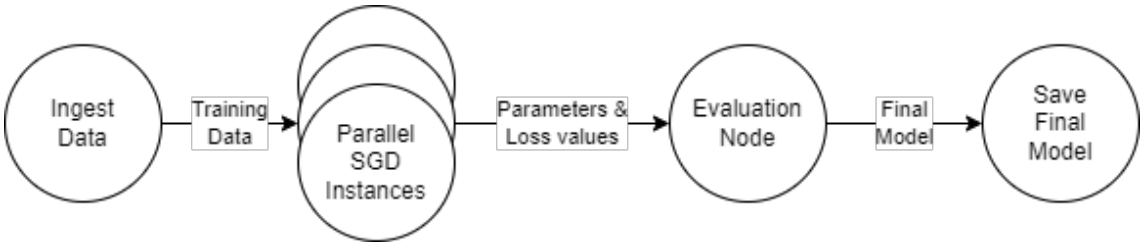

Figure 10: Application pipeline for producing SVC

The computing power needed to run the several SGD instances is provided by a cloud provider from which BrandDelta buys several other services as well. Computing units can be scaled on demand and the costs incurred by increasing the number of units will depend on the time each unit is utilized, as visualized in Figure 11. This way BrandDelta does not need to invest in expensive hardware, but can instead momentarily increase their cloud costs.

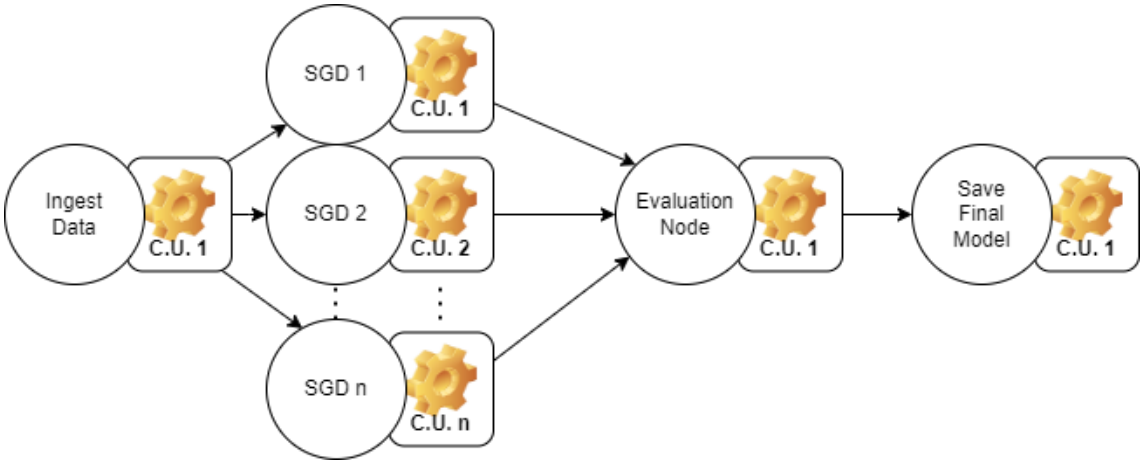

Figure 11: Scaling of C.U.s (compute units) across the pipeline

The operations performed in each node are built in individual python files. In a separate python file, we build the actual DAG, meaning we import all node-files and set the dependencies between them. When building the DAG we can also set a schedule for activating the pipeline. This means that we can automatically retrain the SVC when we ingest new data and automatically produce new outputs at regular time intervals. This is useful as we might know that new data flows into the company on given dates. Also, if clients have requested continuous reporting, we can automatically generate the results needed in the reports according to their requested schedule.

Using the Apache Airflow UI we can see runtime information about the pipeline and in the case of failures the UI can also show us at what node the pipeline failed and where in the operators python file the error occurred. In Figure 12 is the Airflow UI visualization of our deployed DAG. For this specific example the third SGD instance has been designed to fail, if we click on this node the Airflow UI will show us the log for this specific node and we can easily debug from the error messages logged. In this small DAG it is still easy to search and debug without the help of the UI. However, imagine once the pipeline grows into something more like Figure 7 or even more complex, then this interactive UI becomes very helpful.

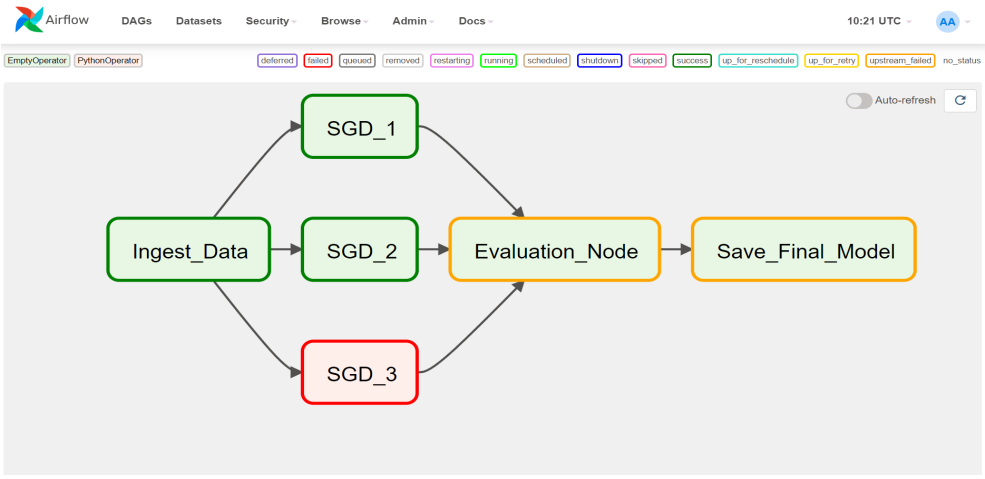

Figure 12: DAG as visualized by the Airflow UI

Since all nodes and the DAG itself are built in python files, we can easily collect these files in a folder. The specific Docker containers needed to run Airflow for our purposes can also easily be collected in another folder. We use a CI/CD tool to package and release our pipeline and relevant docker containers in a way that makes it both easy to update, when node- functionalities or required dependencies are changed, and allows different users to continuously have access to the latest version of the pipeline. This further improves automation and consistency on an enterprise-wide level since we can control which version users are running.

Finally, the SVC pipeline we have built in this chapter can now be added on to the larger pipeline from Figure 7. In Figure 13 below, the branch that used to be Model 5 has now been replaced by our SVC pipeline (the Ingest Data node has been removed as it is common for all branches and clutters the visualization). This section has focused on how one of model-pipelines should be changed, the same changes with relevant internal models will have to made in all orange branches of Figure 13.

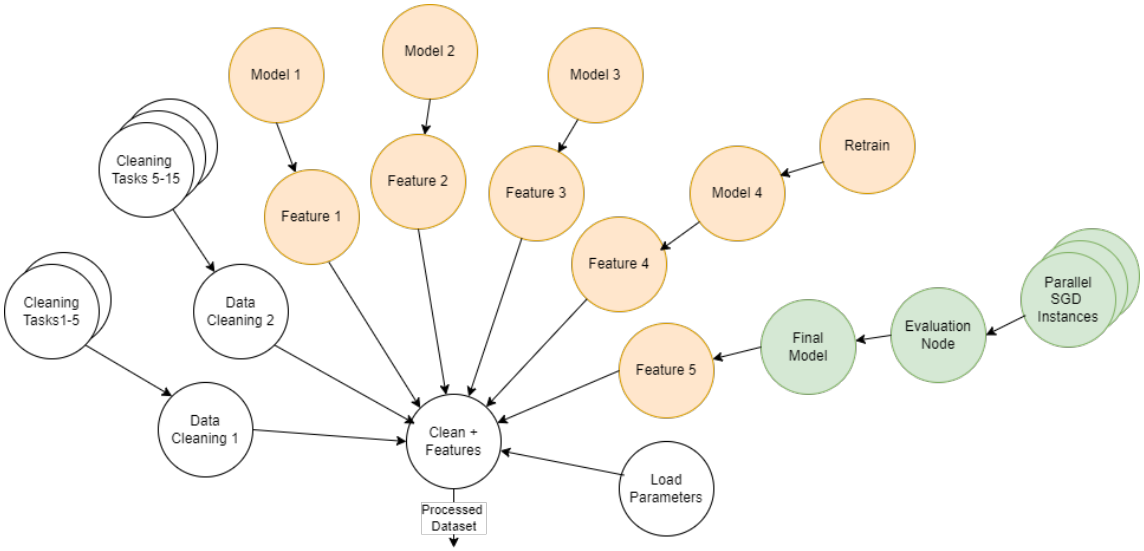

Figure 13: Adding our SVC pipeline to the larger pipeline

6 Conclusion

The purpose of this paper has been to explain how a small company like BrandDelta can implement better MLOps practices for maturing and stabilizing their machine learning applications. This processes of maturing and automating machine learning applications has also been my biggest project and responsibility during my internship at BrandDelta.

The MLOps framework presented in this report, by Antonini et al. (2022), is just one way of thinking of MLOps and is used as a structure to follow for thinking of how we can improve current applications. For the application presented in this paper, the focus has been to improve MLOps while at the same time automating as much of the ML-pipeline as possible, which also is one of BrandDeltas key concerns at this time. The steps we have focused on automating are firstly

the Creation- and Verification steps, where we propose the parallel generation of several models with different hyperparameters and automated evaluation. Secondly, we have also focused on the Package-, Release- and Configure steps, where we propose that the pipeline for our application be built with Apache Airflow which natively supports pipeline objects as DAGs. Airflow also provides environment-independent set up through docker containers as well as runtime monitoring through the Airflow UI. A CI/CD tool have also been connected to the pipeline to allow for automatic updates of the pipelines' operators whenever their functionalities are changed.

The structure of the final proposed pipeline allows BrandDelta to easily train and evaluate several new models as they can scale the compute power needed in the pipeline. Once this proposed structure has been implemented for all BrandDeltas ML-models, retraining of models will be as easy as activating the pipeline, since this retrains the model with the new incoming raw data. The next steps in BrandDeltas MLOps maturity should be to explore and implement automated testing for data drift and model degradation.

References

Airflow.
https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#running-airflow-in-docker

Antonini, Mattia & Miguel Pincheira & Massimo Vecchio & Fabio Antonelli. "Tiny-MLOps: a framework for orchestrating ML applications at the far edge of IoT systems. *2022 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS),* 2022.

Bekkerman, Ron & Mikhail Bilenko & John Langford. *Scaling up machine learning: Parallel and distributed approaches.* Cambridge University Press, 2011.

Boso, Nathanel & Magnus Hultman & Constantinos N. Leonidou & Oluwaseun E. Olabode. "Big data analytics capability and market performance: The roles of disruptive business models and competitive intensity." *Journal of Business Research*, 2022.

BrandDelta.
https://branddelta.com/

Garg, Satvik & Pradyumn Pundir & Geetanjal Rathee & P.K. Gupta & Somya Garg & Saransh Ahlawat. "On Continuous Integration/Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps". *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE),* 2021.

Kreuzberger, Dominik & Niklas Kühl & Sebastian Hirschl. "Machine Learning Operations (MLOps): Overview, Definition, and Architecture". arkXiv:2205.02302. 2022.

Lu, Shuxia, & Zhao Jin. "Improved Stochastic gradient descent algorithm for SVM." *International Journal of Recent Engineering Science (IJRES),* 2017.

Matskin, Minhail & Shirin Tahmasebi & Amirhossein Layegh & Amir H. Payberah & Aleena Thomas & Nikolay Nikolov & Dumitru Roman. "A survey of big data pipeline orchestration tools from the perspective of the datacloud project". *In Proc. 23rd Int. Conf. Data Analytics Management Data Intensive Domains*, 2021.

Neethu, M. S. & R. Rajasree, "Sentiment analysis in twitter using machine learning techniques". *Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT),* 2013.

R. Mitchell et al., "Exploration of Workflow Management Systems Emerging Features from Users Perspectives". *2019 IEEE International Conference on Big Data (Big Data),* 2019.

Ruder, Sebastian. “An overview of gradient descent optimization algorithms”. arkXiv:1609.04747 2017.

Rukat, Tammo & Dustin Lange & Sebastian Schelter & Felix Beissmann. "Towards automated ml model monitoring: Measure, improve and quantify data quality." *ML Ops workshop at MLSys,* 2019.

Scikit-learn. “Scikit-learn: Machine Learning in Python”. *Journal of Machine Learning Research,* 2011. `https://scikit-learn.org/stable/modules/sgd.html#sgd-mathematical-formulat`

Westerski, Adam. "Sentiment Analysis: Introduction and the State of the Art overview". *Universidad Politecnica de Madrid,* 2007.

wXin, Doris & Hui Miao & Aditya Parameswaran & Neoklis Polyzotis. “Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities”. *Proceedings of the 2021 International Conference on Management of Data,* 2021.