BSc Thesis at GeoPhy

Automating Valuations for Real-Estate

A. Geenen
H. Nguyen
R.T. Wiersma

July, 2017

GEOPHY
Real Estate meets Big Data

TU Delft

Automating Valuations for Real-Estate

BSc Thesis at GeoPhy

by

A. Geenen
H. Nguyen
R.T. Wiersma

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Project duration:	April 24, 2017 – July 7, 2017	
Thesis committee:	Dr. C.C.S. Liem MMus,	TU Delft, supervisor
	Ir. O.W. Visser,	TU Delft
	Ir. S.M. Mulders,	GeoPhy

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This document describes the software project as implemented for the Bachelor Thesis Project for Computer Science over the course of ten weeks. The project was commissioned by GeoPhy and aimed at integrating automated valuation models (AVM) into the GeoPhy architecture.

We would like to thank everyone at GeoPhy for their warm welcome towards our team. We have truly enjoyed working together with this bunch of creative innovators and have learned more than we could have imagined at the start of the project. We would also like to thank our coach, Cynthia Liem, for her enthusiasm and inspiration, for guiding us through this process, and helping us come to a solid project with a real use for GeoPhy.

A. Geenen
H. Nguyen
R.T. Wiersma
Delft, June 2017

Contents

Abstract

As GeoPhy is developing its business model and looking into the future of automated valuation models (AVM), this project delivers a proof of concept of a system that automates the training, maintaining, and delivery of machine learning models for automated valuations. In order to achieve this goal, the situation and problem were first analysed. This resulted in an outline of the desired product and requirements in the form of a MoSCoW analysis. An important goal for this project was to incorporate streams of data from a stream processing platform (Apache Kafka) into a service that would train and update models automatically. The second goal for this project was to keep track of the changes in the data in order to detect significant changes in distribution (concept drift) of the target prediction value.

These subjects were studied in literature, reviewing existing and upcoming valuation practices in real-estate, steps needed to perform machine learning tasks, architecture to support big data processing, and concept drift. This resulted in a design made up of four different components: An ETL and data processing component, a modelling component, a Kafka connector, and a client-facing API. An important part to ensure efficiency and scalability of the system is the implementation of concept drift: models are only retrained when the distribution of the target training value has changed significantly.

These components use storage in the form of a Postgres database, disk storage and Elastic Search logs. The logs (on model performance and concept drift usage) can be interpreted through a Grafana dashboard, which is editable through its own GUI.

Finally, to test the success of the project, a testing plan was set up and the code was reviewed by an external group (SIG). The code achieved all the testing milestones and received a 4.5/5 in a mid-development review on maintainability. With this project, the concept of automated valuation models inside GeoPhy's new architecture has been tested and proved and the project is ready to be further developed and used in practice.

List of Figures

List of Tables

1 Introduction

Data is the new oil![1] More and more data is gathered and GeoPhy is one of the 'refineries' of this oil; gathering, enriching, and analysing data for the real-estate market. In an effort to streamline this process, GeoPhy has been working on a new internal architecture that makes use of stream processing to enable the flow of data from one place to the other inside the company.

Together with this renewal in architecture, GeoPhy is exploring the application of Automated Valuation Models as a business opportunity. Current and historical valuation processes are time consuming and use a lot of manpower. Automating these valuations with the help of machine learning models provides an advantage in terms of costs, but also in accuracy.

This project aims to bring these two developments together in a system that takes care of the automation of valuation models and the flow of data to these models. While doing so, it also makes sure that the valuation models are up to date with the latest data and provides insight into the performance of the system with logs and a dashboard. The project is unique, in that it requires a system that is both efficient and scalable, as well as usable and maintainable, while making use of distributed systems and the latest developments in software engineering.

In this report, the project is described and motivated from the initial problem description to the design and implementation of the system. In chapter two, the problem is defined and explained. This problem is further analysed in chapter three, which culminates in a set of requirements in the form of a MoSCoW analysis. In chapter four, the background of the project and possible solutions are explored in existing literature. These explorations have led to the design and implementation of the final system, which is described in chapter five and six. In chapter seven, the results of the project are weighed and tested against the problem and requirements that were set in chapter two and three. This leads to a conclusion in chapter eight, after which the final system is discussed and critically examined in chapter nine. Recommendations based on this discussion are given in chapter ten, together with general recommendations to further develop the system.

[1] The first recorded mention of this was by Clive Humby in 2006, at ANA Senior marketer's summit at Kellogg School

2

Problem definition

This project aims to solve the following challenge: GeoPhy needs a system that can run automated valuation models (AVM) in production (AVM Service), which works in GeoPhy's proposed architecture, consuming stream data, and trains its models based on changes in the market. Simply retraining the models every time a change comes in is inefficient and ineffective. The models take a lot of processing power to train and are designed to generalize well; they will not change much as single values are updated. The system should find a way to determine and detect a significant amount of changes to the data to validate a retraining.

In further detail: the AVM Service should consume a data stream with large volumes of data to perform the following:

- Build a persistent view of the current state of incoming data that can be used by machine learning models.
- Analyse the data stream in order to detect changes to the distribution of the target prediction value (e.g.: the value of houses increase due to an improving economy).

Based on this data, the system should:

- Train an automated valuation model based on the persistent view of incoming data.
- Retrain an automated valuation model only when a significant change in the distribution of the target value has been detected.
- Provide predictions to a user through an API.

AVMs provide predictions for property values based on the data that is readily available (e.g. number of rooms, value) and data that GeoPhy 'deduces' from other sources (e.g. transport options in the neighbourhood). The models are trained using machine learning techniques, such as decision trees and random forests.

The amount of data used to train these models can be enormous: each sub-market has millions of buildings and data is gathered on the property itself, but also the surrounding areas. The size of these datasets can grow into tens of gigabytes and the models require many operations on this data in order to be trained.

As GeoPhy is developing its business model, it is also reworking its architecture in order to support a growth of new services based on the pool of data they are currently building. One of the key components in this architecture is the use of Apache Kafka to communicate between the main database (CoreDB) and GeoPhy's services, AVM being one of them. Kafka communicates the changes of the data (producer) to all of the services (consumers) through a stream of updates. In the case of GeoPhy, this data consists of information on properties. The changes are updates to the real-world properties, like a change in value, extra buildings, or new infrastructure.

GeoPhy should be able to monitor the detection system for changes in distribution of the input data and modify the parameters to optimize its effectiveness. This, in order to make sure that the models are not retrained too sparsely.

3

Problem analysis

The problem definition posed in the previous section will be explored in more detail in this problem analysis. This results in a MoSCoW analysis, through which the success of the project can be measured.

3.1. Models in production

One of the aims for the system is to run the models in a production environment. What does this actually mean and how does this translate to requirements for the software? The production environment for GeoPhy is a server or cluster of servers that has to serve predictions to large commercial parties who expect high quality, dependable software. As mentioned before, GeoPhy uses large volumes of data on millions of buildings. In order to measure the quality that is expected, the International Organization for Standardization provides the following quality measures for systems and software engineering, which will find their way into the list of requirements at the end of this chapter: functionality, performance efficiency, compatibility, usability, reliability, security, maintainability, portability[15]. *Functional:* it should do what it is advertised to do; return accurate predictions through machine learning models. *Efficient:* with the amount of data considered, it should train the models efficiently, so that processing time (a valuable resource) is used well, and it should return predictions quickly, so that end users have a smooth experience. *Compatible:* it should interact with all of the current or planned systems well through the use of Kafka streams. *Reliable:* the service should respond consistently, should be tested thoroughly, and if a model is badly trained, GeoPhy should be able to return to previous versions of the model. *Secure:* internal data should not be exposed to external users and external users should not be able to breach into the system. *Maintainable:* GeoPhy engineers should be able to maintain and modify the code easily and, on change, the system should not break or be notified when it does (testing). *Portability:* the system can be deployed on different machines regardless of their operating system or number of machines.

Regarding usability, the measure can be split up into two cases: usability for GeoPhy data analysts and usability for external users.

3.1.1. Usability: models in GeoPhy

When a customer requests a specific type of valuation model from GeoPhy that has not been implemented yet, a data analyst will explore the task with tools such as DataIku[1] or Python notebooks[2]. In this software, the analyst can iterate easily and explore different kinds of settings and configurations of features to use when training the model. Once this exploratory process is done, the result is a list of data sources, features that provide the best predictions, and settings for the machine learning model that is used (e.g.: number of iterations or tree depth for gradient boosted trees). The data analyst could provide the predictions for desired properties from these exploratory cases, but they would like to integrate the model into the GeoPhy architecture, so it can make use of up-to-date data and automated prediction calculation through an API. In order to do this, the data analyst should be able to transfer their model to the AVM system.

To make this possible, the AVM system must accommodate the most common machine learning techniques that are used by the analysts (XGBoost, Gradient Boosted Regression, Decision Trees). An analyst should not need to dive into the code to add functionality: they should preferably have one place to define necessary datasources and one place to define the model without having to add any other code. The system should take care of any actions that are reused in different models. The modelling framework needs to be documented well and there should be an example of an implemented model, so data analysts can learn and copy from previously created models.

Finally, GeoPhy needs to monitor the performance of the model without having to run any code. As the model is updated, one needs to be able to see if the model's performance actually improved or deteriorated and how it develops over time. This should be done through a dashboard where performance metrics are logged.

3.1.2. Usability: models for users

A user should be able to retrieve predictions fairly simply. In order to accommodate this, the AVM system should provide an API, so the valuation can be incorporated in another GeoPhy product with a front-end component. As the API is first processed by this other component, a user will not directly interact with this API. Still, the API's calls and responses must be understandable to a software engineer who is new to the system. Again, this requires creating a clean and clear design, but also documenting the functionality well.

3.2. Changing markets

As markets change, the distributions of training data change, and the models need to reflect these changes. When, for example, a market plummets, and property values change accordingly, the model should be retrained so the lower prices will be reflected in the predictions. If, however, a change comes through that does not imply a shift in the market, the model does not need to be retrained. Superfluously retraining would only take valuable processing time and does not reflect a stable state of the entire property market.

This means the system should detect a significant concept drift (concept refers to the distribution of the target value), but should also be resilient to meaningless noise in incoming data.

Finally, the implementation of such a detection mechanism implies the use of a model. As models are subject to faults, overfitting or underfitting, this model should be monitored too. GeoPhy should be able to monitor the choices made by the detection algorithm and should be able to return to a previous model if the choice to retrain was made prematurely. Also, the parameters used for this detection algorithm should be tuned to optimize its effectiveness.

3.3. MoSCoW analysis

In order to create a prioritization of the steps necessary to deliver the desired software project, a MoSCoW analysis was done on the points mentioned above. This resulted in the follow

[1] http://dataiku.com
[2] https://jupyter.org

requirements (the requirements are numbered for further reference, the ordering should be ignored): the software …

3.3.1. Must

1. Train a model based on the configuration set by a data analyst.
2. Have a RESTful API that can provide valuations.
3. Calculate predictions on an API call, using a trained model.
4. Version models and save old models for later use.
5. Consume events/values from a Kafka Stream.
6. Process multiple forms of data (data *type* agnostic).
7. Accept data points for models through a uniform interface (data *source* agnostic).
8. Have test coverage above 80%.
9. Score at least 4/5 on the SIG maintainability check.

3.3.2. Should

1. Detect a significant change in distribution of the target value (concept drift) from the incoming data.
2. Retrain a model when concept drift has been detected for the target value of that model.
3. Provide insight into the performance of the concept drift detection.
4. Provide insight into the performance of the model.
5. Consume events from multiple Kafka Streams.
6. Be able to source data from external APIs and databases.
7. Support models that require different kinds of data (numbers, vectors etc.).
8. Have at least one example valuation model that values multi family homes in parts of the US.
9. Provide centralized places to set up the model and data sources.

3.3.3. Could

1. Support different modes of valuation calculations (batch calculations vs. on demand).
2. Have a valuation model that values multi family homes for the entire US with less than 10% MdAPE.
3. Provide tools to perform time series modelling.
4. Automate the feedback process for concept drift detection.
5. Provide exactly one place to set up the model.

3.3.4. Won't

1. Have a user interface for building and setting up models.
2. Use data outside of readily available sources (i.e. no labour intensive data collection).
3. *Predict* sudden changes for the distribution of the target value.

4

Literature Review

After the problem was analysed, an in-depth study of literature concerning the subject material was undertaken. This study attempted to answer the following main question: what is the context and what are best practices to design a system to automate the data flow, training and retraining, and delivery for AVMs?

First, an understanding of the background of valuations and modelling is necessary to understand the context in which the system must perform and the kind of functionality that is to be expected from the system. Current valuation practices are summarized, after which the step is made to more recent and future models in the form of machine learning and neural networks. The process of creating and maintaining such a model in a software system is then assessed, after which the design principles to implement the system are studied. Finally, topics that are specific to the desired system are studied: Apache Kafka and Concept Drift.

These topics resulted in the following sub-questions, answered in subsequent sections:

- How are property values estimated in current real-estate practices?
- What machine learning models are used to estimate real-estate value?
- What are the steps that a software system should take to create a machine learning model?
- What are best practices for designing an architecture for varying prediction models that can be accessed by an API?
- How should a Kafka stream be processed by an application?
- How can concept drift be detected by an application?

4.1. Current valuation practices

According to the International Valuation Standards Committee, “Market value is the estimated amount for which an asset should exchange on the date of valuation between a willing buyer and a willing seller in an arm’s length transaction after proper marketing wherein the parties had each acted knowledgeably, prudently and without compulsion.”[7] Current valuation methods are based on the idea that the model to calculate these valuations should be based on the thought process of a ‘willing buyer’ or a ‘willing seller’ and what these parties are willing to pay or receive for the property being valued. The current valuation methods are grouped by Pagourtzi et al. as follows:

1. Traditional valuation methods:
    - comparable method;
    - investment/income method;

- profit method;
- development/residual method;
- contractor's method/cost method;
- multiple regression method; and
- stepwise regression method.

2. Advanced valuation methods:

- artificial neural networks (ANNs);
- hedonic pricing method;
- spatial analysis methods;
- fuzzy logic; and
- autoregressive integrated moving average (ARIMA).[30]

The thought process of the buyer and seller can most explicitly be traced in traditional valuation methods. These methods try to estimate the value of a house, either on the value of houses that are comparable to the property (comparable method), on the investment and returning profit from letting or using the property (investment/income, profit method), on the cost of developing the property (development/residual), or the cost of replacing the buildings on property if one were to undertake this task again (contractor's/cost method). In the remaining traditional valuation methods, multiple regression (MR) and stepwise regression (SR), one can trace the buyer's thought process in the following way: regression methods try to find the predictive relationship of different attributes to the price of a property. If, for example, a good view is important to buyers (as witnessed in a study by Benson et al.[4]), regression analysis is supposed to find determine the impact of this attribute on the value and create a linear function based on these attributes.

Advanced valuation methods have developed in the last two decades and incorporate new technologies in the valuation mix. Artificial Neural Networks find patterns in a big dataset without having predefined restrictions (aside from the boundaries of the network itself) or assumptions[10]. This gives it the ability to find previously unknown connections between attributes and the valuation. The networked nature of the model makes it extremely hard to understand the 'reasoning' within the model. As developers of TensorFlow described the process of developing their system for Deep Learning (using Neural Networks): "Some of the stuff was not done in full consciousness. They didn't know themselves why they worked."[22] This makes it hard to pinpoint how exactly the model mirrors the parties' thought processes, but the accuracy of valuations does prove its effectiveness[26].

The hedonic pricing method is a version of linear regression that tries to make inferences about non-observable values of different attributes (e.g. air quality). Spatial analysis incorporates geographical data and objects that are close to the evaluated property to determine its value. According to McCluskey et al., this method appears to be the most transparent for assisting mass appraisal in terms of "cost-effectiveness, user applicability and predictive accuracy."[26] Fuzzy logic uses logic based rules with fuzzy determinants to group buildings and has the advantage of taking into account the hierarchy of the market and allowing linguistic variables for evaluation. Finally, ARIMA is a class of regression modelling that includes time series in its method through the moving average.

4.1.1. Evaluating models

In order to evaluate these practices and determine the need for a system based on machine learning principles (aside from ANNs), it is useful to have an assessment rubric for the evaluation methods. As mentioned in the previous paragraph, it is important for a valuation method to mirror the thought process of the potential buyer and seller, as these will eventually make the transaction for the property that is valued. McCluskey et al. provide the following criteria for comparing different modelling approaches:

- predictive accuracy of the estimates;

- conceptual integrity;
- analysis of valuation variation where more than one mass appraisal methodology is applied;
- internal consistency of the model;
- nature of any model adjustments (i.e. the adjustment of the model predominantly spatial/structural or temporal in nature);
- reliability and robustness of the model;
- feasibility in terms of cost and time efficiency; and
- explainability of the model in terms of being able to defend the estimates in a formal setting, such as an appeal tribunal or court.[26]

The predictive accuracy and consistency of the results (conceptual integrity, internal consistency) are usually present in the current models. What can be concluded from McCluskey's comparison of the different methods is that ANNs tend to excel in the area of predictive accuracy and cost and time efficiency (once the model has been trained, calculating estimations is relatively cheap), but lack in consistency (some training sets give the model a better outcome than others) and explainability. Because of these last reasons, McCluskey ends up favouring Geographically Weighted Regression (GWR).

Researchers have also found fault with linear regression models[2, 37]. They argue that some attributes are not linear in nature and that they have a problem with aggregation bias. Through the use of hierarchical models (models that account for hierarchy: houses within streets within neighbourhoods)[2] or models that accommodate the existence of submarkets[37], they try to mitigate these effects.

Another issue that came up in research was the updating of the model to continuously serve accurate valuations. Mak et al. attempted to implement a system that would give mass appraisals for homes based on a hedonic model. They concluded that online access to these appraisals could save consumers a lot of time, but they also noted that "major technical problems of data modelling and accuracy arise from the efforts to merge various data sources into a single integrated product. Keeping the hedonic model results up-to-date after the initial development efforts is also a time-consuming and costly task that might have been underestimated at the outset."[24] This highlights the issue of efficiency of the algorithm and the need to assess when a model should be updated. Mak et al. suggest that a time interval to update the model should be determined by professionals. In this project, it will be attempted to design an automated process to make this determination using concept drift.

It can be concluded from the existing literature that there is room for improvement in the areas of accuracy and efficiency. Neural Networks provide improvement and also account for previously unseen connections in the data. The networked nature of these techniques offer a barrier for explainability, though. Explainability is an important aspect of valuing homes; buyers, businesses and home-owners want to know why a certain house has a certain price and its still hard to provide this explanation with current deep learning techniques.

4.2. Machine learning models

Other machine learning methods might provide an outcome: they can 'find' connections and patterns within data (like non-linear relations) that might not be found with traditional methods and predict valuations with high accuracy. Next to this, they might offer more transparency in the way the model values properties compared to Neural Network approaches. In this section, this approach will be further explored and existing research examined for applicability.

It is out of the scope of this review to compile a comprehensive overview of all research on machine learning. Many different approaches have been tested and each paper finds its approach to be the most effective or provides some insights on how they used their methods.

The research on mass appraisals for real-estate using machine learning methods suffers from a limited availability of data and lack of open source code[23]. Data has become a highly valuable resource for businesses, which makes it hard for researchers to get data within their research budget. The outcome is that datasets for these experiments are relatively small and that resulting techniques are quite specific to those datasets[41]. Nonetheless, it is still useful to survey some approaches and see what these researchers found in their experiments.

The papers that were studied mostly experimented with their models in WEKA (an open source machine learning framework in Java[29]) and MatLab[1] and compared the different methods based on their accuracy measured in MdAPE (median absolute percentage error), RMSE (root mean squared error) and other accuracy measures. Several papers resonate the previously found issue with Neural Networks in their literature reviews: they note that results obtained through neural network techniques are often unstable[1, 16]. Rafiei and Adeli used Deep-belief Restricted Boltzmann Machines to estimate the sale price of a new building before the start of construction[32]. They found that their approach was effective and focused mostly on reducing the dimensionality of the problem; i.e.: selecting the right features for prediction. Crosby et al. proposed a four-step process where data was analysed, extracted and a prediction was derived using Gaussian Process Regression[8].

Antipov and Pokryshevskaya set out to justify Random Forests for mass appraisals and compared the method with nine other methods[1]. They believe Random Forests could become one of the best techniques for mass appraisal because, among other reasons, they achieve good results in comparative studies, adequately work with missing data and are robust to outliers. They expect Random Forests to avoid making mistakes that other techniques run into. Antipov and Pokryshevskaya concluded that their research "validated the application of the Random forest method to the mass appraisal."

Park and Kwon Bae studied four algorithms using the Weka software: C4.5, RIPPER, Naïve Bayesian, and Adaboost[31]. They studied data from Fairfax County in Virginia and compared their techniques using the same data for each method[23]. Their models were trained to predict whether the closing price was higher or lower than the listing price, which is a boolean answer compared to the current issue of continuous valuation. Nonetheless, the researchers found that "a machine learning algorithm can enhance the predictability of housing prices and significantly contribute to the correct evaluation of real estate price." Research by Kontrimas and Verikas echoes this general sentiment. They explored the usefulness of prominent computational intelligence techniques for mass appraisals and concluded that their SVM outperformed traditional regression and MLP based models, implying the need for non-linear modelling in mass appraisals.

The following general tips were derived from the papers. Sub-markets and location are important to account for in any model[19, 23, 41]. Antipov and Pokryshevskaya found that "feature importance diagnostics has revealed that the district, time to the city centre by underground, house type, total area of the apartment and bathroom unit type comprise the two most important groups of price per square meter predictors. The factors of low importance include indicators of inequality between room areas, the floor, on which the apartment is situated, and telephone availability."[1]

4.2.1. Feature selection

Some of the studied literature paid special attention to the selection of features[32]. The approach used by Rafiei and Adeli used a Non-mating Genetic Algorithm to come up with a selection of features that would best predict the market value of a house. Lowrance used an L2 regularizer to find the most important features[23].

4.2.2. Ensembles

In their comparison of machine learning techniques, Kontrimas and Verikas found that a committee (ensemble) of models outperformed the separate predictors[16]. The combining of different techniques has been studied extensively by Lasota, Graczyk and others[13, 19–21].

[1] https://mathworks.com/products/matlab.html

The different options are (1) bagging (bootstrap aggregation), (2) boosting and (3) stacking. (1) When applying bagging, one trains the different models on different parts of the dataset. (2) In boosting, models are trained sequentially on the part of the dataset that was not correctly predicted by the previous model. (3) In stacking, models are trained on existing data, and the outcome of these models is comprised into a new dataset. This new dataset is combined with the original dataset to create a new model.

Graczyk et al. compared these techniques and found that *stacking* provided the lowest prediction error, but tended to be inconsistent in its quality[13]. Bagging was most stable, but performed the least in terms of accuracy. Graczyk et al. conclude that the results show there is no single algorithm that produces the best ensembles and an optimal hybrid should be sought for each dataset.

4.2.3. Concluding

Machine learning techniques (like random forests, SVM, and Adaboost) are effective and provide improvements over traditional techniques and neural networks in terms of accuracy and consistency, but there is no *one* best approach. Finding the optimal model tends to be more of an art in trying out different techniques and optimizing an ensemble for the dataset that is used. From the literature, a couple of techniques have been shown to work and provide better results: random forests, using ensembles. Zurada showed that, overall, AI-based methods tend to perform better for heterogeneous data sets containing properties with mixed features[41]. Based on this review, this project will support ensemble machine learning models and test out the different techniques available in the machine learning toolbox. The key is to support extensibility and a wide range of machine learning models, as new and better methods continue to be developed.

4.3. Modelling steps

In order to support a wide range of machine learning methods, it is useful to have a general idea of the steps needed to create a model through machine learning. Much of this knowledge was acquired from Negnevitsky's work on Artificial Intelligence: A Guide to Intelligent Systems[28]. The first stage in creating a machine learning model, is getting the data. According to Bellotti, important data sources for AVM include real estate property listing information, transaction data from land title registers, data from syndicates of local solicitors or mortgage data from banks[3]. Getting this data might be difficult, as most of it is proprietary.

The second stage is data preparation. This should be done as automated as possible. Performance of the created model largely depends on the quality of the input data. Raw data is often not suitable for learning, but one can construct features from it that are suitable[9]. This process, known as feature engineering, as well as feature selection, can be difficult because it requires domain-knowledge of the problem.

In addition, the data needs to be integrated and cleaned. Cleaning the data improves the quality by removing outliers and impute or remove missing values. Joining the data sets from different sources together often present a challenge, as different sources will use different styles of record keeping, conventions, time periods or keys[39].

After features are selected, extracted, and cleaned. The data is ready for the model to be trained. The dataset is then divided in a training set, on which the model is trained, and a validation set, on which the model is validated after being trained. When the model has reached satisfactory levels of accuracy, it will be put into 'production' and items that need to be classified or appraised are fed into the model and an output is returned.

4.4. Architecture best practices

When designing a system that has to be able to accommodate a variety of models there are a few important points to take into account. Chief among them are the ability to make it possible to easily add new data sources for use by models, and to let the system be "model-agnostic", meaning that models that conform to a minimal interface should be able to be used interchangeably.

4.4.1. Extensibility of prediction model framework

Various architectures that have extensibility as one of the main design requirements utilize a component-based design[25, 33, 34]. Component-based design is centered around the separation of concerns within a software system, meaning that each piece or component addresses a specific functionality. When building components, the goal is to abstract away the interface of a component from the implementation details. This increases flexibility, as programmers are then able to modify underlying logic inside of a component without affecting the functionality of a system outside of the component's boundaries[33].

Some ways in which component-based design can be applied to modelling systems are the splitting of models into their own components[34], or placing models and their data sources together into individual components[25]. One architecture proposed by researchers was the use of so-called "Mini-Apps"[35]. This consists of domain-specific models and algorithms implemented in various programming paradigms (e.g. MapReduce), such that they are agnostic to the architecture surrounding them, and are able to be composed into more powerful workflows and scale easily. An alternative to this is to co-locate the model and data sources. Researchers have proposed a framework called Compositional Inference and Machine Learning Environment (CIMLE) in which this is utilized[25]. This system allows the creation of multiple components that are composed of smaller "primitive" components that include a model of choice, a data manager that takes care of the data loading and cleaning, and interfaces for communicating with the host environment. The framework surrounding these components allows users to configure the data flow and analysis pipeline without necessarily understanding the internals of the individual analytics components. Systems designed using these kinds of component-based designs are ultimately very extensible as they are able to swap in models and data sources without severely impacting the overall system.

4.4.2. Loading data

One of the most cost and resource intensive steps in the design and implementation of a modelling system is the loading of data in such a way that it is readily available, easy to access, and of high enough quality to actively contribute to the performance of a model. The extraction transformation loading (ETL) process, commonly used in industry, provides one such way to pull data from a variety of heterogeneous data sources, combine them so that they are useful, and load them into a system (usually a data warehouse) for further use[36]. As this is such a critical part of a data-driven system, there are various quality and design metrics, grouped by Theodorou et. al, that have been defined for these processes:

- Data quality - How good is the data? (measured in accuracy, completeness, freshness, consistency, and interpretability)
- Performance - The speed of the implementation of the ETL, resource utilization, capacity, and types of ETL modes
- Upstream overhead - The load that is placed on data sources by the ETL process
- Security - The protection of sensitive information during the ETL process, the prevention of unauthorized modifications, and the reliability of the system (e.g. recoverability, availability, and fault tolerance)
- Auditability - The means to explain the ETL process i.e. How the data is sourced? What business rules are applied during processing?
- Adaptability - How resilient to change is the ETL process?
- Usability - How difficult is the ETL to use and/or configure?
- Manageability - The maintainability and testability of the ETL process

Taking these metrics into account when designing the ETL process for an extensible framework, metrics such as adaptability and usability become very important. However, as concluded by Theodorou, these metrics are at odds with one another and so a careful balance has to be struck between the various characteristics.

The ETL process in its current form is relatively static, and often requires the data warehouse, the place where the processed data is stored, to be partially unavailable when new data is loaded in. Ideally this would not be the case, and researchers have proposed a new streaming version of the pipeline in order to allow the data warehouse to be fully accessible at all times[12]. This system, called Stream Data Warehouse (StrDW), allows the on the fly processing of new data as it arrives, and facilitates the use of a variety of different data sources including data streams, external databases, and static files. It uses a load balancer to accomplish this, that can source data from either a database that contains all historical data and an aggregate stream of the most recent data. Once the data can no longer be found in the external streams it is added to the historical data in the database.

4.5. Stream data and Kafka

Kafka is a distributed and scalable message broker, created to process large volumes of data. The basic concepts of Kafka are as follows[17]:

- A stream of messages of a common type is defined by a topic.
- Producers can publish messages to a certain topic.
- Published messages are stored on a set of servers called brokers.
- Consumers can receive published messages by subscribing to one or more topics.

In Kafka, different from most messaging systems, the brokers are stateless and do not maintain information about which messages are consumed by subscribers[17]. Instead, the brokers store messages on the server for a certain time period, after which they are deleted. Therefore, applications that consume from a Kafka stream must maintain themselves which messages are consumed, and ensure messages are finished processing before their retention period. A benefit of this design, is that consumers can replay messages to an old offset.

Applications based on stream processing, often operate on a small window of recent data. The computations are generally independent, and results are near real-time. However, most use-cases found in the literature, such as analysis applications, often combine batch processing and stream processing to provide low latency results. The lambda architecture describes such a design[18]. This architectural pattern consists of two layers: a batch layer for accurate results, and a speed layer for low latency results. By joining the output of both layers, the results always reflect the latest data.

Currently, there are two large distributed stream processing frameworks: Apache Spark and Apache Flink. Apache Spark streaming is based on mini-batch processing, and thus also allows for easy implementation of the lambda architecture. Apache Flink provides a universal dataflow engine designed to for both batch and stream processing in single unified architecture[6].

4.6. Concept drift

A resilient valuation modelling system should be able to react to changes in the underlying dataset, as to minimize the loss of accuracy of the models. These changes in the representativeness of the historical training data to reality are known as concept drift[40]. In mathematical terms, concept drift is a change over time in the joint distribution between input variables X and a target variable y. This concept drift can be split into two categories: real and virtual drift. Real drift occurs when there is a real shift in $p(y|X)$, which does not necessarily require a shift in the input data distribution $p(X)$. Virtual shift is a change in the distribution of the input data $p(X)$ which does not alter $p(y|X)$[38].

Zliobaite created a taxonomy categorizing the different techniques that can be used to handle concept drift in a supervised learning environment. The two main categories are evolving learners and learners with triggers[40]. Evolving learners are systems that handle new input data as it arrives. Examples of such systems are adaptive ensembles, that train many classifiers and then choose which ones to weigh more heavily in the final predictions, and models that tweak their own parameters or design as new data is processed. Learners

with triggers are systems that determine how or when the models in use should be altered, based on heuristic methods. Examples of triggers are change detectors and training windows. Change detectors monitor changes in inputs and outputs in order to detect whether or not concept drift has occurred, while training windows focus on the selection window from which to source the training data.

4.6.1. Change detection

For systems that use models that are not adaptive, triggers must be used. Determining when a model should be retrained would be a use case for change detection methods. One example of a change detection algorithm is the Cumulative Sum CUSUM test[11], assuming a zero-centred process. The test is calculated using $g_t = max(0, g_{t-1} + (x_t - \delta))$ where $g_0 = 0$. If $g_t > \lambda$ then concept drift is alarmed and g_t is set to 0. A variant of the CUSUM test is the Page-Hinckley test (PH), which tracks changes in the normal behaviour of the values being observed. PH is calculated using $m_T = \sum_{t=1}^{T}(x_t - \bar{x}_T - \delta)$ where $\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$ is the average of all observed values and δ is the size of changes that are deemed to be acceptable. $M_T = min(m_t | t = 1 \ldots T)$ is the minimum of these values. Just like in CUSUM, λ is set as the threshold, with concept drift being alarmed once $m_T - M_T > \lambda$.

Other types of change detection include detection windows. Adaptive Sliding Window (ADWIN) uses a fixed sliding window which contains the most recent inputs[11]. Within this sliding window, a comparison is made between sub-windows and when two sub-windows are different enough, concept drift is alarmed. The test is defined as when the mean of the two windows is larger than the Hoeffding bound. This bound is defined as:

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} ln \frac{4|W|}{\delta}}$$

where $|W|$ is the length of a window, δ is the confidence parameter, and

$$m = \frac{2}{\frac{1}{|W_0|} + \frac{1}{|W_1|}}$$

is the harmonic mean of the two sub-windows.

4.7. Conclusion

Current valuation practices were studied and summarized. From this summary, it was concluded that there is room for improvement in the areas of accuracy, by taking into account locality and the non-linearity of the valuation process, and explainability. Then machine learning methods were compared and evaluated for their applicability in mass appraisals. It was found that multiple researchers have been successful in applying machine learning methods and have achieved higher accuracy, consistency and explainability through the use of machine learning methods like decision trees, random forests, SVM, and ensembles of different techniques. The process of creating a machine learning model was summarized to finish up the research of valuation practices.

Then, practices for designing an architecture for component and interacting with stream data, and the Kafka streaming platform were studied. Some important principles for the design of the current application are the interchangeability of different components and allowing for the application to react to changing data on the fly. Finally, concept drift was researched in order to come to a method to decide on retraining and updating models based on changing data.

This research and conversations with GeoPhy have led into an initial design that has evolved during development, which will be detailed in the next chapters.

5

Software Design

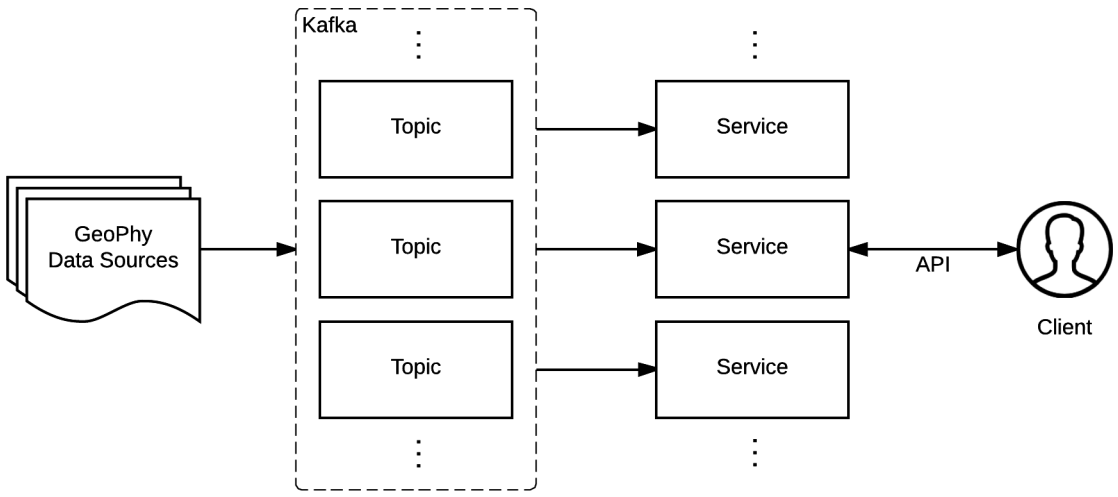
The knowledge from the literature review was combined with explorations in different available components. This resulted in the design that is presented in the following chapters. First, an overview of the system architecture will be given and some guiding design principles. In the next chapter, the separate components are further detailed and motivated.

5.1. Overview

To understand the design considerations made within the software, one must take a step back and look at the bigger picture of the architecture that is in the works at GeoPhy (Figure 5.1).

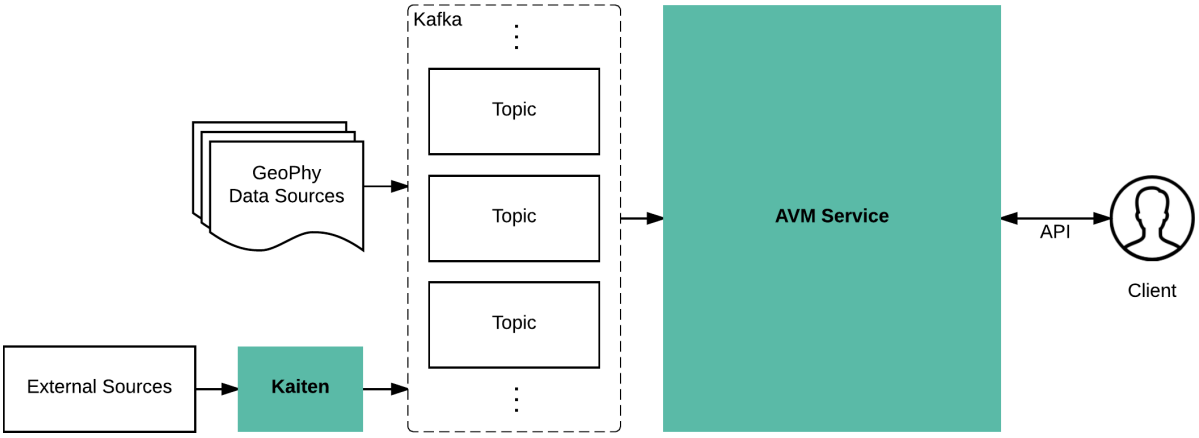
GeoPhy gathers data from many different places, processes it and produces new data. In an effort to organize all this data, they are working towards a system where all the different sources are brought together in one place, organized in topics, and accessible to all the services. Apache Kafka is the platform that enables this kind of data flow through the use of Kafka topics. Each *producer* of data can publish events on a Kafka topic. These events are constrained to *add* and *delete* operations. All the services can follow these events by becoming a *consumer* of a topic. As changes happen from the side of the producer, they are published on the Kafka stream. Kafka communicates the changes to the consumers, who update their view of the data accordingly. In this way, producers and consumers can function independently from each other; Kafka takes care of the persistence and reliability of the topics and is designed to handle large volumes of data, necessary to communicate these streams of updates.

Figure 5.1: Simplified view of GeoPhy architecture



The AVM system consumes all the topics necessary for training models and staying up-to-date. Still, there might be other features necessary to train the models (e.g. number of grocery stores in each neighbourhood) that are not yet published in Kafka topics. From the MoSCoW analysis, it was decided that the system should have one uniform interface to ingest data sources (section 3.3.1, requirement 7). Again, Kafka provides an outcome. By building a publisher (figure 5.2, Kaiten) from the extra data sources (e.g. CSV files, external databases), the system can ingest all these sources from Kafka topics. It then needs to join these different topics together into a dataset that the system can use to train models.

Figure 5.2: Simplified overview of the AVM system interacting with the GeoPhy architecture



In figure 5.2 the general overview of the system is summarized. Kaiten acts as a publisher from external sources to Kafka topics. Within the AVM Service, a data analyst can specify which topics are necessary for the model they are working on and the system consumes these topics to create a workable dataset. Then, the system itself trains the model, keeps it up-to-date and serves predictions to clients through an API. As events come in from a Kafka topic, the AVM Service keeps track of the changes and decides whether the changes imply concept drift. If concept drift happens, the AVM Service can decide to retrain.

5.2. Overarching design principles

In designing the AVM Service, aside from delivering on functionality (making it work), efficiency (making use of efficient distributed services) and compatibility (using Kafka), it was very important to keep the code reliable, secure and maintainable. Reliable, because the stakes are high in GeoPhy's production environment; paying customers expect great service. Secure, because data is a valuable asset. And maintainable, because the product needs to be used and built upon by other engineers after the project is finished. To reach this goal, the following design principles were given high priority: separation of responsibilities and simplicity of solutions.

Separation of responsibilities: if a task *can* be handled independently from another task, it should also *perform* this task independently from another. This makes the system reliable (if one task fails, the other can continue operating), secure (each task is shielded from the data used by other tasks and will only get the data necessary for its operation), and maintainable (changes only need to be made to the separated task).

Simplicity of solutions: when given the choice between an 'easier', but eventually more complex solution and a solution that takes more work at first but provides a more simple solution conceptually, the simple solution is preferred. An example of this is the choice of programming language: Python is widely used as a programming language for data analysis, but many distributed platforms and services perform better in Scala. An 'easy', but eventually more complex solution would be to use both in one system. Two languages need to be translated to each other inside the system and in the future, system engineers would need to be acquainted with two languages instead of one. The conceptually simple solution is to use only one language and put more effort in modelling. This makes the system reliable and

secure (complex solutions can easily create room for errors and intruders to slip through), and maintainable (simple solutions are easier to understand for other engineers).

The application of these principles and further design details go hand in hand with implementation. Therefore, these details will be discussed and explained together with the implementation details in the next chapter.

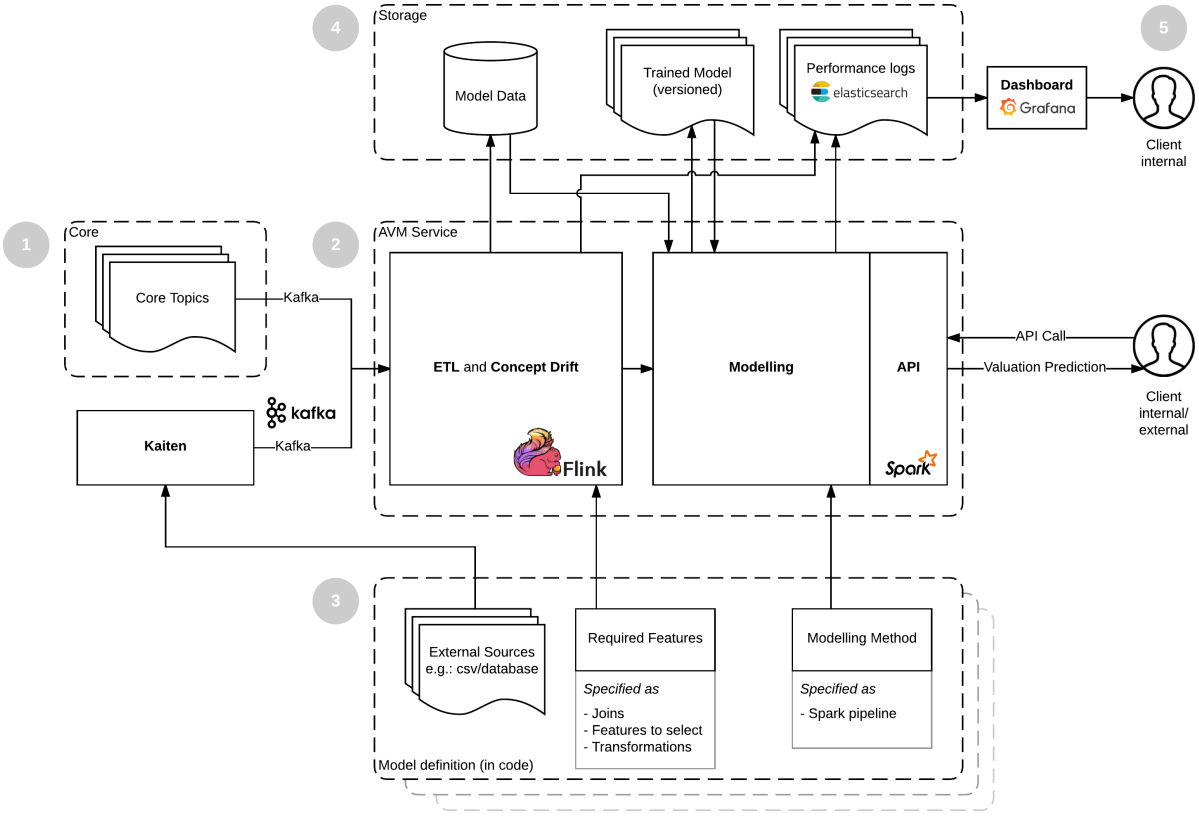
6

Implementation

The system will now be further detailed to provide explanations, motivations and in-depth analysis of each component.

In figure 6.1, the following groups within the system can be distinguished: (1) the data that enters the system through Kafka, (2) the AVM service that is delivered in this project (AVM Service section), (3) the 'interchangeable' model definition that can be expanded by GeoPhy data analysts and software engineers (Model definition section), (4) persistent storage used by the system (Storage section), and (5) the client side where the system provides predictions and gives insight into its performance.

Figure 6.1: Detailed view of the AVM system



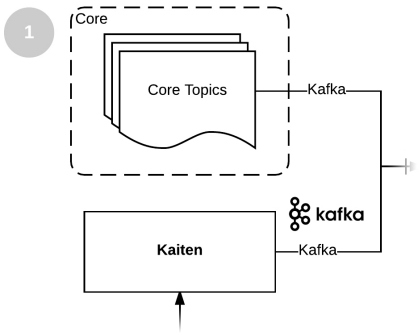
6.1. Programming language considerations

On a general note: Scala[1] was used as the main programming language. Scala is a language that allows for clean and readable code and combines object-oriented and functional paradigms for optimal functionality. Next to this, Scala is well supported and documented for both Kafka and Spark (a machine learning library). Python[2] was considered and used at first, because it is widely used in machine learning applications. However, Kafka support for Python turned out to be underdeveloped and it was decided to shift to Scala. This decision was first discussed with the data analysts, who requested that the methods they use for their models (XGBoost, Gradient Boosted Trees, etc.) would be supported. Spark has support for all these methods[3] and the data analysts agreed that this would be a good option.

6.2. Data entry

On the data entry side (figure 6.2), two main sources are detailed: the core topics, as explained in the previous section on GeoPhy's surrounding architecture, and the Kaiten package. Kaiten (from Kaiten-zushi, the Japanese term for sushi served on a conveyor belt) is in charge of building Kafka topics out of external sources. This is achieved with Kafka Connect[4], a tool within the Kafka platform that can move data from other services into Kafka and vice versa. The connector that is used for this is called Debezium[5] and streams changes from a database into Kafka topics. This is useful for production, when the system should be able to digest sources apart from the readily available GeoPhy Kafka topics, but also (and especially) in the development phase, when the GeoPhy Kafka topics are not yet available and all data sources need to be 'mocked' from CSV files or databases. With Kaiten, all necessary data sources enter the AVM Service through Kafka. The AVM Service can listen to changes on the data by consuming the Kafka topics defined for each model. This uniformity in data flow creates independence; as long as each system produces Kafka topics, they can change the internal implementation and even fail to operate, because all its output is contained within Kafka.

Figure 6.2: Detailed view of data entry part



6.3. AVM Service

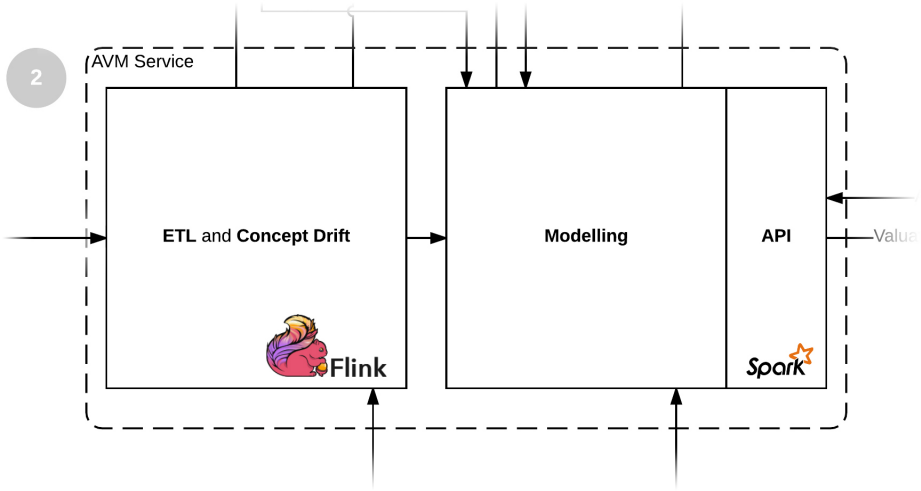
From the MoSCoW analysis, the following responsibilities have been appointed to the AVM service:

- Consume events/values from Kafka Stream.
- Train a model based on the configuration set by a data analyst.
- Version models and save old models for later use.
- Have a RESTful API that can provide valuations.
- Calculate predictions on an API call, using a trained model.
- Detect a significant change in distribution of the target value (concept drift) from the incoming data.
- Retrain a model when concept drift has been detected.
- Provide centralized places to set up the model and data streams.

[1] https://www.scala-lang.org
[2] https://www.python.org
[3] https://spark.apache.org/docs/latest/ml-guide.html
[4] https://kafka.apache.org/documentation/#connect
[5] http://www.debezium.io/

These responsibilities can be divided into three different components, which are reflected in the AVM Service group in figure 6.3: (1) ETL and Concept Drift (processing data), (2) Modelling, and (3) API. In the final product, these components operate independently and are set up in separate packages, following the separation of responsibilities principle. Below, the responsibilities for each component are detailed and it is explained what software was chosen to implement these responsibilities.

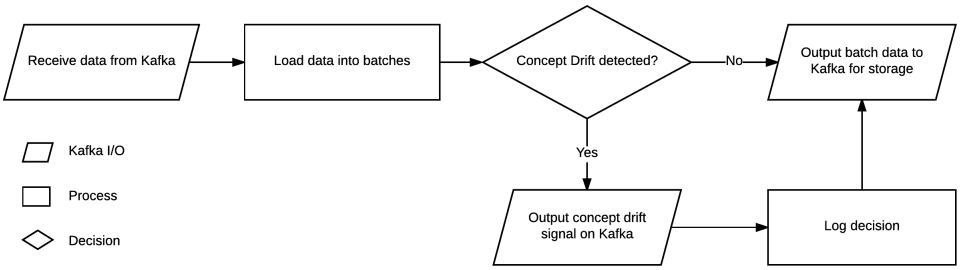
Figure 6.3: Detailed view of AVM service



6.3.1. ETL and concept drift

From the list of responsibilities mentioned above, the *ETL and Concept Drift* component deals with all the responsibilities concerned with loading (ETL) and processing data (concept drift detection). Figure 6.4 illustrates the flow of data through the ETL and concept drift component. The component consumes events from the Kafka stream, processes the events in batches to detect concept drift, joins the necessary topics from Kafka and outputs this to Kafka again. A Kafka connector builds a persistent database from this output stream that can be accessed by the modelling component later on. The ETL component sends the data back into Kafka, instead of writing it to the database itself, because Kafka provides tools to build a persistent database from topic streams through Kafka Connect. The ETL component joins and transforms the topics, but does not need to 'worry' about building a persistent database. Another reason for this is the simplicity principle: this way, all data flows through Kafka.

Figure 6.4: Data flow in the ETL and Concept Drift package



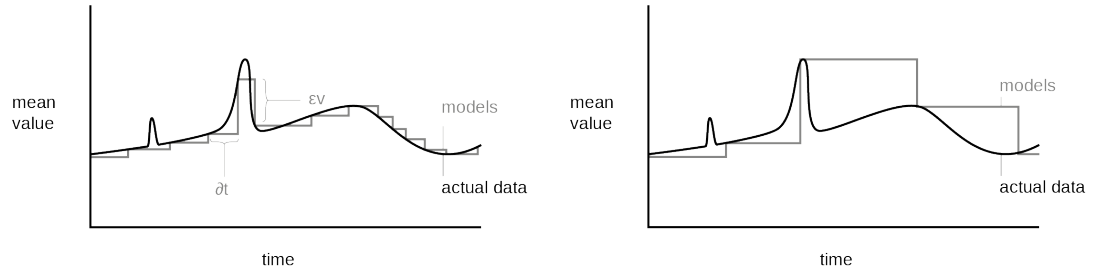
When concept drift has been detected, a signal is sent over a Kafka topic that is consumed by the modelling component. This, again, is done because of simplicity (one data flow) and independence. Even if the modelling component is unable to 'listen' at the moment concept drift occurs, it receives the signal when it is able, because Kafka stores the message. In order to detect concept drift, change detection algorithms (see 4.6.1) were used, specifically

the Page Hinckley test. The choice to use these trigger-based methods is due to the fact that the model training is done using batch processing. This mode of processing precludes the use of adaptive models. Training windows were also not used as they require holding a lot of state in memory, which does not scale very well. The choice for the Page Hinckley test within these change detection algorithms was based on its good documentation and simplicity of concept. The code is written in such a way that other tests can be added later on.

The concept drift detector can be configured with three parameters to set the batch size and sensitivity of the detection algorithm. Currently, logs are produced on the number of decision per time unit and the parameters that were used with these decisions. Another metric that is logged, is the performance of the latest and previous models on the latest data at the moment concept drift is detected. All these metrics are logged directly from the component that makes the calculations or decisions, to ensure that all necessary information is logged, even when another component fails. Using these metrics, the concept drift detection can be fine-tuned in production.

Fine-tuning parameters

The process of fine-tuning these parameters requires some extra attention. Figure 6.5 shows a hypothetical progression over time of the distribution of a target variable given by its mean value. The grey line in these graphs shows the distribution of that same target variable as reflected by the latest version of the model. This line progresses stepwise, as models maintain their state (and the data distribution) until they are retrained to reflect the actual data. When the concept drift parameters are configured well, the grey line follows the black line closely (figure 6.5a), without following each small peak. Concept drift performs badly when it shows behaviour as seen in figure 6.5b. εv, the error in mean value between the model's data and the actual data, is very high in some cases, which will result in low accuracy for the model predictions.



(a) Good concept drift sensitivity

(b) Very low concept drift sensitivity

Figure 6.5: Concept drift fine-tuning. εv is the error in mean value between the model's data and the actual data. δt is the time between concept drift detections.

Because of this, concept drift parameters are initially set to be very sensitive (a very low allowance for εv). High accuracy is more important to clients than efficiency, so it is better to retrain the models more often than necessary. In order to achieve higher efficiency, the approach would be to maximize the allowed εv, while still maintaining *acceptable* accuracy. This boils down to the following question: what is an *acceptable change* in accuracy? The answer to this question is a business decision. If for example, competitors provide predictions with an accuracy of 5% (MdAPE) and the own model is known to perform with an accuracy of 4%, it could be decided to accept a 1% change in accuracy in order to stay ahead of the competition.

This question leads to another question that is relevant to this system and its implementation: How can this change in accuracy be measured? One way to answer this question is to look at the difference in performance, δp, between the newest and the previous version of the model on the latest data. This has been implemented in the system by logging the performance of the latest model and the previous models when a concept drift detection has been made. The δp that can be found in these logs should be compared to the previously

decided acceptable change in accuracy and with this information, the sensitivity parameters can be set to either allow for more or less allowance in accuracy. The question of measuring performance is discussed in further detail in chapter nine.

Choosing software

The ETL and concept drift component needs to be able to process large volumes of data coming in from the streams and it needs to do this in batches based on count: Although the incoming streams are voluminous, the events coming in are not evenly distributed over time, as updates are usually triggered by a new batch of data being incorporated in the CoreDB. This means that a batch window based on time would not be effective: some batches would be enormous and others would be empty. The other option is to base the batch window on count, which is done in the system.

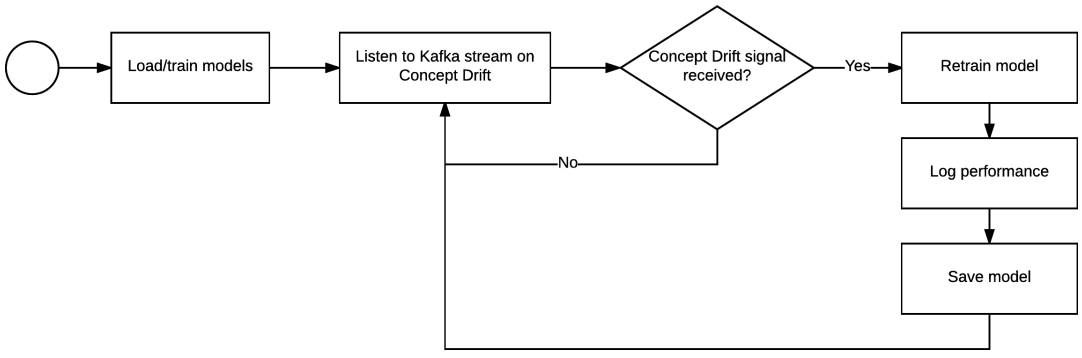
Knowing these requirements, different stream processing platforms were considered. Spark Streaming[6] was the first candidate. Spark Streaming is part of Apache Spark, the platform used in the modelling component, and is designed to handle large volumes of streaming data. Using the same package for ETL and modelling would make a lot of sense, considering simplicity. However, Spark Streaming cannot base batch windows on count. Therefore Apache Flink[7] was chosen. Apache Flink is a modern, distributed, and open source stream processing platform that has the ability to do batch processing based on count. It works well for large scale data processing applications, as is described below, in the section on scalability. Another consideration for choosing this platform is that, like Spark and Kafka, Flink is actively supported by Apache. This means that it is a safe bet for the future.

Flink runs as a standalone service. The code for the ETL and concept drift component is first compiled and then uploaded to Flink as a Flink job. Flink then takes care of distributing the tasks, as described in the section on scalability below.

6.3.2. Modelling

The modelling component is responsible for training models, saving and versioning models, and returning predictions. Figure 6.6 details the steps taken by the modelling component. On start-up, it either loads or trains the models that are defined by the data analysts. If a version of the model has been saved, it loads it from disk and if no previous version exists, it will start training. This can lead to a long start-up process, but the steps are necessary, because the trained models need to be available at any time. After all the models are loaded, this component starts to listen to the Kafka topic on concept drift. When the ETL and Concept drift package has detected concept drift, it will produce a signal on that topic, which the modelling package will receive. When this happens, the affected models are retrained, saved and the performance is logged. It then goes back to 'listening' to the Kafka topic.

Figure 6.6: Flow of processes and data in modelling package



The main criteria for selecting a software platform to support the modelling package are the ability to train models with large volumes of data (millions of buildings), professional

[6] https://spark.apache.org/streaming/
[7] http://flink.apache.org

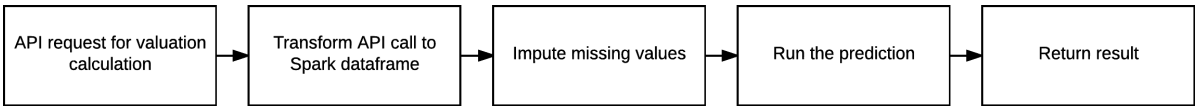
support and documentation, and support for widely used machine learning methods. As discussed before, Python and its machine learning library were also considered, because they are already used by many data analysts. As explained before, Scala turned out to be a better option for the main programming language. This choice of language, combined with the requirement to handle large volumes of data in a distributed environment led to the selection of Apache Spark[8]. It is designed for distributed systems, supported by Apache, and has support for all the machine learning methods requested by the data analysts in its machine learning library. Spark is well documented and from a quick Google search, one can deduce that Spark is widely acclaimed.

Spark runs as a standalone instance inside a virtual machine. The code for the modelling component is compiled and sent to the Spark master as a Spark job. The Spark master distributes the job to several worker instances. When these workers are done, the Spark master aggregates the results (e.g. a trained model, a prediction from a model), as described below in the section on scalability. Models are trained and saved within the Spark virtual machine.

6.3.3. API

The API component is responsible for handling client requests and retrieving predictions from the trained models. The code for the API component is also sent to the Spark master as a compiled Spark job. API requests are made over HTTP in a JSON format to the Spark instance. Missing columns in the API request are imputed with an average and Spark then runs a prediction on the specified model (figure 6.7).

Figure 6.7: Steps taken when an API request is made



6.3.4. Connecting ETL, concept drift and modelling

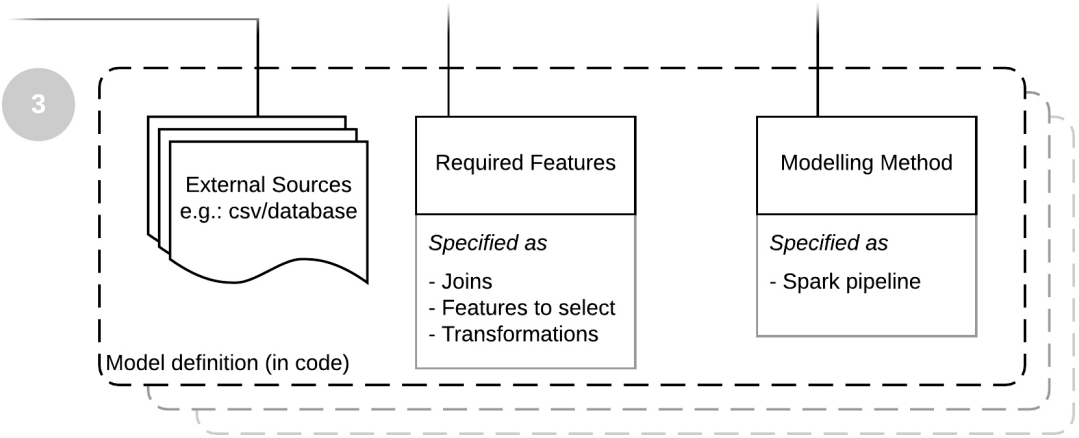
The ETL and Concept Drift responsibilities are joined in one component. Data flows through the concept drift detection into ETL and is then streamed through Kafka to a database. The modelling component can communicate with the database on its own through a JDBC connector (JDBC provides access to many different databases through Java). Concept Drift detection signals are sent over Kafka to the modelling component.

6.4. Model definition

The AVM Service takes care of data flow from Kafka to the machine learning methods. It makes sure all models are up-to-date and that clients can access predictions through an API. The AVM Service does not, however, automatically decide which machine learning methods it should use or what data it should source for its models. This is where the GeoPhy data analysts come in: They explore different configurations of features and models to find the best fit. In order to quickly iterate, they use software like DataIku, where they can visually put together models, or Python and R notebooks, where they can document their process as they code. After the exploration phase, the data analysts needs to define their model in such a way that the AVM Service can use them. This is done in the model definition part of the system (figure 6.8). A model is defined as a sequence of stages (a pipeline). Each stage consists of a transformation on the data. Some examples of these transformations are: imputing missing values (a custom transformation created for this project), transforming the feature columns to a feature vector, and transforming categorical values to numerical values. The final stage in this pipeline is the creation of a model from the data. This is done by defining a model object from Spark's machine learning library and adding it to the pipeline. The definition of this model is written within the package for the modelling component by creating a class

[8]https://spark.apache.org/

Figure 6.8: Detailed view of model definition part

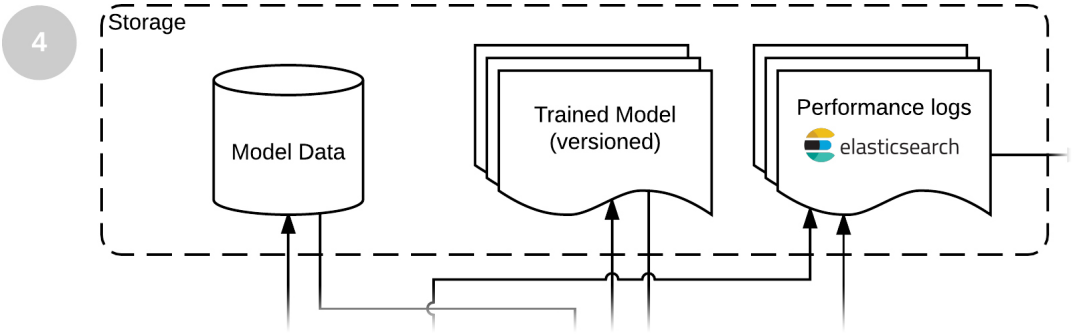


for that specific model that extends from a Model trait. This Model trait holds all reusable functionality, like the steps needed to load, train, and save a model. The features that are necessary for a model are defined as a separate class that extends from a DataSource trait. In this class, the column names need to be defined and the type of action to take when a value is missing or incorrect in the provided data. This DataSource class is used in the Model class to define the data a model will use. This makes it easy to reuse a modelling method on different DataSources or to use different Models on the same DataSource.

External sources need to be loaded into Kaiten, which is done by loading them into the database that Kaiten reads from. Then, the ETL and Concept Drift component also need to know what data should be combined and watched for concept drift. Each model can define a class extending from DataProcessor that defines what should be done with the data.

To summarize: a data analysts has a list of features and data sources for a model. These features are defined in a class extending from DataSource. The system knows how to gather and process the data from a class extending from DataProcessor. Then, the data analyst defines a pipeline within a class extending from Model that prescribes the stages of transformations for the data.

Figure 6.9: Detailed view of storage



6.5. Storage

The system uses persistent storage for three purposes: saving model data, saving trained models, and logging (figure 6.9). First, the *model data*: As described before, the ETL and Concept Drift component output the model data on Kafka. Kafka Connect then loads this data into a database for use by the modelling component. A simple Postgres database is used for this, as it can handle all write and read operations and can already interact with Spark through JDBC. Then, the *trained models*: These models are saved in a format written by Spark. Each version gets its own folder on the disk, named with a unix timestamp. By

listing the folders on the disk, the system can find the latest version and load it when necessary. Finally, the *performance logs*. These logs are saved to Elasticsearch[9]. Elasticsearch is used, because it makes it possible for other applications to search through the logs and create insights even when the logs grow to enormous sizes. Another advantage of using Elasticsearch, is that there are readily available packages to visualize the data in Elasticsearch, as described in the next section.

6.6. Client side

On the client side, there are internal and external clients consuming the API and there is the internal client (GeoPhy) checking the logs (figure 6.10). The API has already been explained in the API component above and does not need further explanation. GeoPhy can inspect the logs and find insights through a dashboard where the logs are graphed and visualized. This dashboard is built with Grafana[10], a good looking, open platform for analytics and monitoring. It is also very easy to set up, as it provides a GUI to add new sources to visualize and it supports Elasticsearch as a data source. On this dashboard, GeoPhy can track the number of decisions made by the concept drift detection and follow the performance of the models.

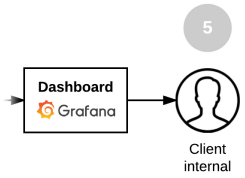
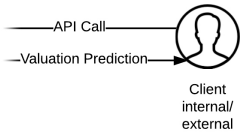


Figure 6.10: Detailed view of the client side

6.7. Technical details

This system brings together many services and technologies. Apache Zookeeper (used to manage Kafka), Apache Kafka, Apache Spark, Apache Flink, Elasticsearch, Grafana, Postgres are all services that need to be installed and started for the system to run. In order to make this easy and scalable, the system uses Docker[11]. Docker runs virtual images of machines that have the services pre-installed. All that is necessary to run the services is a simple command (`docker-compose up`), after which the project can be run. Docker maintains the images, starts them up and even makes it possible for the images to run on multiple machines. The code for the components can then be compiled and uploaded to the responsible Flink and Spark instances (figure 6.11). The machines use persistent storage, so the code does not have to be uploaded each time the instances are restarted.



In order to compile the code and to keep track of all the dependencies, SBT[12] (Scala build tool) was used. Gitlab was used to collaborate on the code, track issues and perform code reviews. Code reviews were extensively used throughout the process as sprints were finished and when merge requests were made.

6.7.1. Scalability

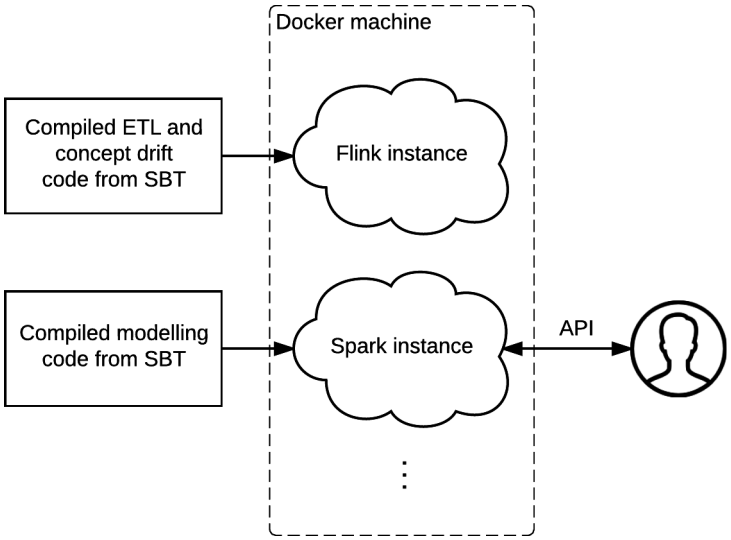
Scalability is defined as the ability of a system to handle an increased amount of work[5], and elasticity refers to the ability of a system to manage its resources to meet demand, meaning that resources can be provisioned or de-provisioned in an autonomous manner[14], as it needs to be able to handle large amounts of input data and valuation requests, often in short concentrated bursts

Kafka

Since Kafka registers its cluster nodes (called brokers) with Apache Zookeeper[13], Kafka can easily be scaled to handle increased message traffic by spinning up more brokers. These brokers each register with Zookeeper, and are then discovered by the other brokers. Kafka also has fault-tolerance mechanisms for de-provisioning brokers. Kafka replicates messages

[9] https://www.elastic.co/products/elasticsearch
[10] https://grafana.com
[11] https://www.docker.com
[12] http://www.scala-sbt.org
[13] https://zookeeper.apache.org

Figure 6.11: Overview of how Docker and SBT are used



across brokers, with a replication factor set per message topic, meaning that, when a broker is spun down, the other nodes will be able to serve the messages. The aforementioned abilities to easily provision and de-provision Kafka mean that it is not only scalable, but also able to change resources on the fly. Unfortunately, this dynamic allocation is not autonomous, but it is easy to manually alter the cluster.

Flink

Apache Flink is designed to be scalable. As such it can be deployed in a cluster with two types of nodes: JobManagers (responsible for coordination and scheduling) and TaskManagers (responsible for executing job tasks). Programs can parallelise tasks in order to maximize throughput, and Flink scales tasks linearly with the number of CPU cores in the cluster. Concept drift can be parallelised by separating the concept drift detection for each feature into a separate task. When scaling up or down to match demand, Flink jobs need to be stopped. The cluster nodes are scaled up and then the job needs to be restarted. Stateful operations in the jobs can be saved in order to ensure that the job restarts with the same state it had when it was stopped, which is then redistributed amongst the new collection of cluster nodes. This means that Flink is not yet elastic, however, while this is being written, dynamic scaling is in the Flink project roadmap.

Spark

Like Flink, Apache Spark is also scalable. In a similar fashion, it has cluster managers that coordinate and schedule, and has executors that run the actual job tasks. Spark scales linearly with the number of executors it has, meaning that it can handle heavy workloads. Spark also supports dynamic resource allocation, which means that it is able to provision and release job executors depending on the workload when running in a cluster. This means that Spark is truly elastic, as this dynamic allocation is autonomous.

7

Results

The AVM system was implemented in a six-week development span, after a three-week research and design phase. In over 250 commits and 30 merge requests, all the different components were explored, studied, and put together. A big part of this time was spent dealing with these new systems and finding ways to make them work well together. As the process furthered, the project definition changed from an actual modelling project toward the system that is now presented. In this chapter, it will be examined what this has lead to. How should the success of this project be tested? Does the code live up to quality standards and what are the ethical implications of this system?

7.1. Testing for success

Towards the end of the project, a plan was set up to test the success of the project, based on the requirements as described in the problem analysis. Each component's desired functionality was described as a user story that could be tested by either running the project itself or writing and running unit tests. This testing plan can be seen in table 7.1, where each individual testing step is detailed. All the testing steps were executed and deemed successful. One can find the performance results in the table 7.2. From these successful tests as well as the design considerations that were made along the system requirements, it was concluded that the project succeeds in fulfilling the requirements (all the must and should requirements) that were set out at the start of the project.

7.2. Code quality

Maintainability is an important part of the quality measures proposed by the ISO group for systems and software engineering. In order to verify the quality of the code in terms of maintainability, the Software Improvement Group (SIG) reviewed the project. The code scored 4.5 out of 5 stars on SIG's maintainability model, implying that the code scores above average. The full 5/5 score was not yet achieved, because of unit size, a measure for the size of individual methods and functions. The size of methods in the code was generally small enough, but the model setup method, for example, proved to be too large. This was done partly by design, as it was aimed to use only one method to define the models, so data analysts would only have one place to work in. In versions of the code after the SIG review, the larger methods were refactored to smaller individual methods to incorporate SIG's feedback. The model setup is still performed in one class, but different responsibilities have been split in different methods. For the final code submission, a testing coverage of 86% was achieved, fulfilling the requirement to have at least 80% testing coverage.

Table 7.1: Testing plan

Component	Story	Succesful When	Testing steps
Model training	A GeoPhy data analyst can transfer their model to the system and the system can train the model to receive a performance close to their original model.	• The model training happens in reasonable amount of time (<30m) • The model performance (MdAPE) is within 5% range of the trained model in the drafting system.	• Write the model • Train the model • Run a prediction on a validation set • Note time/performance
Model prediction	A user can call the API, and will receive a prediction for their house that is made by the trained model.	• An API call can be made to the service • A prediction is returned within 10s	• Write an API call with data from validation set • Run the API call on the system • Note time
Concept drift	Models are automatically retrained when it is detected that the distribution of the target variable (value) has changed significantly.	• The concept drift detection model detects concept drift • The concept drift model notifies the model server of concept drift • The concept drift model filters out noise	• Feed the concept drift detector changing data, check if it signals • Feed the concept drift detector noisy data, check if it does not signal • On signal, check if it notifies the model server • Generate logging on this performance
ETL	A GeoPhy data analyst can specify datasources for their model and the system loads this data through Kafka streams into a persistent database.	• Data is automatically loaded from the Kafka topics into the Postgres database • Data is automatically updated in the Postgres database, based on the Kafka topics	• Generate Kafka stream from Kafka connector • Check if data in database is updated
GeoPhy usability	A GeoPhy data analyst is briefed so they are able to add models and specifiy datasources for these models.	• GeoPhy data analysts are able to understand how the system works • GeoPhy software engineers are able to implement a model in the system • GeoPhy data analysts are able to monitor the performance of the model	• Present the system to GeoPhy staff • Implement a model together with a data analyst • Write a manual (project report) and ask for feedback
Deployment	The system is deployed in a production-like setting (multiple cores, computers).	• The system is able to run in (an) Amazon Web Services instance(s)	• Deploy the system on AWS

Table 7.2: Testing results

Component	Testing steps	Results
Model training	Write the model	Successful
	Train the model	Successful
	Run a prediction on a validation set	Successful
	Note time/performance	14m, MdAPE: 5.12% (compared to 5% of original). Successful
Model prediction	Write an API call with data from validation set	Successful
	Run the API call on the system	Successful
	Note time	9s, this time is the same when multiple houses are requested at once. Successful
Concept drift	Feed the concept drift detector changing data, check if it signals	Automated testing, successful
	Feed the concept drift detector noisy data, check if it does not signal	Automated testing, successful
	On signal, check if it notifies the model server	Successful
	Generate logging on this performance	Successful
ETL	Generate Kafka stream from Kafka connector	Successful
	Check if data in database is updated	Successful
GeoPhy usability	Present the system to GeoPhy staff	30th June
	Implement a model together with a data analyst	To-Do
	Write a manual (project report) and ask for feedback from staff	Successful
Deployment	Deploy the system on AWS	Expected successful, Docker handles this

7.3. Ethical implications

When the system goes into production, it will have an indirect impact on people's personal lives. Companies that use the valuations produced by the system, could base decisions to extend loans, what to do with their portfolio, and how they impact their customers on these valuations. The ethical value of this influence is twofold: more accurate valuations can prevent 'bubbles' of incorrect predictions that build to a burst and cause economic misfortune. On the other hand: when the system is too reactive and incorporates every change or peak in values, it is prone to overfitting. This could, for example, mean that a small dip in property values is directly incorporated in the valuation model and can lead to predictions that show an exaggeration of this dip, causing unnecessary panic. This impact needs to be taken seriously, as people's homes and livelihood are at stake. This is done by allowing GeoPhy to look into the concept drift logs and tune the models, giving more accurate predictions and checking the performance of the system as it is operating.

The importance of accurate predictions is intertwined with the ability to explain predictions. As seen with the European General Data Protection Regulation (GDPR), legislations are requiring organisations that implement algorithms to be able to explain how a decision on a data subject was made. Aside from a legal requirement, this is also an epistemological right: people have the right to know an explanation about decisions that are made about them. Mittelstadt et al. mapped the debate on ethics for algorithms[27]. They created a topology of injustices that can be committed with algorithms. When decisions cannot be comprehended, this is the injustice of inscrutable evidence. Many machine learning methods (like the neural networks, as discussed in the literature review) behave like a black box: the models are trained through a logical process, but the resulting model predicts values in an obscure way, making it inscrutable. Therefore, the focus of this project was the use of machine learning models that had some explanatory power. The decisions made by this system should be explainable, at least to the degree that clients can understand how a decision would be made in general. This is possible with decision tree algorithms, as it is relatively easy to explain.

Another consideration is privacy. This system uses a lot of data about a lot of different properties. Most of this information is already available online through real-estate websites and government records, but privacy could still be an issue for some attributes. Therefore, data is not based on persons, but on properties and measures were taken to make sure the data is securely stored and shielded from malicious attacks. Some of these security measures during development were, for example to not upload any data on Git, but to transfer the data with USB-drives. Next to this, GeoPhy's internal network is shielded from outsiders and the valuations can only be retrieved through the API call.

8

Conclusion

As GeoPhy is developing its business model and looking into the future of automated valuation models (AVM), this project has attempted to deliver a proof of concept of a system that automates the training, maintaining, and delivery of machine learning models for automated valuations. In order to achieve this goal, first the situation and problem was analysed. This resulted in an outline of the desired product and requirements in the form of a MoSCoW analysis. An important goal for this project was to incorporate streams of data from Apache Kafka into a service that would train and update models automatically. The second goal for this project was to keep track of the changes in the data in order to detect significant changes in distribution (concept drift) of the target prediction value.

These subjects were studied in literature, reviewing existing and upcoming valuation practices in real-estate, steps needed to perform machine learning tasks, architecture to support big data processing, and concept drift. This resulted in a design, described in chapters five and six.

The project goals were achieved by combining several different services into four different components:

- Kaiten component: Kafka connect service that outputs external sources to Kafka topics.
- ETL and concept drift component: Apache Flink service that processes incoming data to detect concept drift and transforms the data to be used by the modelling package.
- Modelling component: Apache Spark service that creates Spark jobs to train models, save them and update them based on the data and concept drift signals from the ETL and concept drift package.
- API component: Apache Spark service that creates Spark jobs from incoming API calls.

These components use storage in the form of a Postgres database, disk storage and Elasticsearch logs. The logs (on model performance, concept drift usage) can be interpreted through a Grafana dashboard, which is editable through its own GUI.

Finally, to test the success of the project, a testing plan was set up and the code was reviewed by an external group (SIG). The code achieved all the testing milestones, has 86% testing coverage, and received a 4.5/5 in a mid-development review on maintainability. The project is ready to be delivered to GeoPhy and will be explained to the data analysts and software engineers through a learning session. With this project, the concept of automated valuation models inside GeoPhy's new architecture has been tested and proved and the project is ready to be further developed and used in practice.

9 Discussion

The code is delivered and many of the choices have been motivated, but there is always room for improvement and discussion. This chapter revisits some of the choices made in design that others might see or do differently. Recommendations to suit these remarks will follow in the next chapter.

9.1. Over-engineering

One might take a look at all the services and platforms that were used for this system and argue that the sheer amount means the system was over-engineered. Why not just use a simple machine learning library and a database? Why are distributed applications like Kafka, Flink and Spark used? Don't they give a lot of useless overhead? First of all, Kafka is an external requirement and fits the needs of the company well. This was not a point of discussion. Spark is also necessary to handle the volumes of data used in production. Currently, the models are built on 80.000 to 800.000 entries, but in the near future, the models should be able to support millions of entries. Therefore, using Spark is futureproof; it can handle this many entries, because it does so in a distributed fashion.

Then why use Flink? Granted, the streams of data coming in are measured in a number of changes per day, but there are situations when GeoPhy acquires a batch of new data and this new data is all pushed to the Kafka stream. The ETL and concept drift section should be able to handle this amount of data. Flink is made for that job. The other option would be to create a separate batch processing section for this project that is more lightweight, but this would take a lot of time and would have to be replaced when the streams do increase in size eventually.

Another advantage of using these large platforms is that they are designed to follow the industry standard or they set the industry standard. Using the platforms forces the design of this system to follow these standards and design paradigms (e.g. master-worker in Spark, connectors and sinks in Flink), and these paradigms in turn are used, because they facilitate high-quality software engineering as is demonstrated by the critical acclaim of these platforms.

9.2. Storage

During the project, the question arose whether it was a good fit to use Postgres to store the modelling data. The main reason to not use Postgres, is that it adds extra overhead to the access of data when used by Spark. Spark reads the database every time it trains a model and does so several times within a training routine. On the development computers, this added a considerable amount of time to the training process. Why not write to a distributed file system or database from Flink that Spark can *directly* use, as Spark is the only 'consumer' of this data? It was decided to continue with Postgres during development, as an overhead for reading and writing the database would be inevitable. Also, on faster production machines the overhead would have less impact. After some tweaks to the fetch- and partition size of the JDBC connector, the database performance came closer to that of a CSV read and the problem of high training times was solved. Aside from these overhead considerations, it is desirable to have all the functionality of a database that Postgres provides, like data integrity, persistence, and a schema. Postgres has good integration with Flink and Spark and was already implemented. For future development, other options could still be explored that make use of distributed data storage.

9.3. Concept drift

The concept drift section is a novel addition to the AVM pipeline. Therefore, there are some areas that need extra development or thought.

9.3.1. Market cycles

First of all, a much heard request from GeoPhy's customer is the ability to detect sudden drops in the market. The housing market tends to follow cycles of about eight years each. The market grows during the cycle and property values rise until the market crashes and property values suddenly drop. This is one of the situations where concept drift should trigger and retraining should start. However, the nature of this situation makes it hard to detect the sudden drop: the crash of the market means that properties are sold less, which means that there are (almost) no new transactions to base the decision on. The current implementation of concept drift is not able to detect this situation, because it is dependent on new data. Detection of the start or end of cycles needs to be implemented in another algorithm inside the processing layer. This algorithm should probably take in other streams of data, like market analysis. It might even conclude from the absence of data that a cycle has ended. The system that is delivered is easily extensible to incorporate this extra processing layer.

Another problem with the situation of the end and start of cycles is the question whether to use only new data to train a model and to discard data from the previous cycle or to train on all the available data. How should a system know for sure that a new cycle has started? What data should it discard? One solution could be to maintain two models when the system has the suspicion that a new cycle has started. One model that uses all the previous data in case the suspicion was faulty and another one that is trained on only the latest data in case the suspicion was correct. This latter model will be less accurate, as it has to be built on sparse data, but it could be presented alongside the 'conservative' valuation as a 'cutting edge' valuation.

9.3.2. Performance

Another problem with concept drift is the question how to measure its performance in order to fine-tune the algorithm's parameters. Concept drift performs well when the (re-)trained models give accurate predictions for *reality* and when models are not updated superfluously. One way to attempt checking the accuracy with reality, is to predict valuations for a separate validation dataset at that moment. This, however, can be misleading: when a validation set is used to check the accuracy of the models, the data is indirectly used to check its own accuracy. It should not come as a surprise that a dataset is an accurate reflection of itself.

One solution would be to track a concept drift decision over time. The accuracy of the dataset would then be tested and logged on a validation set at the moment of retraining, but

also at later times after the data has received a number of updates. If the accuracy of the model suddenly drops, this means that the concept drift detector responded to a peak in the data that does not reflect reality. In order to make this possible, a separate validation dataset needs to be maintained that is used for each validation interval. It can never be used for training (validation results would become skewed), the dataset should be updated as changes come in, and it should be a good representation of the entire dataset.

Another solution would involve tracking user feedback. If users can give feedback on their valuation in some way, this would be a good reflection of the performance of the model in reality. GeoPhy could see user ratings for each concept drift decision and decide whether the decision was faulty or correct. This, however is not a feasible expectation from the customers: GeoPhy's customers do not provide feedback on the valuations; they just receive the reports and choose a partner to work with based on model performance.

This project has partly incorporated the first solution: it measures the performance of previous versions each time the model is retrained. In this way, the performance of one version is tracked over time and compared to more recent versions. It remains to be seen what the negative effects of too many concept drift decisions could be aside from efficiency. The machine learning models that are used have been designed to avoid overfitting and tend to make good generalizations. Retraining the model on the entire dataset each time an update comes in could still provide good predictions. Concept drift is mainly put in place to avoid updating the models and using a lot of processing power each time an update comes in. Therefore, the current metrics (number of detections over time, previous and current model performance on latest data) provides sufficient insight into concept drift for now and can help fine-tuning the parameters that determine what should be considered a 'change in distribution'.

One important take-away from the above mentioned problems is that the concept drift section is not yet ready to be fully left to its own devices. It needs to be tested and monitored, because the production environment has such high stakes. For the design of this system, attention was given to this by spending extra time on logging, writing them to a searchable data structure (Elasticsearch), and making these logs interpretable through a dashboard.

9.4. Training models on startup

All models in production are either loaded from disk or trained on startup. This is done, because the models should be accessible for predictions at any time. It does, however, mean that startup can take quite a long time and that the system is unresponsive for this startup time. A solution to this is prioritize the loading and training of some models. High priority models are trained right away, while low priority are trained on a separate thread after the other models have been loaded. This means that the low priority models take a longer time to train, as a part of the processing power is then dedicated to serving predictions. Since the models have lower priority, this is deemed okay. This solution is not implemented in the current system, as there is only one model to be implemented and this prioritization is not yet relevant.

9.5. Usability in GeoPhy

The current system provides several places inside the code to setup a model. It is easy to reuse sections of code and extend the provided traits in order to build new models, but some knowledge of the system and programming paradigms is still necessary in order to make it work. This makes it harder for data analysts to deploy their models to a production environment. This report can give insight into the workings of the system, but a simpler interface to define models could go a long way to help speed up the creation and production of new models. Some solutions for this problem are recommended in the next chapter.

9.6. System size on disk

In order to run the project, quite a few services have to start up and the images have to be loaded. The code that was written for this project is lightweight and easy to maintain

(as demonstrated by the high SIG score), but all the added services do contribute to quite a large system size. One of the reasons this has happened, is the timespan of the project. Docker images can be custom built, but there are readily available Docker images for most of these services that have been tested and pre-built. Building custom Docker images takes unnecessary time, as these pre-built Docker images prove to have higher support and have a quicker turnover for development. For now, these Docker images work fine, but as can be seen in the recommendations, it might prove helpful to build custom Docker images that are tweaked for GeoPhy's usecase.

10 Recommendations

The discussion has left some problems open for development. In this chapter, the solutions to these problems will be explored and some additional recommendations for the development of the system are listed in order to hand off the project to GeoPhy for use in their business.

10.1. Storage

The Postgres database works well for the current system, but when the database grows to even larger sizes, it is recommended to test out distributed databases (like Apache Cassandra). The database will preferably have a direct interface for Spark to manipulate and access the data without too much extra overhead.

10.2. Concept drift

The solutions discussed in section 9.4.1 on market cycles should be explored more. A model to detect market cycles is a separate model from concept drift and could be developed. This model has more of a predictive function than the descriptive power of concept drift. When this market cycle detection works, it is recommended that two models are maintained when a new cycle is detected: one with all the available data as a training set (conservative model) and one with only the latest data (cutting edge model). When a new cycle is detected, the conservative prediction is shown together with a notification that a new cycle might be on hand and the cutting edge prediction belonging to that cycle. This should be done with care: when the system detects too many new cycles, users start to mistrust the system, even when it is correct.

In order to measure performance, a metric should be developed to measure the success of concept drift in its accuracy of modelling reality. The solutions proposed in the discussion (section 9.4.2) can serve as a start to developing this metric. This requires a separate *global* validation set that is used to track the performance of a model over time. For now, the 'detections per time unit' metric serves well to monitor the eagerness of the concept drift algorithm and the machine learning methods are considered to generalize well.

10.3. Training models on start up

When more models are added to the system, it is recommended to prioritize the training of models as described in section 9.4. This prioritization could even be implemented in automated fashion: as the system detects more API calls for a model, the priority of that model is increased.

10.4. Usability in GeoPhy

In order to improve the usability of the system for data analysts, all the configuration settings can be moved to one location. One config file, where the necessary features, processing steps, and modelling methods are defined. This makes it harder to customize and tweak elements, which is a drawback in the production environment were efficiency is more important than usability. The costs of the time used by data analysts or software engineers needs to be weighed against the cost of running non-optimal models in production in order to make these decisions.

It should also be considered whether it is necessary to have interchangeable modelling methods. Most machine learning methods can be used for many different tasks; the differentiating factor between different models is the set of features. It could be opted to only allow for the creation of feature-sets to define models, which simplifies the creation of models, but does remove a level of independence for the data analysts.

Moving all the settings to one file could facilitate the development of a GUI to set up models. The GUI just needs to output one config file, and the system then uses this to run the models.

10.5. System size on disk

As described in the discussion, some Docker images require a lot of space. It is recommended to explore what Docker images can be replaced with custom images and to create these custom Docker images. The images should be set up with only the necessary components and can be tweaked to the system's needs.

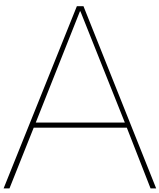
10.6. General recommendations

- In order to simplify the conceptual design of the system, it is recommended to follow the development of Spark Streaming. When Spark Streaming enables the definition of batches based on count, it could be a good move to move from Flink to Spark Streaming. This takes away the hassle of compiling and uploading code to different systems, conflicting dependencies, and different development schedules for both systems.

- The API imputes missing values with averages. This could be extended by looking up the property in the database and imputing missing values with the values in the database if the property exists in the database. The other way around: If the API request provides new information on a building, this could be incorporated in the database.

- Build out the Grafana dashboard in order to create more insight into the usage and performance of the AVM Service based on the logs. This can be done through the Grafana GUI.

- An important part of valuations are the reports that support them. Automated reports could also be generated from the system, as an extension to the existing functionality. This also contributes to the effort of explaining the decisions made by the algorithms that are used.



Original project description

Predicting the value of single family homes in the US by means of an automated valuation model. The valuation model will run on approximately 200M single family homes in the US and should, in order to be useful, at least perform better then the accepted error rate of 10% for human appraisals whilst giving a better understanding of the composition of the valuation. We already have access to a vast amount of data including a working valuation model for Multi Family homes in the US that can be used for reference.

A.1. Challenges

Early studies in literature and within GeoPhy have shown that any single statistical model for predicting single home values the type of model best fit is strongly dependent on external conditions like the location, time, availability of data. This results in our modelling team needing to do extensive study before being able to choose the best fitting model. We see a lot of promise in the application of Ensemble Learning for combining and selecting various modelling approaches as well as in the initial selection of parameters applicable to the model.

A second, often overlooked, aspect of the model has to do with audit and acceptance:

- In order to be used by financial institutions the model has to be auditable by a third party. This would be relatively straightforward when using a single model but when various models are combined this is less trivial. So the final result, aside from having a valuation and it performance indicators should also include an auditable trace on how these values were constructed.
- Within the current architecture all changes on the data are available as a stream, this could trigger the updating of the valuation model. For this to work efficiently it is necessary to know the influence the changed datapoint has on the final valuations [**?**].

A.2. Deliverables/Requirements

- Valuation MdAPE < 10
- Valuation data available as API endpoint
- Models available as a Service within the company architecture
- Valuation service acts as a stream consumer on the existing data stream from the core database

B

Software used and links to documentation

- Apache Kafka: https://kafka.apache.org/documentation/
- Apache Spark: https://spark.apache.org/docs/latest/
- Apache Flink: https://ci.apache.org/projects/flink/flink-docs-release-1.3/
- Elasticsearch: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- Grafana: http://docs.grafana.org
- Postgres: https://www.postgresql.org/docs/9.6/static/index.html
- Scala: https://www.scala-lang.org/documentation/
- SBT: http://www.scala-sbt.org/documentation.html
- Docker: https://docs.docker.com

C

Process description

In this appendix, we will describe the process from our own perspective (as reflected in the writing style). Four parts of the process are examined: agile project management, coding and testing, code reviews, and the group process.

C.1. Agile

We adopted an agile approach to project management. This was extra helpful for this project, as the project definition changed several times in the pursuit of a relevant product that would fit the software engineering bill. We went from building a model for single family homes and sourcing the data ourselves to the system that is presented now in the first three weeks. As we learned more about Kafka and distributed systems, the design itself evolved with our knowledge and choice of the systems. No time was wasted, however, as the initial research and project planning phase still found its way into the final product.

In order to guide this process well, we created a project planning that detailed regular meetings with our TU Delft coach as well as company management, project milestones, and external deadlines. In these meetings, we presented our ideas for the project, technical designs, and implementations. After these meetings, we would come together as a group to discuss the steps necessary to develop the product and how we would incorporate the feedback from the meetings. These steps were entered into Trello and assigned to each member of the team with a one-week deadline. As we were working at GeoPhy's HQ in Delft, we could talk to our GeoPhy coach any time we needed feedback, and we made use of this opportunity several times.

When we entered a more code-focused period of the project, we switched from Trello boards to Git issues. This way, we could link merge requests and commits to specific issues and track the progress of the project. Towards the completion of the project, each member of the team worked on their part of the code and report and the workload was shared equally.

Since the project definition changed many times, it became necessary to set up a testing plan to define and measure the success of the project towards the end of the project. At the start of the project, this would have been useless, as the requirements would change over time. Towards the end, however, we needed to know whether the project could be deemed successful or not and this was achieved with the testing plan.

C.2. Coding and testing

Our project started in Python and moved to Scala as we found out that Kafka provides better support and documentation for Scala. This meant we switched from PyCharm to IntelliJ as an IDE and had to set up a new project from scratch. This change could be overcome, as the biggest work in the project was to understand the complex project material and requirements and how to fulfil these with Spark, Flink, and Kafka, not necessarily enormous amounts of code.

Tests were written as we went along. Usually, a merge request consisted of new functionality and fixes, together with tests written to test this functionality. Because of the nature of this project (large datasets, distributed frameworks), a lot of the testing before merge requests was done by way of running the project for ourselves and checking to see if it compiled and would run. Towards submission deadlines, we made extra efforts to write unit tests, so the integrity of the system could also be monitored automatically.

C.3. Code reviews

Critical code reviews were performed before a merge request was actually merged. No merge request has been merged without at least one comment and each merge request was reviewed by every member of the team. Because we worked together in the same room each day, we could easily discuss issues that came up in code review face-to-face. A lot of the maintainability, quality, and software design principles from our education were applied in these reviews. This resulted in maintainable, understandable code, with high quality standards. Builds were run in a continuous integration (CI) pipeline on Gitlab. Only passing builds were merged.

C.4. Group process

We worked together in the same building for most of the project. Because we knew each other from previous projects and since we worked so closely together for so many hours each day, we were able to co-operate very effectively. Group lunches, activities, and interactions with the rest of the company added to this positive working environment. Group morale was high and shared interest in the success of the project, not ego or pride, was clearly noticeable when discussing design or software decisions and reviewing each other's work.

D

Feedback from SIG

D.1. First submission

De code van het systeem scoort 4,5 ster op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'MultiFamilyModel.pipelineSetup'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld 'Create model' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

Glossary

Apache Flink

is an open-source stream processing framework.

Apache Kafka

is a distributed streaming platform responsible for the flow of data inside GeoPhy's architecture.

Apache Spark

is a fast and general engine for large-scale data processing, including machine learning, stream processing, and SQL.

API

is a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.

AVM

acronym for automated valuation model.

concept drift

is the occurrence that the statistical properties of a target variable changed, which the model is trying to predict.

CoreDB

is GeoPhy's main database with property data used by all its services.

ETL

is an acronym for extraction transformation loading.

MdAPE

is an acronym for median absolute percentage error.

RMSE

is an acronym for root mean squared error.

Bibliography

[1] Evgeny A. Antipov and Elena B. Pokryshevskaya. Mass appraisal of residential apartments: An application of random forest for valuation and a cart-based approach for model diagnostics. *Expert Systems with Applications*, 39(2):1772 – 1778, 2012. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2011.08.077. URL http://www.sciencedirect.com/science/article/pii/S0957417411011894.

[2] Iván Arribas, Fernando García, Francisco Guijarro, Javier Oliver, and Rima Tamošiūnienė. Mass appraisal of residential real estate using multilevel modelling. *International Journal of Strategic Property Management*, 20(1):77–87, 2016. doi: 10.3846/1648715X.2015.1134702. URL http://dx.doi.org/10.3846/1648715X.2015.1134702.

[3] A. Bellotti. Reliable region predictions for automated valuation models. *Annals of Mathematics and Artificial Intelligence*, pages 1–14, 2017. doi: 10.1007/s10472-016-9534-6. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85009887592&doi=10.1007%2fs10472-016-9534-6&partnerID=40&md5=5d1b189b33025a2af32cba376abaf12a.

[4] Earl D. Benson, Julia L. Hansen, Arthur L. Schwartz, and Greg T. Smersh. Pricing residential amenities: The value of a view. *The Journal of Real Estate Finance and Economics*, 16(1):55–73, 1998. ISSN 1573-045X. doi: 10.1023/A:1007785315925. URL http://dx.doi.org/10.1023/A:1007785315925.

[5] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM. ISBN 1-58113-195-X. doi: 10.1145/350391.350432. URL http://doi.acm.org/10.1145/350391.350432.

[6] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Data Engineering*, 38(4), 2015.

[7] International Valuation Standards Committee. Glossary, Jan 2014. URL https://www.ivsc.org/standards/glossary. Accessed: 2017-05-03.

[8] H. Crosby, P. Davis, T. Damoulas, and S.A. Jarvis. A spatio-temporal, gaussian process regression, real-estate price predictor. 2016. doi: 10.1145/2996913.2996960. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85011103556&doi=10.1145%2f2996913.2996960&partnerID=40&md5=915fd9ad4ec1f7b5471cc4ca66be1208.

[9] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012. ISSN 0001-0782. doi: 10.1145/2347736.2347755. URL http://doi.acm.org/10.1145/2347736.2347755.

[10] Y. El Hamzaoui and J.A.H. Perez. Application of artificial neural networks to predict the selling price in the real estate valuation process. pages 175–181, 2011. doi: 10.1109/MICAI.2011.14. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84856113492&doi=10.1109%2fMICAI.2011.14&partnerID=40&md5=ff2678bc3b01f279d35cc59aa8dcda97.

[11] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 2014. doi: 10.1145/2523813. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84901228061&doi=10.1145%2f2523813&partnerID=40&md5=d02bb1b645cabcda78b8ea4a25eff5de. cited By 200.

[12] M. Gorawski and A. Gorawska. Research on the stream etl process. *Communications in Computer and Information Science*, 424:61–71, 2014. doi: 10.1007/978-3-319-06932-6_7. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84903453888&doi=10.1007%2f978-3-319-06932-6_7&partnerID=40&md5=90d49564e731d13287db0b871171a407.

[13] M. Graczyk, T. Lasota, B. Trawiński, and K. Trawiński. Comparison of bagging, boosting and stacking ensembles applied to real estate appraisal. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5991 LNAI(PART 2):340–350, 2010. doi: 10.1007/978-3-642-12101-2_35. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-77956543024&doi=10.1007%2f978-3-642-12101-2_35&partnerID=40&md5=f35ee81db551245bcc37e5fefeda2c9e.

[14] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, 2013. USENIX. ISBN 978-1-931971-02-7. URL https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst.

[15] ISO/IEC 25023:2016. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality. Standard, International Organization for Standardization, Geneva, CH, June 2016.

[16] Vilius Kontrimas and Antanas Verikas. The mass appraisal of the real estate by computational intelligence. *Applied Soft Computing*, 11(1):443 – 448, 2011. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2009.12.003. URL http://www.sciencedirect.com/science/article/pii/S1568494609002579.

[17] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. 2011.

[18] J. Kroß, A. Brunnert, C. Prehofer, T.A. Runkler, and H. Krcmar. Stream processing on demand for lambda architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9272:243–257, 2015. doi: 10.1007/978-3-319-23267-6_16. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84944738642&doi=10.1007%2f978-3-319-23267-6_16&partnerID=40&md5=69188e20f2b4ea70b75fbc9ebbb7a624. cited By 3.

[19] T. Lasota, P. Sachnowski, and B. Trawiński. Comparative analysis of regression tree models for premises valuation using statistica data miner. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5796 LNAI:776–787, 2009. doi: 10.1007/978-3-642-04441-0-68. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-70450273517&doi=10.1007%2f978-3-642-04441-0-68&partnerID=40&md5=e231c3a2d8b9deea38fc3463871c33db.

[20] T. Lasota, Z. Telec, B. Trawiński, and K. Trawiński. Exploration of bagging ensembles comprising genetic fuzzy models to assist with real estate appraisals. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5788 LNCS:554–561, 2009. doi: 10.1007/978-3-642-04394-9_67. URL https://www.scopus.com/inward/

record.uri?eid=2-s2.0-76249108618&doi=10.1007%2f978-3-642-04394-9_67&partnerID=40&md5=f041cf0036ba26d49f082c80758b81aa.

[21] T. Lasota, B. Londzin, Z. Telec, and B. Trawiński. Comparison of ensemble approaches: Mixture of experts and ada boost for a regression problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8398 LNAI(PART 2):100–109, 2014. doi: 10.1007/978-3-319-05458-2_11. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84958523774&doi=10.1007%2f978-3-319-05458-2_11&partnerID=40&md5=2ced2040647e794e68c53bd98dcf6142.

[22] Gideon Lewis-Kraus. The great ai awakening. *The New York Times Magazine*, 2016.

[23] Roy E. Lowrance, Yann Lecun, Dennis Shasha, and Roy E. Lowrance. Predicting the market value of single-family residential real estate, 2015.

[24] Stephen W.K. Mak, Lennon H.T. Choy, and Winky K.O. Ho. Hedonic models, internet□based technologies, and the provision of online property appraisal. *Construction Innovation*, 8(2):92–105, 2008. doi: 10.1108/14714170810867023. URL http://dx.doi.org/10.1108/14714170810867023.

[25] S. Marotta, M. Metzger, J. Gorman, and A. Sliva. Modular analytics management architecture for interoperability and decision support. volume 9831, 2016. doi: 10.1117/12.2224142. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84987910541&doi=10.1117%2f12.2224142&partnerID=40&md5=e8ac02eeff33de731a1e602af114165a.

[26] W.J. McCluskey, M. McCord, P.T. Davis, M. Haran, and D. McIlhatton. Prediction accuracy in mass appraisal: a comparison of modern approaches. *Journal of Property Research*, 30(4):239–265, 2013. doi: 10.1080/09599916.2013.781204. URL http://dx.doi.org/10.1080/09599916.2013.781204.

[27] Brent Daniel Mittelstadt, Patrick Allo, Mariarosaria Taddeo, Sandra Wachter, and Luciano Floridi. The ethics of algorithms: Mapping the debate. *Big Data & Society*, 3(2): 2053951716679679, 2016.

[28] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[29] The University of Waikato. Weka 3 - data mining software in java, 2017. URL http://www.cs.waikato.ac.nz/ml/weka/. Accessed: 2017-05-08.

[30] E. Pagourtzi, V. Assimakopoulos, T. Hatzichristos, and N. French. Real estate appraisal: A review of valuation methods. *Journal of Property Investment & Finance*, 21(4):383–401, 2003. doi: 10.1108/14635780310483656. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84986145250&doi=10.1108%2f14635780310483656&partnerID=40&md5=99f99b75216d410179c82300a7ded1b0.

[31] B. Park and J. Kwon Bae. Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data. *Expert Systems with Applications*, 42(6):2928–2934, 2015. doi: 10.1016/j.eswa.2014.11.040. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84949116609&doi=10.1016%2fj.eswa.2014.11.040&partnerID=40&md5=e4a47969f35a2ea2b16197762a60ca26.

[32] M.H. Rafiei and H. Adeli. A novel machine learning model for estimation of sale prices of real estate units. *Journal of Construction Engineering and Management*, 142(2), 2016. doi: 10.1061/(ASCE)CO.1943-7862.0001047. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954548028&doi=10.1061%2f%28ASCE%29CO.1943-7862.0001047&partnerID=40&md5=67827ce287829229a65806cff3603968.

[33] M. Stoicescu, J.-C. Fabre, and M. Roy. Architecting resilient computing systems: A component-based approach for adaptive fault tolerance. *Journal of Systems Architecture*, 73:6–16, 2017. doi: 10.1016/j.sysarc.2016.12.005. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85009811529&doi=10.1016%2fj.sysarc.2016.12.005&partnerID=40&md5=bebecc3a108d79a7040a5e5dce9ac8ce.

[34] S.R. Sukumar, R. Kannan, S.-H. Lim, and M.A. Matheson. Kernels for scalable data analysis in science: Towards an architecture-portable future. pages 1026–1031, 2017. doi: 10.1109/BigData.2016.7840703. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015161333&doi=10.1109%2fBigData.2016.7840703&partnerID=40&md5=63b953a209c32e45d560e4ba69a51049.

[35] S.R. Sukumar, M.A. Matheson, R. Kannan, and S.-H. Lim. Mini-apps for high performance data analysis. pages 1483–1492, 2017. doi: 10.1109/BigData.2016.7840756. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015231353&doi=10.1109%2fBigData.2016.7840756&partnerID=40&md5=7b6ebb9d6819db8a60b376251a8c1595.

[36] V. Theodorou, A. Abelló, W. Lehner, and M. Thiele. Quality measures for etl processes: from goals to implementation. *Concurrency Computation*, 28(15):3969–3993, 2016. doi: 10.1002/cpe.3729. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84951309892&doi=10.1002%2fcpe.3729&partnerID=40&md5=7815fa12f19acf08bd0d4c307493db14.

[37] C. Watkins. Property valuation and the structure of urban housing markets. *Journal of Property Investment & Finance*, 17(2):157–175, 1999. doi: 10.1108/14635789910258543. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84986180783&doi=10.1108%2f14635789910258543&partnerID=40&md5=eaac8dee0d24d532c2c5757db3145691.

[38] Gerhard Widmer and Miroslav Kubat. *Effective learning in dynamic environments by explicit context tracking*, pages 227–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993. ISBN 978-3-540-47597-2. doi: 10.1007/3-540-56602-3_139. URL http://dx.doi.org/10.1007/3-540-56602-3_139.

[39] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[40] Indre Zliobaite. Learning under concept drift: an overview. *CoRR*, abs/1010.4784, 2010. URL http://arxiv.org/abs/1010.4784.

[41] Jozef Zurada, Alan S. Levitan, and Jian Guan. A Comparison of Regression and Artificial Intelligence Methods in a Mass Appraisal Context. *Journal of Real Estate Research*, 33(3):349–388, 2011. URL https://ideas.repec.org/a/jre/issued/v33n32011p349-388.html.

BEP Infosheet

Automating Valuations for Real-Estate

Problem definition GeoPhy needs a system that can run automated valuation models (AVM) in production (AVM Service), which works in GeoPhy's proposed architecture, consuming enormous streams of data, and trains its models based on changes in the market.

Project Highlights

- Automated data flow from streaming platform (Apache Kafka)
- Automated training and loading of models (Apache Spark, Flink)
- API access to valuation predictions
- Concept drift detection for efficient retraining
- Highly scalable and containerised for cloud deployment

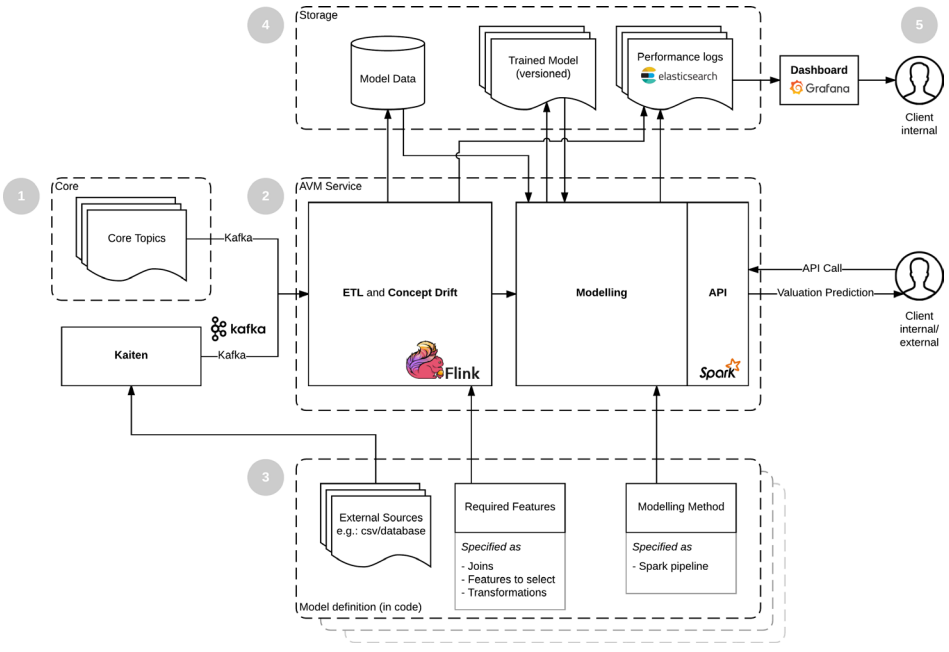
Alex Geenen
a.t.geenen@student.tudelft.nl

Hung Nguyen
h.nguyen-3@student.tudelft.nl

Ruben Wiersma
r.t.wiersma@student.tudelft.nl

System Design

1. Uniform Kafka data input
2. AVM Service
3. Extensible model definitions
4. Storage and logging
5. Client access for monitoring and prediction retrieval



GEOPHY
Real Estate meets Big Data

TU Delft