Bachelorarbeit

Developing an Autonomous Trading System: A Case Study on AI Engineering Practices

Marcel Grote

Course of Study: Softwaretechnik
Examiner: Prof. Dr. Stefan Wagner
Supervisor: Dr. Justus Bogner

Commenced: March 30, 2022
Completed: September 30, 2022
Abstract

Today, more and more systems are using AI to efficiently solve complex problems. While in many cases this increases the system’s performance and efficiency, developing such systems with AI functionality is a more difficult process due to the additional complexity. Thus, engineering practices are required to ensure the quality of the resulting software. Since the development of AI-based systems comes with new challenges, new engineering practices are needed for such development processes. Many practices have already been proposed for the development of AI-based systems. However, only a few practical experiences have been accumulated in applying these practices. This study aims to address this problem by collecting such experiences. Furthermore, our objective is to accumulate evidence of the effectiveness of these proposed practices. Additionally, we analyze challenges that occur during such a development process and provide solutions to overcome them. Lastly, we examine the tools proposed to develop AI-based systems. We aim to identify how helpful these tools are and how they affect the resulting system.

We conducted a single case study in which we developed an autonomous stock trading system that uses machine learning functionality to invest in stocks. Before development, we conducted literature surveys to identify effective practices and useful tools for such an AI development process. During the development, we applied ten practices and seven tools. Using structured field notes, we documented the effects of these practices and tools. Furthermore, we used field notes to document challenges that occurred during the development and the solutions we applied to overcome them. After the development, we analyzed the collected field notes. We evaluated how the application of each practice and tool simplified the development and how it affected the software quality. Moreover, the experiences collected in applying these proposed practices and tools and the challenges encountered are compared with existing literature. Our experiences and the evidence we collected during this study can be used as advice to simplify the development of AI-based systems and to improve software quality.
1 Introduction

This introductory chapter explains the context of this research field. This is followed by a description of the study’s research objectives. Subsequently, we describe our contributions and, lastly, we explain the outline of this thesis.

1.1 Context

Today, more and more critical software systems are based on AI, such as autonomous cars, power grid management software, or autonomous stock trading systems [1]. While this enables functionality that was previously impossible, it also comes with additional complexity and new engineering challenges [2]. Examples are efficiently managing large amounts of data [3], ensuring system safety and reliability [4][5], or choosing the right architecture [6]. Because the failure of such a critical system can have serious negative consequences [7], it is important to ensure the quality of the developed system. Therefore, especially the development of critical systems requires guidelines [8] and best practices [9][10][11]. A vast amount of software engineering work has been done on such guidelines and practices for general software development, but compared to this, only few work has been done regarding AI engineering practices. Although there are a variety of AI engineering papers on challenges, practices, and guidelines, very little work has been done to gather practical experiences when applying proposed practices. This study aims to fill this gap.

1.2 Research Objective

The first objective of this study is to collect experiences and evidence for the effectiveness of proposed AI engineering practices by applying them during the development of an AI-based system. Furthermore, we aim to synthesize generalizable insights on experienced challenges and solutions to overcome these challenges. Lastly, this study aims to validate the usefulness of the proposed tools for developing AI-based systems. We aim to identify how these tools simplify the development process and how they affect the resulting software.

1.3 Contribution

The use of software engineering practices has increased dramatically in the last 40 years [12][13]. Today, companies are making more and more efforts to incorporate practices into the development process. However, there are also AI engineering practices that are rarely adopted [11]. We assume that this is due to the fact that there is only little literature on these practices. This study shows how effective such AI engineering practices can be. Therefore, we hope that our work motivates
development teams to adopt more of these practices. Another contribution of this study is the analysis of ten proposed AI engineering practices and the validation of the effectiveness of each practice. We show that applying each of these ten practices can improve the development process and should thereby be considered in any machine learning development process. Furthermore, the experiences collected during this development process can be helpful advice to apply these practices more effectively. In addition, we provide detailed information on challenges that can occur during such a development process and solutions to overcome them. For each challenge, we reviewed existing literature and evaluated how we addressed these challenges. Lastly, we analyzed seven proposed tools for AI development. We evaluated how helpful these were during the development and identified limitations for several of these tools.

1.4 Outline

This thesis starts by explaining the relevant fundamentals of this study. The related work chapter describes already existing studies on AI practices, tools, and challenges. Chapter four explains the study design. The results are listed in the fifth chapter, followed by the conclusion in chapter 6.
2 Fundamentals

In this chapter, we explain the fundamentals relevant to this study on artificial intelligence, AI engineering, autonomous systems, and the stock market.

2.1 Artificial Intelligence

Artificial intelligence is commonly defined as “a technology that enables a machine to simulate human behavior” [14]. Machine learning is seen as a subset of this [15]. It is defined as “the capacity of systems to learn from problem-specific training data to automate the process of analytical model building and solve associated tasks” [16]. Machine learning uses mathematical models of data to enable a machine to learn on its own [17].

2.2 AI Engineering

Many software systems fail every so often. To minimize this risk and ensure that the system fulfills all requirements, it is widely common to follow so-called software engineering practices. In 1968, Fritz Bauer described the term Software Engineering as “the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines” [18]. Since then, the field of software engineering has evolved massively [19]. The increasing number of AI-based systems introduced a significant change. The development of such systems differs from the development of traditional software [20]. Microsoft researchers identified three fundamental differences: 1) data discovery, management, and versioning are more complex, 2) development requires a wider set of skills, and 3) achieving a modular design is more difficult, since AI components can be entangled in complex ways [10]. These differences created the need for new software engineering practices for the development, maintenance, and evolution of AI-based systems [21]. As the use of artificial intelligence continued to grow [22], concerns about safety, security, and ethics increased.

As a result, the US Office of the Director of National Intelligence (ODNI) started funding a national initiative to advance the discipline of artificial intelligence engineering for defense and national security at the Carnegie Mellon University Software Engineering Institute (SEI) [23]. SEI has defined AI Engineering as “a field of research and practice that combines the principles of systems engineering, software engineering, computer science, and human-centered design to create AI systems in accordance with human needs for mission outcomes” [24].

Additionally, SEI has introduced the three pillars of AI Engineering[25]:

- **Human-Centered AI** systems are designed focusing on the human perspective [26]
- **Scalable AI** systems have the ability to operate at the required size, speed, and complexity [27]
• **Robust and Secure AI** systems operate reliably at the expected performance level even under stress [28]

### 2.3 Autonomous Systems

With the rise of applications such as industry 4.0 and the Internet of Things, autonomous systems become increasingly important [29]. By integrating machine learning functionality, these autonomous systems become even more powerful. Additionally, autonomous systems are capable of performing unsupervised operations and making decisions without human intervention [30][31]. However, this lack of human interaction creates additional challenges for the development of such systems, for example, assuring the system’s safety and ethical concerns.

### 2.4 The Stock Market

There are 60 major stock exchanges around the world [32]. On a stock exchange, different financial instruments can be traded, such as stocks, bonds, and commodities [33]. A stock is a security that represents a fraction of the ownership of the issuing corporation [34]. Stock trades are done mainly during the opening hours of the stock market. Some trades can also be done outside of the opening hours. These trades are called after-hours trades. Those used to be limited to institutional and high-net-worth investors, but today many trading platforms offer after-hours trades even for the average investor [35].

Since owning a stock means that the investor owns a fraction of the corresponding corporation, the investor is entitled to a fraction of the corporation’s earnings, the dividend. The corporation’s board of directors determines how frequently and how much dividend is paid [36].

The term penny stock refers to a special category of stocks. These are stocks whose share prices trade below 5 US dollars [37]. Investing in such small stocks can be extremely profitable, but this comes with an enormous risk [38].
2.4 The Stock Market

Stock courses are commonly visualized using candlestick charts. Such a chart consists of a sequence of candlesticks. Each candlestick visualizes the Open, High, Low, and Close (OHLC) values of the corresponding time interval.

The Open value refers to the first price in that time interval, the Close value refers to the last price in the time interval, the High value refers to the maximum price in the time interval, and the Low value refers to the minimum price in the time interval. Common time intervals are one-minute, 15-minute, hourly, daily, and weekly intervals. If the stock decreased its price during that time interval, meaning that the Close value is less than the Open value, the candlestick is depicted in red; otherwise, it is shown in green. Furthermore, the volume of a stock describes the number of shares that were traded during the corresponding time interval.

There are many different investment strategies. They can be broken down according to how long the investor intends to hold the investment before selling it. In this study, we mainly consider holding
the investment for less than a day; this is described as intraday trading. In some cases, we also consider holding the investment for up to one week; such investments are referred to as short-term investments.
3 Related Work

Various research on AI engineering has already been done. By conducting a literature survey, Serban et al. [11] identified 29 engineering practices to develop machine learning systems. Furthermore, they conducted a survey with 313 practitioners to identify the degree of adoption of these practices. They concluded that some practices should receive more attention, while some other practices should receive less. Akkiraju et al. [39] present 33 best practices from the authors’ personal experience for different stages of the life cycle of an AI model. They claim that these practices help organizations achieve a higher level of maturity. Zhang et al. [40] conducted a survey on techniques to test machine learning systems. Unlike the previous two articles, this survey only covers practices for testing machine learning models. In addition, Zhang et al. summarized the proposed testing tools. On the same topic, Landset et al. [41] conducted a survey of open source tools to simplify the learning task of such systems. They concluded that the choice of tools should depend on the application that is being built. Furthermore, Ashiku et al. [42] have conducted a case study in which they used tools to develop machine learning models with the aim of evaluating the acceptance of transplanted kidneys. They analyzed these tools mainly on the basis of how much they simplified the development of the corresponding model. Although we analyze certain tools in this paper, our objective is to analyze them in a broader approach to also identify how the use of these tools affects the software quality of the overall system.

Additionally, several studies have examined challenges during the development of machine learning systems. Lwakatare et al. [3] conducted a case study to identify the challenges companies face during the development of machine learning components. They concluded that a lot of effort is needed to manage those challenges that they identified as the most important challenges. Nascimento et al. [43] conducted interviews with developers from three companies that develop machine learning systems to identify the main challenges and propose solutions to overcome these challenges. Furthermore, L’Heureux et al. [44] summarize data challenges for machine learning algorithms. In addition, they present approaches to overcome these challenges. They state that since data sets grow larger, addressing these challenges becomes increasingly important.

In summary, although there are many articles on practices, challenges, and tools, only little work has been done to collect practical experiences. Existing work mostly focuses either exclusively on practices or challenges or tools and in many cases analyzes those only in a specific context, such as testing machine learning models. Additionally, most of these articles focus mainly on how much these practices, tools, or solutions to challenges simplify the development of the corresponding machine learning model. They often ignore the resulting effect on the software quality.
4 Case Study Design

This chapter explains the design of the study. We first list the research questions we aim to answer. Subsequently, we explain the case analyzed in this study. Furthermore, we describe which practices and tools were selected. Lastly, we explain how the data during this study was collected and analyzed.

4.1 Research Questions

A significant number of practices have been proposed. Since it is impossible to follow all of them, it is necessary to collect experiences on how effective these practices are. During this study, we apply some of these proposed practices and aim to analyze the effectiveness of each practice.

RQ1: How effective were the proposed practices in respect to developing an AI system?

Software development processes are generally filled with challenges, but developing software with machine learning components comes with a variety of new challenges. We aim to identify and analyze the challenges that occur during this study.

RQ2: What challenges did occur during the case study?

After these AI-specific challenges occur, we attempt to overcome them in the best possible way. The solutions chosen for these challenges are analyzed and compared to possible alternatives. We aim to identify the most suitable solutions in terms of the time needed to perform the solution and the effect on the software quality (e.g., loss of performance or interpretability).

RQ3: How were the challenges addressed?

Finally, as more and more tools for machine learning development processes arise, we analyze a set of tools to identify how much these simplify the development process and how they affect the software quality.

RQ4: What tools were helpful during the development of an AI system?

4.2 Case Description

To answer these questions, we conducted a single case study. In this case study, an example AI system was developed. The system was designed to contain machine learning functionality to predict upward movements in stock prices. We selected this development process as the analyzed case in this study, because there already are several papers on feasible machine learning models for this use case [45][46][47] and developing such a system appeared to be challenging, making it ideal in terms of collecting as many insights as possible.
The development process took three months. It started with the requirements analysis, followed by the design stage and the development. The development was planned to begin with the development of the overall system and end with the development and tuning of the machine learning component. After the machine learning component was finished, it was integrated into the overall system during the integration stage. Finally, the system was deployed.
Figure 4.2: Business Process Model and Notation diagram of the system’s workflow

Figure 4.2 shows a BPMN diagram of the system’s workflow according to our first design. The general objective of this system is to make a profit through its investments. To achieve this, we designed the system to choose stocks to observe every morning before the stock market opens, then as soon as the stock market opens, frequently retrieve the prices of these stocks in one-minute intervals, and invest based on the predictions made by the machine learning components. We
decided that the system should invest at the Nasdaq\textsuperscript{1} stock market since this is the largest global electronic market for trading stocks [48]. Before the beginning of the development process, the literature was surveyed for suitable best practices, which were then applied during the development. Any encountered challenges and the use and effects of each practice and tool were documented and analyzed afterwards.

4.3 Selection of Best Practices

4.3.1 Literature Survey

A literature survey identified two articles that both propose a large set of best practices that affect the entire development process of a machine learning system, starting with the requirements analysis and ending with the deployment and maintenance. Serban et al. [11] accumulated 29 best practices and grouped them into 6 categories: Data, Training, Deployment, Coding, Team, and Governance. Furthermore, Akkiraju et al. [39] present 33 best practices. We grouped these practices into the same categories as Serban et al. In Table 4.1 we list all 62 practices from both articles.

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Title</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use Sanity Checks for All External Data Sources</td>
<td>[11]</td>
</tr>
<tr>
<td>2</td>
<td>Check that Input Data is Complete, Balanced and Well Distributed</td>
<td>[11]</td>
</tr>
<tr>
<td>3</td>
<td>Write Reusable Scripts for Data Cleaning and Merging</td>
<td>[11]</td>
</tr>
<tr>
<td>4</td>
<td>Ensure Data Labelling is Performed in a Strictly Controlled Process</td>
<td>[11]</td>
</tr>
<tr>
<td>5</td>
<td>Make Data Sets Available on Shared Infrastructure (private or public)</td>
<td>[11]</td>
</tr>
<tr>
<td>6</td>
<td>Define Data Requirements According to Business Needs</td>
<td>[39]</td>
</tr>
<tr>
<td>7</td>
<td>Define a Data Acquisition Strategy</td>
<td>[39]</td>
</tr>
<tr>
<td>8</td>
<td>Apply Data Selection to Select Suitable Training Data</td>
<td>[39]</td>
</tr>
<tr>
<td>9</td>
<td>Create Data Annotation Guidelines to Achieve Consistency with Data Labeling</td>
<td>[39]</td>
</tr>
<tr>
<td>10</td>
<td>Augment Data using Synthetic Techniques as Applicable</td>
<td>[39]</td>
</tr>
<tr>
<td>11</td>
<td>Standardize and Automate Quality Check Procedure</td>
<td>[39]</td>
</tr>
<tr>
<td>12</td>
<td>Maintain Data Lineage</td>
<td>[39]</td>
</tr>
<tr>
<td>13</td>
<td>Govern Data</td>
<td>[39]</td>
</tr>
<tr>
<td>14</td>
<td>Curate Training Data for Specific Business Needs and Continuously Learning</td>
<td>[39]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Title</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Share a Clearly Defined Training Objective within the Team</td>
<td>[11]</td>
</tr>
<tr>
<td>16</td>
<td>Capture the Training Objective in a Metric that is Easy to Measure and Understand</td>
<td>[11]</td>
</tr>
<tr>
<td>17</td>
<td>Test all Feature Extraction Code</td>
<td>[11]</td>
</tr>
<tr>
<td>18</td>
<td>Assign an Owner to Each Feature and Document its Rationale</td>
<td>[11]</td>
</tr>
<tr>
<td>19</td>
<td>Actively Remove or Archive Features That are Not Used</td>
<td>[11]</td>
</tr>
</tbody>
</table>

\textsuperscript{1}https://www.nasdaq.com

18
4.3 Selection of Best Practices

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Peer Review Training Scripts</td>
</tr>
<tr>
<td>21</td>
<td>Enable Parallel Training Scripts</td>
</tr>
<tr>
<td>22</td>
<td>Automate Hyper-Parameter Optimisation and Model Selection</td>
</tr>
<tr>
<td>23</td>
<td>Continuously Measure Model Quality and Performance</td>
</tr>
<tr>
<td>24</td>
<td>Share Status and Outcomes of Experiments Within the Team</td>
</tr>
<tr>
<td>25</td>
<td>Use Versioning for Data, Model, Configurations and Training Scripts</td>
</tr>
<tr>
<td>26</td>
<td>Use Cross-Validation</td>
</tr>
<tr>
<td>27</td>
<td>Keep Your Options Open during Feature Selection</td>
</tr>
<tr>
<td>28</td>
<td>Understand Performance Tradeoffs with Feature Processing</td>
</tr>
<tr>
<td>29</td>
<td>Master the Art of Feature Representation</td>
</tr>
<tr>
<td>30</td>
<td>From Experimentation to Production: Design your Compute Strategy</td>
</tr>
<tr>
<td>31</td>
<td>Data and Model Versioning for Efficient Collaboration and Experimentation</td>
</tr>
<tr>
<td>32</td>
<td>Modularizing Train Code</td>
</tr>
<tr>
<td>33</td>
<td>Plan for Train to Serve Handoff Management</td>
</tr>
<tr>
<td>34</td>
<td>AI Models are Rarely Perfect on Day-One. Plan for Continuous Improvements</td>
</tr>
<tr>
<td>35</td>
<td>Automate the Train Pipeline</td>
</tr>
<tr>
<td>36</td>
<td>Be Prepared to Iterate between Train and Test</td>
</tr>
<tr>
<td>37</td>
<td>You can Test All You Want but Real-world Can Still Shock You! Don’t Judge a Machine Learning Model in First Iteration</td>
</tr>
<tr>
<td>38</td>
<td>Testing is Not Just a One-time Build Activity. It is Continuous Throughout an AI model’s life cycle. Keep the Test Data Sets Updated</td>
</tr>
<tr>
<td>39</td>
<td>Whose Side Is the Real 'Truth'? Sometimes Machine Learning Models Are Both Right and Wrong!</td>
</tr>
<tr>
<td>40</td>
<td>Scenario Testing for Applications Consisting of Multiple Machine Learning Models</td>
</tr>
<tr>
<td>41</td>
<td>Adversarial and Long Tail Testing</td>
</tr>
<tr>
<td>42</td>
<td>Set Proper Goals for AI Models to Mitigate Undesirable Biases and Start with Test Cases</td>
</tr>
<tr>
<td>43</td>
<td>Declare Your Biases to Establish Trust</td>
</tr>
<tr>
<td>44</td>
<td>Do We Always Need Full Explainability? Let the use case drive the needs and select machine learning algorithms accordingly</td>
</tr>
<tr>
<td>45</td>
<td>Diagnose Errors at Scale</td>
</tr>
<tr>
<td>46</td>
<td>Error Validation and Categorization</td>
</tr>
<tr>
<td>47</td>
<td>Version Models and Manage their Lineage to Better Understand Model Behavior Over Time</td>
</tr>
</tbody>
</table>

**Code**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>Run Automated Regression Tests</td>
</tr>
<tr>
<td>49</td>
<td>Use Continuous Integration</td>
</tr>
<tr>
<td>50</td>
<td>Use Static Analysis to Check Code Quality</td>
</tr>
</tbody>
</table>

**Deployment**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>Automate Model Deployment</td>
</tr>
<tr>
<td>52</td>
<td>Continuously Monitor the Behaviour of Deployed Models</td>
</tr>
</tbody>
</table>
4 Case Study Design

<table>
<thead>
<tr>
<th></th>
<th>Proposed Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>Enable Shadow Deployment</td>
</tr>
<tr>
<td>54</td>
<td>Perform Checks to Detect Skews between Models</td>
</tr>
<tr>
<td>55</td>
<td>Enable Automatic Roll Backs for Production Models</td>
</tr>
<tr>
<td>56</td>
<td>Log Production Predictions with the Model’s Version and Input Data</td>
</tr>
<tr>
<td>57</td>
<td>Continuously Monitor Models for Misalignments</td>
</tr>
<tr>
<td>58</td>
<td>Continuously Monitor Models for Drift</td>
</tr>
<tr>
<td>59</td>
<td>Use A Collaborative Development Platform</td>
</tr>
<tr>
<td>60</td>
<td>Work Against a Shared Backlog</td>
</tr>
<tr>
<td>61</td>
<td>Communicate, Align, and Collaborate with Multidisciplinary Team Members</td>
</tr>
<tr>
<td>62</td>
<td>Enforce Fairness and Privacy</td>
</tr>
</tbody>
</table>

**Table 4.1: Proposed Practices**

### 4.3.2 Selected Best Practices

Of the 62 proposed practices in Table 4.1 we selected 10 practices that we expect to improve the development process the most.

1. **Standardize and automate quality check procedure** (Practice 11) to ensure that only certified data is used for training or testing [39].
2. **Error validation and categorization** (Practice 46) to provide insights on when and why the machine learning model fails, so that the reliability can be improved [39].
3. **Capture the training objective in a metric that is easy to measure and understand** (Practice 16) to avoid entangled measurements and thereby increase interpretability [11].
4. **Use cross-validation** (Practice 26) to avoid testing a machine learning component on data that it has already seen [39].
5. **Continuously measure model quality, performance and drift** (Practice 23 & 58) to detect and fix errors early [11][39].
6. **Review training scripts** (Practice 20) to ensure the quality of such an important script [11]. The original practice proposes conducting peer reviews, but since most of the development is carried out by one developer, we generalize this practice to allow us to additionally evaluate the reviews conducted solely by this single developer.
7. **Test all feature extraction code** (Practice 17) to ensure that the generated data is consistent and does not introduce bugs [11].
8. **Automate hyper-parameter optimization and model selection** (Practice 22) to save time and increase model quality [11].
9. **Log predictions with the model’s version and input data** (Practice 56) to provide insights on how the model can be improved [11]. The original practice only proposed logging predictions when the system is deployed, but because we expect this practice to also be useful before deployment, we generalize this practice.

10. **Collaborate with multidisciplinary Stakeholders** (Practice 61) to simplify the development and improve the resulting software by obtaining domain-specific know-how [11]. The original practice only considers collaborating with team members, but we generalize this practice to additionally include any relevant stakeholder.

### 4.4 Selection of Tools

We conducted another literature survey to identify proposed tools for developing systems with machine learning functionality. Since we decided to use Python to build this system, we surveyed white and gray literature specifically for Python tools. For these proposed tools, we then considered how helpful we expect them to be during this development process. Since most of the proposed tools were for specific use cases, such as developing neural networks or computer vision, we decided to use only three of these proposed tools: scikit-learn\(^2\) [49][50][51][52] to simplify model development, Pandas\(^3\) [49][50][51][52][53] to achieve higher performance when handling data and Matplotlib\(^4\) [49][50][51] to obtain more insights on how to improve the model by visualizing data. Additionally, we used Git\(^5\) since our personal experience has shown that such a version control tool is immensely helpful for such a development process. Furthermore, when we started developing the machine learning component, we realized that we needed a way to store the trained scikit-learn models. A short gray literature survey revealed that this can be easily achieved using pickle\(^6\) [54]. Therefore, we decided to also use pickle to store our scikit-learn models. In addition, the machine learning engineer who took part in two meetings suggested during the first meeting using Optunity\(^7\) to automate the hyper-parameter optimization, and during the second meeting after reviewing our machine learning component he additionally suggested using NumPy\(^8\) to improve the model’s performance. We followed his advice and applied both tools during the development.

\(^2\)https://scikit-learn.org
\(^3\)https://pandas.pydata.org
\(^4\)https://matplotlib.org
\(^5\)https://git-scm.com
\(^6\)https://docs.python.org/3/library/pickle.html
\(^7\)https://optunity.readthedocs.io
\(^8\)https://numpy.org
4.5 Data Collection

Before the development, we considered for each practice at what moments during the development this practice might be effective. We summarized the results in Table 4.2.

<table>
<thead>
<tr>
<th>Practice</th>
<th>Effective Moments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardize and automate quality check procedure</td>
<td>When retrieving data</td>
</tr>
<tr>
<td>Error validation and categorization</td>
<td>When a model fails</td>
</tr>
<tr>
<td>Capture the training objective in a metric that is easy to measure and understand</td>
<td>Before developing a model</td>
</tr>
<tr>
<td>Use cross-validation</td>
<td>Testing a model</td>
</tr>
<tr>
<td>Continuously measure model quality, performance and drift</td>
<td>After a model is trained and after deployment, additionally on a regular basis</td>
</tr>
<tr>
<td>Review training scripts</td>
<td>After the first version of code is finished and after significant code changes have been made</td>
</tr>
<tr>
<td>Test all feature extraction code</td>
<td>After the first version of code is finished and after significant code changes have been made</td>
</tr>
<tr>
<td>Automate hyper-parameter optimization and model selection</td>
<td>During the hyper-parameter optimization and model selection process</td>
</tr>
<tr>
<td>Log predictions with the model’s version and input data</td>
<td>When aiming to improve the model and after deployment</td>
</tr>
<tr>
<td>Collaborate with multidisciplinary Stakeholders</td>
<td>When certain milestones are achieved such as finishing the design or the system is ready to be deployed</td>
</tr>
</tbody>
</table>

Table 4.2: Effective moments for each practice

When we reached one of these points, we then applied the corresponding practice. Each time a practice was followed, we documented how the practice was applied and the perceived effect using structured field notes. Each field note contained the title of the practice that was applied, a description of how the practice was applied, the perceived effect, the corresponding date, and a ranked score on an ordinal scale (−, −, 0, +, ++) to describe how effective we perceived applying this practice was. In some cases where we were able to easily measure the effect of applying a practice, for example, an improvement of the model’s precision or performance, we additionally documented such measured results. For any performance measurements, we used the Intel Core i7-8550U with four physical cores with a clock rate of 1.8 GHz, a 16 GB RAM, Windows 10 21H1 64-bit as operating system and the Python 3.10 interpreter.

In addition, we used the same field notes to document the use and effect of tools during the development. In total, 37 field notes were accumulated, 28 on the application of practices and 9 on the use of tools. Furthermore, we used field notes to document any challenges that occurred. These field notes contained a description of the challenge, the corresponding date on which the challenge
4.6 Data Analysis

occurred, the source that made us aware of the challenge, such as a stakeholder, for example, and a description of the solution we applied to overcome this challenge along with how effective this solution is. As soon as we realized that a challenge had occurred, we immediately created a new field note for this challenge. Finally, after overcoming a challenge, we completed the corresponding field note by adding the applied solution. In total, we collected 34 field notes on challenges during this development process, which we grouped into 11 AI-specific challenges.

4.6 Data Analysis

To answer RQ1 and RQ4, we interpret the corresponding field notes regarding how much applying each practice or tool has simplified the development process and how much it affected the software quality attributes of the resulting system, such as readability, accuracy, or performance. In addition, we compare our results with existing literature on these tools and practices. To answer RQ2, we analyze the augmented field notes on the challenges encountered and compare them with existing literature on challenges during such a machine learning development process. To answer RQ3, for each challenge we analyze the documented solution we applied and evaluate them with respect to how easy it was to apply each solution, how effective it solved the corresponding challenge, and the effects on software quality. Furthermore, in some cases, we compare the applied solution with alternatives that were also considered. Additionally, we compare our solutions with existing literature.
5 Results

In this chapter, we present the results of our study. First, we present how effective the applied practices were during the development process. For each practice, we describe how it was applied and subsequently analyze the effects of its application. To answer the second and third research questions, we then list all challenges that we encountered during the development. For each challenge, we provide a description and explain how we addressed it. Afterwards, we compare the challenges encountered during the development and the solutions applied to overcome them with existing literature. Lastly, we present how helpful the applied tools were. For each tool, we describe how it was used and subsequently analyze how it affected the development process and the resulting software.

5.1 Effectiveness of AI Engineering Practices (RQ1)

5.1.1 Results of Field Notes

Standardize and Automate Quality Check Procedure

**Application** This practice was applied twice. The first time was after receiving intraday stock data using a Python script. The script was required to wait 12 seconds after every API request to avoid violating the API limits. Since the data for each stock was split into 24 files, there was great concern that some files might be lost. As a result, we extended the script so that it automatically checks if any files are missing. If files were missing, the script would log a corresponding error message and try to retrieve that missing file again. On average, one file contained 6,911 lines of data. However, there also existed some files for which most of the data was missing, for example, files that consisted only of 10 lines. To avoid training the machine learning component with this erroneous data, we extended the script again to additionally count the lines for each file. If a file contained less than 100 lines, the script would automatically delete it.

Later in the development process, we additionally obtained daily stock data. Each file contained the OHLC and volume data in daily intervals. To ensure that for each file no days were missing, we implemented another quality check procedure. Since on weekends and some holidays the stock market is closed and thus no data exists for these days, identifying whether a gap is erroneous was a challenging task. The procedure we implemented compares the gaps contained in each file with the gaps contained in all other files. This procedure revealed that of the 716 stocks, the data for 17 contained gaps. In total, 137 days were missing. Since for many of these stocks only one or two days were missing, we decided to still use these as training data and only ignore those stocks for which more than two days of data were missing. The biggest gap we discovered was over a period of 28 days.
Analysis  Since we received more than 9,000 files, the use of automated procedures was critical to identify files that are likely to have a negative effect on the machine learning component. In our case, applying such procedures was effective as it helped us detect a large amount of gaps in the intraday and daily data. However, it is extremely difficult and time-consuming to ensure that the data is in perfect condition as it would have required us to retrieve the same data from at least one other source to allow us to compare it with the already obtained data. Nonetheless, without the use of automated quality procedures, we would have only been able to detect a fraction of the existing data quality issues. Due to the fact that even the best machine learning algorithms perform poorly when trained with bad data [55], fixing these data quality issues was critical to improve the model’s precision and thereby increasing the system’s effectiveness. As a result, applying this practice in our case improved the development process.

Error Validation and Categorization

Application  As proposed by Akkiraju et al. [39] we used a confusion matrix to identify training errors. Such a confusion matrix depicts the number of correct and incorrect predictions.

![Confusion Matrix](image)

**Figure 5.1:** Confusion matrix

Such a visualized matrix easily shows why a model is not effective. For the matrix in Figure 5.1 the most frequent occurrences are false negative predictions. Such predictions occur when the model mistakenly predicts positive data points as negative. Although such a high number of false negative predictions is not desired, when we analyzed how severe the effect of this error is, as recommended by Akkiraju et al. [39], we realized that such a high amount of false negatives is less severe than initially expected. This is because these false negative predictions would make the system only miss out on good investments, but it would not lose any money. As a result of this realization, we categorized this error as less important than, for example, a high amount of false positives predictions, since these incorrect predictions would make the deployed system lose money by making unprofitable investments. Therefore, we focused on reducing the number of false positive predictions. Another training error that occurred many times was underfitting the model. This was caused by using too many features. Some models, for example, were trained with more than 60 features. This error was also detected by using such a confusion matrix. Since this error resulted in low model
5.1 Effectiveness of AI Engineering Practices (RQ1)

quality, we categorized this as a severe error and prioritized fixing it. Additionally, we detected some less severe errors, for example, that the model does not consider any market correlation, resulting in some false predictions. For example, if in one week the stock price of most car companies is massively decreasing, this might indicate that one should not invest in any car companies during this week. Although not fixing this issue might result in the system losing money after thorough consideration, we categorized this error as too costly to fix. Thus, we ignored this issue and instead focused on improving the model in a less time-consuming manner.

**Analysis** Akkiraju et al. [39] propose that by prioritizing errors based on the severity and business value of fixing them, it is possible to spend time and resources more effectively to improve the model. Our case proved this thesis. By prioritizing errors, we avoided wasting time fixing those errors that barely impact the system’s effectiveness. Instead, we spent more time fixing those errors that have a strong impact on the system’s effectiveness. Therefore, we were able to develop a more effective system by using our time more efficiently. Thus, applying this practice improved the development process.

**Capture the Training Objective in a Metric That Is Easy To Measure and Understand**

**Application** We first defined the training objective for the machine learning model as predicting whether a stock increases by at least 1.6% during the next day with at least a 90% accuracy. However, the models developed according to this easily understandable training objective were ineffective. As a result, we had to redefine the training objective. This time, we defined the overall objective as developing a system that uses machine learning functionality to invest profitably in stocks. Since at this point we did not know how to achieve this objective, we had to experiment with multiple approaches. As a result, we had different training objectives depending on the approach. Examples are predicting whether a stock price will increase in the next five days above a specific threshold or predicting whether a stock price will decrease by at least 1.6% before the end of the current day. Furthermore, the precision required for the model to make the system profitable also varied depending on the approach. This made it difficult to incorporate the desired precision or accuracy into the training objective. We generally defined that the precision has to be high enough for the system to be profitable using this model. However, such a metric is difficult to measure since one needs to simulate how the deployed system would perform. Although this conflicted with the proposed practice, we decided to continue using “The system should make a profit” as part of the training objective because anything else would have been less effective.

The deployed system uses 19 different models. For each model, we defined the training objective as predicting whether a stock price will increase above a specific threshold. Additionally, we simulated how much profit the entire system would make using the predictions of all 19 models and used the results of this simulation to identify whether the general objective of making a profit is achieved. As a result of these easily understandable training objectives, during the second meeting with the machine learning engineer, in which he reviewed our machine learning component, we were able to quickly explain to him the overall objective, as well as the objectives of each individual model.

**Analysis** As the meeting with the machine learning engineer has shown, following this practice definitely simplifies the communication during such a development process, which is extremely helpful, especially when working in a team with several developers. However, one should expect
that the training objective may change during development. Ensuring that all team members are aware of the new training objective requires good communication within the team. Additionally, as our case has shown, there are cases where it is impossible to capture the training objective in a metric that is easy to measure. We had to develop a complex simulation to determine whether the training objective was achieved. As a result, although it is preferable to capture the training objective in an easily measurable metric, one might have to accept that it is impossible in their case.

Use Cross-Validation

Application  Almost always, we tested models using holdout cross-validation, which splits the entire data randomly into two mutually exclusive data sets [56], one for training the model and one for testing it. By splitting the data, we ensured that the results are more reliable since the model was tested only on data that was not part of the training set. To discover how important the use of cross-validation is, we trained and validated a random forest consisting of decision trees once with cross-validation and once without it. The random forest trained by using cross-validation achieved a precision of 65%, whereas the random forest trained without the use of cross-validation achieved a precision of over 99%. Such a strong increase in precision is clearly caused by testing the model on data that it had already used for training. This strongly indicates that test results can only be trusted when using cross-validation.

Furthermore, to validate a k-nearest-neighbor model, we used an efficient implementation of the leave-one-out cross-validation. The leave-one-out cross-validation uses all data points as a training set except one which is used as a validation set [57]. This process is carried out for each data point [57]. We decided to use this cross-validation technique since it enabled us to validate the model on more data. However, instead of splitting the data into two sets and repeating this for each data point, we modified the k-nearest-neighbor algorithm in such a way that it always ignores the nearest neighbor and instead also uses the k + 1 nearest neighbor to make a prediction. This implementation achieves the same results as the conventional technique of splitting the data in each iteration, but is much faster. To prove this, we additionally implemented the conventional technique. We tested both validation algorithms on the same model using the same 13,512 data points. Both algorithms determined the same model precision, but the conventional algorithm needed 128 seconds for this calculation, while the efficient implementation only needed 10 seconds.

We additionally aimed to identify how much cross-validation affects the test results. We tested three different k-nearest-neighbor models with \( k = 5 \), \( k = 50 \) and \( k = 100 \). Each model was validated once without cross-validation and once using leave-one-out cross-validation. Figure 5.2 shows the results of those validation procedures.
5.1 Effectiveness of AI Engineering Practices (RQ1)

Figure 5.2: Comparison between the scored precision when using cross-validation and the scored precision when not using cross-validation for k-nearest-neighbor models

Furthermore, we identified that the validation procedures always achieved different results when testing the same model because the data was randomly split into training and test sets. In some cases, the model’s precision would differ by up to 5% just by using different training and test data sets. As a consequence, to ensure that the scored precision is representative, it was critical to train and test the model several times and average the results.

Analysis As already proposed by many authors, the use of cross-validation is essential to develop machine learning systems. With our experiments, we have proven this thesis. However, as shown in Figure 5.2 the effect of using cross-validation varies strongly depending on the model. While for the model with \( k = 5 \) not using cross-validation, falsely indicates a more than 12% higher precision, for the model with \( k = 100 \) the calculated precision is only 0.29% higher compared to using cross-validation. Although in some cases, using cross-validation barely affects the results, we still recommend using it to ensure that the results are accurate.

Furthermore, one should be aware that for some specific models, there might be more efficient techniques to implement cross-validation. In our case, the more efficient implementation of the leave-one-out validation for the k-nearest-neighbor model was more than 12 times faster compared to the conventional technique. Cheng et al. [61] and Cawley et al. [62] also proposed several efficient strategies for certain cross-validation algorithms.

Continuously Measure Model Quality, Performance and Drift

Application During the tuning process of the machine learning models, it was necessary to continuously measure the model quality to identify whether the latest change improved or worsened the model. Furthermore, since we used the model quality to identify which model has the best potential and should thereby be focused on during the development, for example, after drastically changing the feature selection, we measured the new quality of each model to decide which model we should focus on. By doing so, we aimed to avoid futile use of our time. At the beginning, we measured the model quality using the model’s accuracy. However, we quickly realized that in this
particular case, the model’s precision is a better metric to describe the model quality. This is due to the fact that false positive predictions are unprofitable investments that the model predicts to be profitable, and would thereby make the system lose money. However, false negative predictions are profitable investments that the model predicts to be unprofitable and would thereby make the system miss out on a good investment, but the system would not lose any money. This meant that for the models in our case, having fewer false positives is more important than having fewer false negatives. Since the precision of a model only describes the relative number of true positives out of all positive predictions, measuring the model quality using the model’s precision is more helpful to improve the model in such a way that the system performs more effectively.

Many of the models we used had a strong bias, for example when the training data was unbalanced. This made us realize that when the deployed system retrieves new training data, it should always determine what effect this new data has on the model quality. As a result of this, we built the system in such a way that after training a model, it always measures the new model quality, and in the event that the model quality has decreased, the system continues with the previous version.

The final version of the system makes approximately 760 machine learning predictions each day before the stock market opens. On average, this takes 39 seconds. This time was important to determine at what point the system should start its decision-making process to decide on which stocks it should invest in. For such a short period of time like 39 seconds, we could implement the system to start the decision-making process 10 minutes before the stock market opens. However, if the machine learning component is drastically changed or more data is used, this decision-making process could take longer than 10 minutes, resulting in the system investing later than planned. To avoid such a negative effect, continuously measuring the performance of the selected model was critical.

During the development, we did not discover any model drift. However, we expected that the state of the stock market would impact the model’s precision. To verify this hypothesis, we tested one model purely on data from September 2008, since this was the month the stock market crashed. The model we chose for this test predicts whether a stock will increase its price by more than 2.6% during the next day. This model had a precision of 63% when tested on random data. However, when testing the model purely on data from September 2008, it only scored a precision of 59%. This indicates that the state of the market can have a significant impact on how well the model performs. While such a stock market crash occurs rarely, there are frequently events that also impact the stock market, for example, the war in Ukraine [63]. As a result of this war, some stocks experienced a massive price increase, such as the Rheinmetall stock [64] whereas some other stock prices decreased significantly, for instance, the EPAM Systems stock [65].

**Analysis** In our case, continuously measuring model quality was essential to avoid futile use of our time, measuring model performance was critical to ensure the system’s correctness, and measuring model drift was important to ensure that the system’s effectiveness does not unexpectedly decrease. In summary, applying this practice improved the development process and the resulting system.
Review Training Scripts

Application  At the beginning of the machine learning component development stage, we used the scikit-learn tool, which offers a variety of machine learning algorithms. By using this tool, we did not have an actual training script, since we only called the already implemented scikit-learn algorithms.

```python
def train(X_train, y_train):
    model = sklearn.svm.SVC()
    model.fit(X_train, y_train)
    return model
```

Figure 5.3: Training script

Such a short training script made a review useless. However, later in this development stage, we developed three training scripts for two different machine learning algorithms without using scikit-learn. The first algorithm groups data points into classes and makes predictions based on the most common label of all data points in that specific class. Reviewing the code sparked the idea of integrating an optimization procedure that aims to identify the optimal weights for this model. Although this later increased the model’s precision from 59% to 61%, this improvement contained two bugs. A variable was used for two different purposes. This caused an infinite loop. This bug was discovered because the training process took much longer than expected. As a result, we performed a review of the newly added code. After 5 minutes of reviewing the code, we had identified the bug. Fixing this bug took only a couple of seconds, since we only had to add another variable. Although discovering this bug was critical for continuing the development process, we only conducted the review due to our suspicion. If we had conducted the review as soon as the implementation was finished, we would have only saved 10 minutes.

The second bug was not discovered during the first review. Instead, the bug was detected because the newly added optimization process did not show any effect. This bug was caused by a floating-point error. The Python interpreter returns 0.10999999999999899 as the result of the calculation 1.011 - 1. The bug was discovered by debugging the newly added code. Afterwards, we reviewed the entire training script again, thereby discovering several ways to improve code readability, such as renaming variables or extracting methods.

The second training script we implemented is a k-nearest-neighbor algorithm. The first review we conducted of this training script did not reveal any bugs. However, it revealed several ways to increase code readability. Furthermore, we conducted a second review with a machine learning engineer. The engineer pointed out several ways in which the training script could be further improved, for example, by using NumPy to achieve greater performance and modifying the optimization procedure to achieve a better tuned model. We used his proposal to develop another k-nearest-neighbor training script. However, during the development of this second training script, a bug was erroneously introduced. Elements of a list are removed in a loop based on an index. Since removing an element shortens the list, in some cases, the index for the next loop iteration was not correct anymore. This bug was not identified during the conducted review because it was imperceptible. After testing the training script, we finally discovered this bug. However, fixing this bug introduced another bug, which was fortunately discovered quickly by reviewing the just modified code section.
Analysis   Conducting software code reviews is already a well-established best practice, as these reviews tend to improve software quality [66]. Our case has proven this thesis. By conducting such reviews, we were able to detect bugs in the code and consequently improve the software quality by fixing them. As our development process has shown, especially for such complex code as training scripts, some bugs are difficult to detect. Additionally, during the review with the machine learning engineer, we obtained new knowledge on how to further improve the training script, resulting in higher performance and model quality, thereby increasing the effectiveness of the overall system. We conclude that conducting such reviews is an important practice since it enables one to detect bugs and improve model quality. Furthermore, reviews can be helpful to increase code readability or obtain new knowledge from team members or stakeholders. However, one should be aware that one review might not be enough. In our case, by fixing bugs that were detected during a review, we unintentionally introduced new bugs. Therefore, we strongly recommend that after fixing bugs or modifying the training script, one should conduct another review to detect any newly introduced bugs. Nonetheless, in some particular cases, when tools such as scikit-learn are used, for example, this practice becomes obsolete.

Test all Feature Extraction Code

Application   During the development of the machine learning component, the most time was spent on the feature selection. This made it difficult to decide at which point we should write tests for the feature extraction code because these tests would be outdated quickly and would need to be changed frequently. This could cost unnecessary time, especially since the feature extraction code is highly complex. For instance, our first feature extraction script reads through thousands of lines just to create a single data point. This made the creation of dummy data to test this code extremely time-consuming. Instead, we decided to conduct tests manually by comparing the extracted features with the corresponding data. The first bug we discovered in the feature extraction code was caused by two duplicate lines:

```
X_train.append([c/o, h/c, l/c, v])
```

This line adds a new data point consisting of four features to the X_train list, which then is split into two lists that are used to train and test the model. However, by modifying one of these lines and accidentally forgetting to also modify the second line, the profitable data points had different features than the unprofitable data points. As a result, the tested model achieved an accuracy of 99.5%. Because such a high accuracy raised some suspicions, we decided to test the feature extraction code. We did this by logging the extracted features with the data used to create these features. In a matter of seconds, we realized that the data points looked different. For profitable data points, the last feature was an integer value, whereas for the unprofitable data points, the last feature was a float value. By reviewing the code, we quickly identified the cause of this. After fixing this bug, the model’s accuracy decreased massively. Therefore, we had to work extensively on improving the model. This drastically changed the feature extraction code. However, by doing so, another bug was accidentally introduced. This bug also falsely increased the model’s accuracy, but this time the model only achieved an accuracy of 90%. Due to this, we did not suspect any bugs and continued working on the feature selection for four more days before testing this code again. This time, it was not immediately obvious that a bug corrupted the extracted features. There were several points in the code at which features were created, depending on how many days it took until the stock’s
price reached a specific threshold. Therefore, we decided to use equivalence class testing to ensure that all lines of code that create features and add them to the training set are covered. For each equivalence class, we compared one of the extracted features with the corresponding stock data. We discovered that for most equivalence classes, the extracted features were in fact correct, but for one equivalence class, the extracted features were incorrect. After reviewing the code again, we finally detected this bug and were able to quickly fix it. However, the work done on the previous four days was unnecessary.

**Analysis** We draw five lessons from this experience. First, testing the feature extraction code is critical to ensure the model’s, and thereby also the system’s, functional correctness. Second, if we had tested this code more frequently, we would have saved four days of work. Third, if the model performs unexpectedly well, this should always raise suspicion, and one should verify whether this is caused by any bugs. Fourth, an existing bug might only corrupt a fraction of the data. To avoid overlooking such bugs, one should ensure that the conducted test covers the entire code, for example, by conducting equivalence class tests. Fifth, as many papers have already proposed for software development in general, one should avoid code duplicates because inconsistent changes to these duplicates can introduce bugs [67] and code duplicates often times lower software quality [68]. Since systems with machine learning functionality are commonly more complex, maintaining software quality is even more important to ensure that, for example, changes to the system can be made easily.

**Automate Hyper-Parameter Optimization and Model Selection**

**Application** Automating the hyper-parameter optimization always requires writing additional code. In some cases, especially when only considering a tiny set of values for the hyper-parameter, writing this additional code took more time than just manually trying out each combination of considered values for these hyper-parameters. However, in those cases in which we aimed to optimize a model for which each training process took several minutes, it was always more effective to automate this optimization process since this would allow us to use this time to work on something else.

Furthermore, we discovered that one always needs to pick a trade-off between the time the optimization process will take and how much this optimization will improve the model. This is due to the fact that the more data one uses and the more fine-grained the optimization is, the more time it will take until the optimization process is finished, but it will also be more likely that the optimization process achieves a better result. Therefore, we aimed to execute these hyper-parameter optimization processes during the night when we were not working. This meant that on the next day we could immediately use the results of the optimization process to further tune the model. Although such a long optimization process requires more resources and more advanced planning, the precision of the tuned models was always at least 0.5% better compared to the same model tuned with a hyper-parameter optimization process that took less than one hour.

Additionally, we discovered that the performance of the corresponding model can have a huge impact on the performance of the hyper-parameter optimization process. We used two different implementations of the k-nearest-neighbor algorithm. One using a conventional implementation and one using the NumPy Python tool. Since our final version uses 19 individual k-nearest-neighbor models, the aim of the hyper-parameter optimization was to find the optimal k in the range between
5 and 50 for each of these 19 models. For the k-nearest-neighbor model with the conventional implementation, this optimization process took more than 52 minutes, while for the model with the NumPy implementation, it only took 3 minutes. This made working with the NumPy implementation much more convenient. Alternatively, this increased performance could be used to execute a more fine-grained optimization process in the same amount of time. This would then most likely result in a higher model quality.

Furthermore, we implemented a method that would validate the precision of 13 different scikit-learn models. Implementing this method took us 15 minutes. On the contrary, the execution always took more than five hours. We ran this method every time significant changes were made during the feature selection process. We then selected the model that scored the highest precision to focus on since we assumed that this model has the best potential. The last time we executed this model, the scikit-learn k-nearest-neighbor model scored the highest precision. However, the deployed system uses a different k-nearest-neighbor model that was implemented without using scikit-learn because this model consistently achieved even better precision.

**Analysis**  Automating the hyper-parameter optimization has already been proposed by many authors due to its positive effect on the model quality [69][70][71][72]. Our development process has shown that this is definitely justified, since by using automated hyper-parameter optimization we managed to improve the model’s precision by more than 4% in some cases. However, if one wants to, for example, test how well a model performs using a different weighting algorithm, automating this process will in many cases cost more time than doing it manually. However, although it required extra code, automating most hyper-parameter optimization processes during our development was an essential step to increase the deployed model’s precision in order to make the system effective. Additionally, automating the model selection simplified the development process since it identified which model has the best potential and thereby enabled us to use our time and resources more effectively. However, depending on which models are considered during the model selection process, the computing time differs strongly. We encountered many models for which the validation process only took a few seconds. Therefore, if one only considers such models for the model selection, the computing time will be extremely short, but because we additionally considered many models for which the validation process took much longer, the model selection process took more than 5 hours. To allow one to easily plan ahead, one should be aware of how long such a model selection process might take.

Furthermore, since tuning each model would have been extremely time-consuming, we mostly used models that were not tuned. Thus, one may select a certain model to focus on, although there are other models with greater room for improvement. In addition, as our case has shown, one should be open to the possibility that an even better model may exist that has not been considered.
Log Predictions with the Model’s Version and Input Data

Application At the beginning of the feature selection, we had no use for the logged predictions since we were using between 20 and 65 features at this point. This made it difficult to analyze why the model made incorrect predictions. Instead, we aimed to mostly automate the feature selection. This made logging the predictions unnecessary. However, later in the feature selection process to further improve the model, we aimed to identify which additional features increase the model’s precision. To achieve this, we first analyzed the feature values for the logged predictions, but because such a large number of values is difficult to comprehend, we created charts of the logged data. We used two charts for each prediction, a candlestick chart of the OHLC values from those days that were used as input data, and another chart showing the volume trend on these days. By analyzing a series of these charts and adapting the selected features, we were able to increase the model’s precision from 62% to 66%. Although this made logging the input data extremely helpful, since we were only working on one model at the same time, we did not find any use for the logged model version at this point. This changed later in the development process, at which point we used predictions from a set of models each predicting a different stock price increase, for example, an increase of more than 2.1%. For this use case, logging the model’s version was essential to enable us to differentiate between these models.

One version of the system we worked on would make approximately 50,000 predictions every day if deployed. Logging all of these predictions would approximately create 12 gigabytes of data each year. Because we expect this to be highly inefficient, if we had deployed this version, we would have only logged the positive predictions that the system actually invests in. This would have reduced the size of created data to less than 120 megabytes each year.

Analysis During the development, logging predictions is an essential step to generate insights on how the model can be further improved. Our case has shown that this can significantly improve the model quality and is thereby an effective practice. Furthermore, because developers usually aim to improve the model even after deployment, logging the predictions made by the deployed model is definitely not a bad choice. However, in those cases where the model frequently makes thousands of predictions, one should consider before deployment how this huge amount of data should be handled, for example, by implementing a more advanced infrastructure.

Collaborate with Multidisciplinary Stakeholders

Application We conducted three meetings with two external stakeholders. The first meeting was with a professional stock broker and took 44 minutes. The purpose of this meeting was to review the completed design of the system. Furthermore, during the requirements analysis and design stages, we wrote down domain-specific questions for anything we were uncertain about, to then obtain an expert’s opinion on these questions. In total, we had 10 questions written down. At the beginning of the meeting, we explained the current design of the system. This was followed by the stock broker giving feedback. Subsequently, we used those 10 questions as a guide throughout the remaining meeting. We then used the new insights acquired by the answers of the stock broker to ask additional questions to obtain even more domain knowledge. The most interesting answer was to the question: “How high does the precision of the machine learning model need to be?” We had expected that the deployed machine learning model would be required to have a precision of at least
90%. However, the stock broker explained to us that even for the best stock traders, only 55-60% of their investments become profitable. This was highly important information since it made us realize that instead of trying to develop a model that almost never makes incorrect predictions, we should instead aim to develop a model whose precision is also in the 55-60% range, but additionally focus on reducing the amount of money the system loses on its unprofitable investments and increasing the amount of money it gains on its profitable investments. Due to the fact that this changed the system’s requirements, it additionally impacted the system’s design. Nonetheless, this turned out to be the right decision. Although we were able to develop a model that predicted profitable investments with a precision of 83%, the remaining 17% of unprofitable investments the deployed system would make would on average lose more money than the 83% of profitable investments gain. As a result, we decided to stop working on this model and instead focused on models that lose less money with their false predictions.

The second meeting was held with a machine learning engineer and took 54 minutes. The purpose of this meeting was to further improve the design of the system. Since we had already started developing the machine learning component, we were also able to obtain feedback on how to further improve it. We received a large amount of advice for different stages of the machine learning development process, such as model selection, feature selection, hyper-parameter optimization, and deployment. Since at this point we did not have much experience developing such systems, this advice proved to be extremely helpful. For example, he gave us the advice to calculate the feature importance for each feature to identify which features are improving the model. We followed this advice several times for different models and always achieved increasing the model’s precision.

The last meeting was held with the same machine learning engineer and took 70 minutes. The purpose of this meeting was to review the finished system. The engineer pointed out multiple improvements, such as using NumPy which resulted in the model performing more than 20 times faster. Furthermore, he pointed out a way to improve the hyper-parameter optimization algorithm. This increased the model’s precision by 1.5%.

**Analysis** Since AI is applied in many different domains, developers often need to collaborate with stakeholders who possess the specific domain knowledge. Our case has shown that a deep understanding of the domain is critical to develop an effective system. If developers lack this knowledge, it is essential to collaborate with domain experts. Priorkowski et al. [73] have conducted a study at IBM and found that, in contrast to our development process, AI developers at IBM repeatedly ask domain experts throughout the development. Priorkowski et al. state that this is due to the fact that domain knowledge cannot be transferred during one meeting, since as the team progresses during the development, new questions arise. During our development process, we experienced the same. After the meeting with the stock market expert, as we progressed during the development, new questions about the stock market domain arose. This shows that the collaboration between developers and domain experts should be carried out continuously throughout the development process rather than conducting a single meeting.

Additionally, the meeting with the stock market expert was also important to cover the human-centered aspect of our development process. The aim of this development process was to develop a machine learning system that invests the money of human users. As a result, minimizing the risks that the system loses money is critical to ensure user satisfaction. Since during the meeting with the stock market expert, we also obtained knowledge on minimizing these risks, this meeting incorporated a human-centered AI perspective proposed by many authors, such as Riedl [74] and Qadir et al. [75]. Furthermore, because the system we developed uses investment strategies invented
by humans but on a much larger scale, since this system makes investment decisions on thousands of stocks in a matter of seconds, it thereby amplifies the human ability to make profitable investment decisions. Providing such superhuman capabilities to empower humans is seen as another aspect of human-centered AI [76].

In addition, due to the fact that the meetings with the machine learning engineer improved the system’s performance and effectiveness, this case shows that conducting reviews with other machine learning developers to share knowledge can significantly improve software quality. The fact that such code reviews can increase software quality is already a common best practice for general software development [77], due to the increased complexity of machine learning systems, we expect such reviews to have an even greater effect during AI development processes. In our case, the second review with the machine learning engineer resulted in the model being 20 times faster. As a result, conducting such reviews within the team or with external stakeholders is definitely an important practice.
5 Results

5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

How Should the Machine Learning Functionality Be Incorporated?

**Challenge Description** The most fundamental question during the design stage was: in what stocks should the system consider investing? Our first idea was to let the system prefilter the possible investments so as not to overload the machine learning component. There are plenty of prefiltering algorithms that the system could use, such as searching for stocks that match a certain pattern, for example, a strong price increase. The machine learning component could then make predictions on the possible investments that were not filtered out. Alternatively, the system could do no prefiltering at all. Instead, the system could always let the machine learning component make a prediction and invest based on these predictions. To avoid wasting time, it was important to know early on what the overall investment strategy would be and what prefiltering would work best since the machine learning model would differ depending on the chosen prefiltering functionality. Furthermore, deciding what kind of stock data the model should use for its predictions turned out to be quite challenging. During the design stage, the use of intraday data seemed the most promising, but during the development stage, none of the models trained with such intraday data performed as well as expected. There was always a lack of precision, which caused all models to fail the requirements established in the requirements analysis stage. This raised the question whether there might be other data, such as daily stock data, that would achieve better results. Additionally, depending on how the machine learning component is incorporated, the requirements differ. For example, if the model only uses daily stock data, the predictions can be made before the stock market opens. Thus, the model has multiple hours to calculate the predictions. However, if the model uses intraday data to make predictions based on stock price changes during the last minute, it would be essential that the model finishes its prediction in a matter of seconds. These differences in the requirements would change the way the overall system is implemented, and in some cases, it might even be impossible to develop a system that satisfies these requirements.

**Solution** Since during the design stage we did not know what the best way to incorporate the machine learning component would be, we decided to design the system based on what we expected to be the best way. We then first developed the overall system according to this design before starting the development of the machine learning component. This turned out to be a huge mistake. Because the first design did not prove to be effective, we had to incorporate the machine learning component differently. As a result, the overall system also needed to be drastically changed. In conclusion, first developing the overall system while being uncertain about the design’s effectiveness wasted a lot of time. To avoid making the same mistake again, we decided to change the overall system as soon as we identified the most effective way to incorporate the machine learning component. To detect the most effective way, we implemented a variety of machine learning models according to different investment strategies, such as waiting for the stock price to make an upward jump and then betting against this stock. We then compared these models based on their precision. In the end, the most effective strategy we discovered was to buy stocks as soon as the stock market opens based on the model’s predictions using daily stock data but only considering stocks that increased their price by at least 10% on the previous day, and to sell the bought shares as soon as the predicted profit
5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

or a predetermined loss is reached or at the end of the day. We then changed the overall system according to this new design. Therefore, we were not required to make any more changes to the overall system.

Available Data for the Deployed System

Challenge Description So that the model can make a decision about which stocks the system should invest in, it needs data. However, most APIs that offer such stock data have request limits. When using intraday data, it would be essential to retrieve current stock prices multiple times each minute. But due to these API limits, the system can only do this for a couple of stocks, depending on how high the request limit is. This would drastically reduce the effectiveness of the system since the fewer stocks are observed, the less likely it becomes for the system to find a profitable investment. To avoid this, we decided during the design stage to use the lemon.markets API\(^1\) since the limits for this API were 200 requests per minute. As this seemed to be more than enough, we developed the overall system using this API. Later, during the development of the machine learning component, the lemon.markets API changed these limits to only 10 requests per minute. In the design stage, we planned to retrieve the prices every 10 seconds for each stock that is being observed. Since with each request the data for one stock can be received, this means that the system is only able to observe 60 stocks. As a result, continuing to use the lemon.markets API would have reduced the system’s effectiveness.

Solution Since the lemon.markets API changed its request limits to a much lower rate, we first decided to restrict the system to only consider investing in a small set of stocks. The selection of these stocks would be made every day. For example, choosing the 60 stocks that increased the most on the previous day. However, later we discovered the Alpaca API\(^2\). This API offers similar functionality as the lemon.markets API, but the Alpaca API allows users to make up to 1000 API requests per minute. Just like the lemon.markets API, one request to the Alpaca API can retrieve the data for multiple stocks. But whereas for the lemon.markets API the number of stocks per request is limited to ten, for the Alpaca API there is no such limit. The number of stocks per request is limited only by the Python request module that only allows HTTP requests with fewer than 65549 characters. On average, this equals a request for 9580 stocks. Since there are only 10996 stocks that can be traded using the Alpaca API, only two requests are needed to retrieve the most current data for all of these stocks. Because of this enormous advantage, we switched to using the Alpaca API instead of the lemon.markets API. The downside to this was that it required a lot of work to change which API the system uses.

When we first tested the Alpaca API, we discovered that longer requests take significantly longer to receive a response. As a result, we decided to use each request for 1000 stocks. By using these smaller requests and with the use of multithreading, it only took on average 8.7 seconds to retrieve the data for all available stocks on the Alpaca API. Therefore, we no longer had to worry about what data is available for the deployed system.

\(^1\)https://www.lemon.markets
\(^2\)https://alpaca.markets
Finding the Right Data Source

**Challenge Description**  Finding usable data to train the machine learning model proved to be particularly challenging. In the first design version, we decided that the most effective data for this system would be intraday stock data in one-minute intervals. Because such data has an enormous size, it was difficult to find a source that offers such data. We found a few APIs that offered such historical intraday stock data, but the quality of the data was never perfect.

![Figure 5.4: Snippet of the received OHLC stock data for Nutex Health Inc.](https://www.alphavantage.co)

Many files, such as the depicted file in Figure 5.4, have gaps. These gaps are usually only a few minutes long, but in some cases several hours are missing. Because we expected that these gaps have a serious impact on the model quality, selecting the right data source seemed to be critical for the success of this development process.

**Solution**  To find the optimal source, we conducted a literature survey. This survey revealed that the most popular API for this purpose is the Alpha Vantage API\(^3\). There were also several other APIs that were proposed. But after retrieving a small set of files from the most proposed APIs, it became clear that the data from the Alpha Vantage contained the smallest gaps. Since we expected such gaps to have a significant effect on the machine learning component, we decided to use the Alpha Vantage API.

**Receiving the Data**

**Challenge Description**  The Alpha Vantage API offers stock data for the last 720 days. The data for each stock is split up into 24 CSV files, each containing the data for one stock for 30 days. The API only allows 500 requests per day. Each request retrieves only one of these files. Since there are 4927 stocks that can be traded on the Nasdaq stock exchange, retrieving the data for all 4927 stocks takes 237 days. Especially challenging was the fact that the most recent file contains the data for the last 30 days, the second most recent file contains the data for the time period between the 31st most recent day and the 60th most recent day, and so on. As a result, if we had received the most recent file of a stock on one day and the second most recent file on the next day, since the second file was received a day later, the last day in the first file would be identical to the first day in the second file.

\(^3\)https://www.alphavantage.co
5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

To avoid such duplicates for each stock, we had to retrieve all files on the same day. Additionally, the API also has a limit of 5 requests per minute. Due to this, executing these 500 requests takes 100 minutes. Another issue was that although the Alpha Vantage API offers stock data for the last 720 days, on average only the data for the last 497 days is available.

**Solution**  We made the decision to focus only on those stocks whose combined value of all shares (market cap) exceeds two billion dollars. There are only 707 Nasdaq stocks with a market cap greater than two billion dollars. Retrieving the data for these 707 stocks would require only 16,968 API requests. With 500 requests per day, this would take 34 days. However, because of the additional five requests per minute limit, we had to develop a Python script that in a loop would request one file and then wait for 12 seconds. After receiving the data for 412 stocks, we decided to first focus on developing an effective model that performs well on this data before retrieving more data.

**Time-Consuming Feature Extraction**

**Challenge Description**  At the beginning of the machine learning component development stage, at which point we still trained models using intraday data, the size of the training data was 3.57 gigabytes. Extracting features from this data always took between 3 and 18 minutes. This massively slowed down the tuning process. Since during the tuning process we were frequently changing which features were being used, storing the extracted features would not help solve this issue. Later in the development process, we started using only daily data. For this data, it took only 14 seconds on average to extract the features. Since this short period of time could be used to reflect on the changes made and think about further improvements, the time needed for the feature extraction was no longer an issue.

**Solution**  To reduce the time required to extract the features from the intraday data, we decided to only use the data for 10 stocks. Since extracting the features for these 10 stocks took less than a minute, this sped up the tuning process. As soon as a model performed satisfactorily on this small set of data, we planned to then start testing the model on a larger set of data. After the feature selection process, we stored the extracted features because loading them would always take less than one second, thereby massively increasing the performance. This completely solved this challenge, but storing and loading the extracted features every time they are needed is only effective when the selection of which features are used does not change frequently.
5 Results

Data Quality Issues

Challenge Description Many intraday files contained gaps. In most cases, only a couple of minutes were missing, but in some cases, there were even multiple hours missing. One exceptional case was the stock of Chesapeake Energy. Of the 24 received files, only the first 15 and the last two files contained data. The seven files between them were empty. This resulted in a 210-day gap. We assumed that such gaps negatively affect the machine learning component. However, it was difficult to estimate how large the impact of these gaps is. This made it challenging to decide what gaps were acceptable and could be ignored and what gaps needed to be addressed. On average, each file contained 6437 lines. However, there were also files where most of the data was missing. Instead, they contained only a couple of lines.

Figure 5.5: 12th received CSV file of the Liberty Sirius XM Group Series B stock

These files were useless for the machine learning component, but manually removing them takes a lot of time. Writing a script that automatically removes all files below a certain size also requires time and knowledge of what an adequate threshold is.

Solution Since this system is supposed to work autonomously, we determined that it is essential that the system automatically resolves these data quality issues. To achieve this, we implemented an automated quality check procedure. This procedure would count the amount of lines that each file contains. If a file contains fewer than 100 lines, all files for this stock would be automatically deleted to prevent the machine learning component from training with this data. Since with this solution many gaps in the data remained, we also considered using data from different sources to fill these gaps. This would definitely achieve the best results, but this is an extremely complex task, because it would require a more complex infrastructure to handle this additional data. Furthermore, modifying the script to retrieve the missing data and fill up the corresponding gaps would have been enormously time-consuming. However, this was not necessary since we decided to test the models on only 10 stocks, as testing these models with all the data in some cases took up to 18 minutes. For these 10 stocks, we chose the stocks whose data had the best quality. This made the poor quality of the other data irrelevant at that point of time. Since we did not know whether this was the right data anyway, we decided to not put more effort into improving the data quality until we developed a model that would be effective on the data for those 10 stocks. This turned out to be the right decision since we later discovered that the models using daily data performed much better than the models using intraday data. Because of this, we stopped working on models using intraday data, and the deployed system does not even contain this intraday data. This means that if we had put more effort into improving the data quality before starting the model development, we would...
5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

have wasted a significant amount of time. Therefore, we recommend first developing a model that satisfies one’s expectations on a small set of high-quality data before massively spending time to improve the overall data quality, especially if one is uncertain whether this data is the right choice.

Unwieldy Amount of Files

Challenge Description We received a total of 9888 files for 412 stocks. Using these files to train the machine learning model would make the training process much more complicated. The training script would be much simpler if the data for each stock was merged into one file. In this case, the training script would only have to read 412 individual files that are not linked together. Furthermore, managing such a large number of files is more complicated.

Solution To reduce the number of files, we modified the Python script that retrieves the stock data to merge all 24 files of one stock into a single file after receiving them. Modifying the Python script did require additional work, especially since the first line of every original file contained the latest values, but extracting the features required reading from the oldest line to the newest. As a result, for each stock, the contents of all 24 files needed to be reversed. However, since this massively simplified the feature extraction code because for each stock, only one file needed to be read, merging these files thereby improved the code interpretability and simplified the remaining development.

Different Types of Data

Challenge Description Since the models using intraday data did not perform as well as expected, we decided to focus on training models using daily stock data. To get such data, we could have simply reused the intraday data we already had by calculating the daily OHLC values for each day, but this would have slowed down the performance of the feature extraction code. Furthermore, since daily stock data is much smaller, the Alpha Vantage API offers such stock data for a longer period of time than the intraday data. Because we assumed that more data would improve the machine learning functionality, we decided to additionally retrieve daily stock data from the Alpha Vantage API. Due to its small size, the data for each stock was packed into one file. This made it easier to use these files compared to the intraday files. But there was still an issue with these daily files. Surprisingly, the data in these files was in a completely different format compared to the intraday files.
Handling two different types of data in different formats seemed to unnecessarily complicate the development process. We planned to reuse most of the already existing training script, but the different format for the daily data would have required making many changes, which would have cost unnecessary time.

**Solution**  To avoid working with different data formats, we extended the Python script that retrieves the daily data to automatically change the data into the same format as the intraday data. The implementation of this additional code took approximately 2 hours. This was due to the fact that this new code introduced several bugs that corrupted the data. If we had not discovered them, this could have led to disastrous consequences. This shows that when transforming the training data, one should always carefully ensure that the data quality does not decrease.

**Feature Selection**

**Challenge Description**  The most challenging task during the development of the machine learning component was deciding what features might indicate an upward stock movement and are thereby useful for the machine learning model. This was due to the fact that there are a large number of possible features that one can choose from, and additionally, we lacked the stock trading expertise a professional trader has. This made identifying the most effective features extremely challenging.

**Solution**  To find the best features, we tested models using a wide variety of features. For each feature, we then calculated the corresponding feature importance.
5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

We plotted the importance scores to easily identify which features had a negative impact on the model and consequently adapted the selected features. Later, during the tuning process, we used two types of charts to visualize the input data of false predictions made by the models to identify new features that could improve the model’s precision.

**Figure 5.7:** Feature importance chart

![Feature importance chart](image)

**Figure 5.8:** Candlestick chart of the NetEase stock

![Candlestick chart](image)
The first chart we analyzed was a candlestick chart, consisting of seven candlesticks. The first six visualize the input data that the model used for its prediction. The last candlestick shows the day on which the model considered investing, indicating how profitable the investment would have been.

![Volume chart of the NetEase stock](image)

**Figure 5.9:** Volume chart of the NetEase stock

The second chart shows the stock’s volume on the same days. By analyzing a series of these charts, we aimed to detect patterns that are likely to indicate an upward stock movement. We then identified with what features the model is able to recognize these patterns. In one case, by using these new features, we were able to increase the precision of the corresponding model from 62% to 66%. However, since none of the models performed as well as expected, we additionally conducted a literature survey on similar machine learning models. We then trained our models using the features proposed by different authors. The best precision was achieved using the features presented by Vijn et al. [81]. They propose the following features:

1. Stock High minus Low price \((H - L)\)
2. Stock Close minus Open price \((C - O)\)
3. Stock price’s seven days moving average
4. Stock price’s fourteen days moving average
5. Stock price’s twenty-one days moving average
6. Stock price’s standard deviation for the past seven days

Furthermore, we added the stock’s volume as another feature since this increased the model’s precision even further. Moreover, by testing different variations of these proposed features, we discovered that the model can be improved by altering the first two features.

1. \((H/L) - 1\)
5.2 What Challenges Did Occur (RQ2) and How Were These Challenges Addressed (RQ3)

2. \( (C/O) = 1 \)

By using these two features instead of the first two proposed by Vijh et al., we improved the model’s precision by 2%. Finally, we deployed the model using these seven features. We conclude that calculating the importance of each feature and visualizing the input data of the model’s false predictions are extremely effective practices during the feature selection process. Furthermore, conducting a literature survey to identify proposed feature selections can save a huge amount of time. However, one should be aware that the proposed features may not be the optimal selection and one should therefore spent time with the aim to further improve the proposed selection.

Model Selection

**Challenge Description** At the beginning of the machine learning component development stage, a literature survey revealed that the use of a support vector machine (SVM) model to predict stock movements is highly effective [82][83][84]. Therefore, we decided to also use such a model. However, this SVM model performed poorly. As a result, we started testing many different models using different machine learning algorithms and different data sets (intraday or daily data). Since none of these models showed any great results, we were unable to identify which model had the most potential and should thereby be focused on. Especially difficult was deciding which type of data should be used. However, due to the fact that further development of any of these models is immensely time-consuming, to avoid wasting time, it was important to know early on which model we should be focusing on.

**Solution** Since we were unable to determine which data has the greatest potential, we focused on the models using daily data since this was proposed by many authors [85][86][87]. Furthermore, we used an automated model selection method to detect the model with the highest precision. The first time we ran this method, the decision tree model achieved the best results. We thereby focused on improving this model, but because we were frequently changing what features the model uses, we were certain that it was likely that with these new features a different model would achieve better results. Therefore, we decided to continuously execute this model selection method to identify whether a different model performs better when using these new features. At the end of the development process, the model selection method showed that the k-nearest-neighbor model performs best. As a result, from this point on, we focused on developing an even more effective k-nearest-neighbor model and finally deployed the system using this machine learning algorithm.

Biased Models

**Challenge Description** At one point during the machine learning component development stage, all the models we tested would either almost always suggest investing or almost always suggest not investing. For many models, this is common behavior when trained with unbalanced data. Since we trained the models with prefiltered data, such as only considering investing in stocks whose prices increased by 10% the day before, the training data contained up to five times more profitable investments. However, balancing the training data did not solve this issue. These models would still be strongly biased towards either positive or negative predictions.
Solution We partially solved this problem by testing even more models and eventually discovering that the decision tree and k-nearest-neighbor models only had a small bias. This bias was reduced even more by improving the feature selection. We assume this is due to the fact that these new features are better indicators of the profitability of an investment and, as a result, improve the decision-making capabilities of the model. Furthermore, we modified the k-nearest-neighbor algorithm to include the number of data points for each label as a weight, which reduced the model’s bias even more.

5.2.1 Analysis of Challenges

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>How Should the Machine Learning Functionality Be Incorporated?</td>
</tr>
<tr>
<td>2</td>
<td>Available Data for the Deployed System</td>
</tr>
<tr>
<td>3</td>
<td>Finding the Right Data Source</td>
</tr>
<tr>
<td>4</td>
<td>Receiving the Data</td>
</tr>
<tr>
<td>5</td>
<td>Time-Consuming Feature Extraction</td>
</tr>
<tr>
<td>6</td>
<td>Data Quality Issues</td>
</tr>
<tr>
<td>7</td>
<td>Unwieldy Amount of Files</td>
</tr>
<tr>
<td>8</td>
<td>Different Types of Data</td>
</tr>
<tr>
<td>9</td>
<td>Feature Selection</td>
</tr>
<tr>
<td>10</td>
<td>Model Selection</td>
</tr>
<tr>
<td>11</td>
<td>Biased Models</td>
</tr>
</tbody>
</table>

Table 5.1: Challenges occurred during the case study

We found existing literature on all of the challenges we encountered, except for the first two challenges. We assume that this is caused by the fact that these two challenges are highly domain-specific and thereby do not occur in most development processes. Gandomi et al. [88] state that some sources are unreliable, thereby using data from such sources requires more work to preprocess this data. During our development process, we also encountered many unreliable sources, resulting in the challenging task of finding the optimal data source (Challenge 3). Furthermore, Zhou et al. [89] have described receiving data (Challenge 4) as a challenge, since, depending on the domain, it can be a costly process. L’Heureux et al. [44] name processing performance as a challenge when working with a large amount of data, since the more data one uses, the more time is needed to perform the computations. In our case, the volume of data resulted in the feature extraction becoming extremely time-consuming (Challenge 5), thus negatively impacting the development process. L’Heureux et al. propose adapting how data is stored and processed to improve performance. This agrees with our solution to store the extracted features. In addition, L’Heureux et al. also present data quality issues (Challenge 6) as a common issue. To overcome this challenge, they propose cleaning the data, for instance, removing low-quality data. This is also the solution we applied by removing data that contained gaps. Furthermore, L’Heureux et al. state that a high volume of data, for example an unwieldy amount of files (Challenge 7) is challenging during the development. To overcome this challenge, they propose data manipulations, such as transforming the data. We applied the same solution by merging all 24 files for each stock into a single file. Additionally, L’Heureux et al. present handling data with semantic heterogeneity, for example, different types of data (Challenge 8).
as a challenge. However, they do not propose any solutions to this challenge. Moreover, L’Heureux et al. state that feature selection (Challenge 9) is a challenging task due to spurious correlations, for instance. They propose using deep learning algorithms since “it uses data representations rather than explicit data features” [44]. Zhou et al. [89] also name feature selection as a challenge, and furthermore agree that the use of deep learning algorithms overcomes this challenge. However, during our development process, instead of using complex deep learning algorithms, we analyzed the predictions the model made using charts and calculated the importance of each feature to improve the feature selection. Additionally, we conducted a literature survey to make use of already existing work. In our case, the literature survey was highly effective. This was mainly due to the fact that using machine learning functionality to make stock market predictions is an extremely popular problem. However, such a solution will not be effective for those machine learning problems for which only a small amount of literature exists. In addition, L’Heureux et al.[44] present variance and bias (Challenge 11) as challenges during such a machine learning development process. They argue that, especially when using large amounts of data, the model might become biased, and thereby lead to inaccurate output. However, they do not present a solution to this challenge. Lastly, Monteiro et al. [90] present model selection (Challenge 10) as a challenging task due to the variety of machine learning algorithms and possible configurations. To overcome this challenge, they propose automated model selection. Although this is also the approach we chose, we only used automated model selection to identify the model with the best potential, and then used automated hyper-parameter optimization to identify the best configuration for this model. On the other hand, they propose considering several configurations for each model during the model selection. We expect this to be even more effective. However, this will drastically decrease the performance of the model selection script. Since in our case, the computing time of the model selection method was already more than five hours, considering multiple configurations for each model would have required a significant amount of additional time and resources.
5 Results

5.3 Helpful Tools (RQ4)

During the development of the machine learning component, we used different tools to simplify the development process. In this section, we analyze how helpful each tool was during our development process.

5.3.1 scikit-learn

Application

The first tool we used is scikit-learn⁴. Scikit-learn is a Python module for developing machine learning models. It offers a variety of different machine learning algorithms. Instead of having to implement such a complex algorithm, one can simply create a model using the following line:

```
model = sklearn.svm.SVC()
```

This example code creates a support vector machine model. To train this model, the `fit` method is called with the training data and the corresponding labels:

```
model.fit(x_train, y_train)
```

To then make a prediction, the `predict` method is called:

```
model.predict(X)
```

Since by using this tool we did not have to implement machine learning algorithms ourselves, we saved a tremendous amount of time. Additionally, since only a few lines of code are needed to test a machine learning algorithm, in a short period of time, we were able to implement a model selection method that validates many different machine learning algorithms to determine which algorithm has the best potential. Another advantage of scikit-learn is that it offers a variety of settings for each model, such as different weighting methods. This made it very easy to improve the models since we did not need to have a lot of knowledge about these different settings. We were able to just try each of them and compare the results to identify which performs best. However, using a tool like scikit-learn comes with a downside. Without the use of any tool, we implemented a k-nearest-neighbor algorithm ourselves. This provided much more flexibility, which allowed us to implement it in a way so that the model is more likely to return a negative prediction to increase the precision. Because scikit-learn does not offer such a setting for its k-nearest-neighbor algorithm, the highest precision achieved by the scikit-learn model was only 71.3%, whereas our implementation achieved a precision of 73.3%. On the other hand, the scikit-learn model achieved a higher performance than the model we implemented ourselves. The scikit-learn model needed approximately 0.24 seconds to make 40540 predictions, whereas our implementation of the k-nearest-neighbor algorithm needed approximately 0.31 seconds for the same number of predictions.

⁴https://scikit-learn.org
Analysis

Using scikit-learn enabled us to easily implement new models with just a few lines of code and, in addition, enabled us to drastically change the model’s functionality by simply changing the parameters passed to the model’s constructor, such as selecting a different weighting function. This massively simplified the model selection and hyper-parameter optimization processes and additionally increased the system’s changeability. However, this increased changeability is limited by the functionality that scikit-learn offers. For instance, if one wants to use a specific weighting algorithm whose implementation is not included in the scikit-learn package, one has no choice but to implement the entire machine learning algorithm without using scikit-learn. Such changes would be highly time-consuming, thereby reducing the overall changeability. Furthermore, in our case, the best precision was achieved by the k-nearest-neighbor algorithm that we implemented without using scikit-learn. As a result, using a scikit-learn model in the deployed system would lead to more false predictions. Due to the fact that the system loses money on these false predictions, this would additionally decrease the system’s effectiveness. However, on the other hand, using the scikit-learn k-nearest-neighbor model would increase the system’s overall performance. In conclusion, as already proposed in several papers [91][92][93] due to the fact that scikit-learn massively simplifies the implementation of machine learning models, one should always consider using this tool. However, as Hao et al. [91] have already stated, there exist cases for which scikit-learn does not offer the optimal methods. The development process we carried out is one of these cases. The k-nearest-neighbor algorithm we implemented without using scikit-learn proved to be more effective, and thereby improved the overall system. As a result, the use of scikit-learn in our case was not as effective as expected. We thereby recommend also considering implementing machine learning algorithms without using scikit-learn, especially when the scikit-learn models perform poorly.

5.3.2 pickle

Application

Pickle\(^3\) is a Python module for object serialization. We used it to store the trained scikit-learn models. This only required a few lines:

```python
file = open(filename, 'wb')
pickle.dump(model, file)
```

Loading the model was also easily accomplished:

```python
file = open(filename, 'rb')
model = pickle.load(file)
```

Since the training process of some models took several minutes, we saved a large amount of time by storing the trained models using pickle.

\(^3\)https://docs.python.org/3/library/pickle.html
5 Results

Analysis

Due to its ease of use, pickle simplified the development process. It enabled us to directly store the Python object of a scikit-learn model. However, since pickle converts these Python objects into a byte stream [94] one cannot view or modify this data, thus reducing the system’s interpretability and changeability. Additionally, the use of pickle decreases the system’s security since it is possible to create malicious pickle data that can cause the execution of arbitrary code [94]. Alternatively, one can convert the attributes of such a Python object to a JSON\(^6\) string and store this string in a JSON file [95]. Although this process is more complex, in contrast to pickle, the created JSON files can be viewed and modified. Thus, the use of JSON increases not only the system’s security but also its interpretability and changeability.

5.3.3 NumPy

Application

NumPy\(^7\) is a Python library for implementing numerical computations efficiently [96]. Our first implementation of the k-nearest-neighbor algorithm was done without using NumPy. During the training process, this algorithm determines the optimal k. This process took on average 185 seconds to identify the optimal k out of 50 values. We then implemented a second version of the k-nearest-neighbor algorithm using NumPy. This reduced the time needed for the training process to an average of 9 seconds. While the model using NumPy is much faster, the precision was identical to the model implemented without using NumPy. However, since the increased performance is caused by NumPy computing the euclidean distance to all training data points at once instead of iterating over the training data and computing the euclidean distance individually for each point, this algorithm is more complex, which reduces the readability and makes it more likely that bugs are introduced. In our case, the algorithm contained two bugs that we only detected by performing a back-to-back test in which we discovered that the precision of the algorithm using NumPy was 2% lower than the precision of the first algorithm not using NumPy.

Analysis

Many papers propose using NumPy [97][98][99][100] to achieve a higher performance. In our case, we significantly increased the system’s performance by making use of the numerical calculations on multi-dimensional arrays offered by NumPy. However, this also created additional complexity, thus decreasing the system’s simplicity and readability. As a result of this, two bugs were introduced unintentionally. If we had not detected and fixed these bugs, they would have reduced the system’s functional correctness. We conclude that the NumPy tool can be extremely powerful, but when using this tool, one should always intensively review the corresponding code to ensure the code’s correctness.

\(^6\)https://www.json.org/json-de.html
\(^7\)https://numpy.org
5.3 Helpful Tools (RQ4)

5.3.4 Pandas

Application

Pandas\(^8\) is a Python library for data manipulation and analysis [101]. We used this library to create Pandas DataFrames that contained the extracted features. We then stored these DataFrames to avoid having to extract these features every time we trained a model. Extracting these features took between 14 seconds and 7 minutes, depending on the data used. By storing those features in a Pandas DataFrame, we only needed to extract them once. After that, we would load the features from the corresponding DataFrame. This always took less than a second, resulting in an extreme time gain. Reading through a DataFrame of 13,512 data points 1000 times took in total 156.73 seconds. To have a comparison, we also stored the 13,512 data points in a CSV file. Reading through this CSV file 1000 times took only 72.54 seconds. This is more than twice as fast.

Analysis

While several papers have already proposed Pandas as a powerful tool for data analysis [102] [103] [104], Pandas was only partially helpful during our development process. This is due to several reasons. We received the data in CSV files, which made it easier to extract the features directly from these files, as this process would usually take less than 30 seconds. This made implementing the feature extraction using Pandas too time-consuming. Furthermore, although we could have used Pandas for our implementation of the k-nearest-neighbor model, using NumPy seemed more suitable for this algorithm since it offers calculations on multi-dimensional arrays. As a result, we only used Pandas to store and load the extracted features. Although using CSV files instead of Pandas for this use case was faster, we continued using Pandas instead because the time difference was on average below 0.1 seconds, and using Pandas was less complex, and thereby increased the system’s simplicity and interpretability.

5.3.5 Optunity

Application

To simplify the hyper-parameter optimization process, we decided to use the Python library Optunity\(^9\). We first used Optunity to tune a scikit-learn decision tree model that predicts whether a stock will increase by at least 1.6% during the next day. We were aiming to find the optimal values for the parameters \texttt{max\_depth}, \texttt{min\_samples\_split} and \texttt{min\_samples\_leaf}. To solve such a hyper-parameter optimization problem, Optunity offers a variety of solvers:

- Grid Search
- Random Search
- Particle Swarm Optimization

\(^8\)https://pandas.pydata.org
\(^9\)https://optunity.readthedocs.io/
5 Results

- Nelder-Mead simplex
- CMA-ES
- Tree-structured Parzen Estimator
- Sobol sequences

We decided to use the particle swarm optimization solver, since this is the recommended solver according to the Optunity documentation [105]. We determined that in this case it is best to aim at optimizing the precision of the model, since for the system, it is highly important to make fewer false positive predictions, because these predictions would make the deployed system lose money. The tuning process considered a value for max_depth between 5 and 99, a value for min_samples_split between 2 and 19, and a value for min_samples_leaf between 1 and 39. This tuning process took 35 minutes. We validated the tuned model using 100 individual cross-validations, each consisting of 4054 test data points. The model scored an average precision of 70.83%. Before tuning, the model had an average precision of only 66.15%. To be able to compare this result, we implemented a grid search solver that for the same decision tree model tested every combination of values for max_depth, min_samples_split and min_samples_leaf that were also considered by the previous Optunity optimization process. This grid search took 6.5 hours. The resulting tuned model had an average precision of 70.74% for 100 individual cross-validations.

We repeated the same experiment with a scikit-learn support vector machine model. The Optunity optimization process considered different values between 0 and 10 for the hyper-parameter C and different values between 0 and 1 for the hyper-parameter gamma. This tuning process took 78 minutes. The tuned model scored an average precision of 70.21% for 100 individual cross-validations. Prior to tuning, the model had a precision of 67.84%. To compare this result, we implemented another grid search solver that would test 81 combinations of different values for C between 1 and 10 and values for gamma between 0.1 and 1. This Grid Search tuning process took 62 minutes. The tuned model had an average precision of 70.25% for 100 individual cross-validations, thereby achieving a better tuning result in a shorter period of time.

The one downside we experienced using Optunity was that it is more difficult to estimate how much time the tuning process will take. This is due to the fact that Optunity does not show the current progress. Although it is possible to implement a counter variable that counts how many iterations have already been performed, since Optunity uses more complex algorithms, predicting how many additional iterations will be performed is highly difficult. It is therefore possible that Optunity might be used in a too fine-grained manner, resulting in a longer computing time than expected.

Analysis

Simm et al. have already concluded that the use of Optunity enables “easy integration of sophisticated tuning strategies” [106]. In our case, the use of Optunity allowed us to integrate a complex particle swarm optimization strategy in less than 5 minutes. The potential speed-up achieved by using Optunity simplifies the development process, and if Optunity is incorporated into the deployed system, such a speed-up increases the system’s performance. Furthermore, the higher precision achieved by using Optunity to tune the decision tree model would have improved the system’s effectiveness if we had integrated this model into the deployed system. This proves that Optunity can be a highly helpful tool in some cases. On the other hand, the two tuning processes for the support vector machine model have shown that there are some cases where the use of Optunity
5.3.6 Matplotlib

Application

Matplotlib\textsuperscript{12} is a Python data visualization library. We used it several times during the feature selection process. The first use of Matplotlib was to automatically create bar graphs of the calculated feature importance scores to identify the features that negatively impact the model. The second use was to create candlestick charts of the stock data and charts of the stock’s volume for the false predictions made by the model. By analyzing these charts, we were able to identify patterns that are likely to indicate an upward stock movement. We then used these new insights to improve the corresponding model. This increased the model’s precision by 4%.

Analysis

To develop better machine learning models, it is critical to understand the data [110]. In our case, we managed to improve model quality by analyzing Matplotlib graphs of stock data. Especially since the feature selection process was a bigger challenge than expected, a tool like Matplotlib was a tremendous help. However, there are several alternatives to Matplotlib, such as Plotly\textsuperscript{13} or seaborn\textsuperscript{14}. All three of these tools offer the main plotting capabilities [111], therefore all three are

\textsuperscript{10}\url{http://hyperopt.github.io/hyperopt}
\textsuperscript{11}\url{https://optuna.org}
\textsuperscript{12}\url{https://matplotlib.org/}
\textsuperscript{13}\url{https://plotly.com}
\textsuperscript{14}\url{https://seaborn.pydata.org/}
effective tools for most use cases. Nevertheless, if one wants to create a less common type of chart, such as in our case candlestick charts, we recommend researching which plotting tool is best suited for this specific use case.

5.3.7 Git

Application

Due to the fact that the models we trained with daily stock data showed much better results than the models trained with intraday data, we decided to completely focus on the models using daily data. This meant that we had no use for the 11.2 gigabytes of intraday data. Since the daily data had a size of 635 megabytes, almost 95% of the data did not serve a purpose. Especially when working in a team in which multiple developers are required to have the corresponding data on their computers, having that much unnecessary data will be a big inconvenience. This raised the question of how to handle this unnecessary data. Since we were using a Git\textsuperscript{15} repository for version control, we were able to create several branches for each type of data, thus each branch only contained the relevant data.

Analysis

Using Git as a version control tool is already common practice, but due to the complexity and amount of data needed to develop machine learning systems, the use of such tools becomes even more important during such a development process. Using different branches depending on what data is used improved the project’s structure, and thereby simplified the development since each branch only contained one type of stock data. However, although Git is able to track source code changes, training data on Git is generally not versioned\textsuperscript{112}. In our case, we relied on retaining several folders for each data version, such as data-daily-processed, data-daily-unprocessed and data-daily-transformed. This made data management more complex. Alternatively, additional data versioning tools, such as Neptune\textsuperscript{16}[113] or Pachyderm\textsuperscript{17}[114] can be used. Furthermore, instead of using Git at all, one can use a version control system specifically built for machine learning projects, such as DVC\textsuperscript{18}[115]. In conclusion, although Git was a helpful tool during the development process, there are more effective version control tools for developing machine learning systems. Therefore, especially when working with complex or large amounts of data, we recommend considering using different version control tools or using special data versioning tools in addition to using Git.

\textsuperscript{15}https://git-scm.com
\textsuperscript{16}https://docs.neptune.ai/
\textsuperscript{17}https://www.pachyderm.com/
\textsuperscript{18}https://dvc.org/
6 Conclusion

This last chapter summarizes the conducted case study and its contributions. In addition, we discuss the limitations of these contributions. Lastly, we present future research possibilities in the AI engineering field and discuss the impact of this study.

6.1 Summary

By conducting a case study, we collected insights and evidence on the effectiveness of applying ten engineering practices during the development of an AI-based system. We analyzed the effects of each practice and, in some cases, proposed ways to improve those practices, such as using efficient strategies for cross-validation. For each practice, we have shown that its application can improve the development process by enhancing the software quality of the resulting system or simplifying the development. However, we have also shown that, in some cases, applying these practices is less effective. For example, automating hyper-parameter optimization if only a few different configurations are considered and the validation time is extremely short, or using cross-validation makes only a minor difference for certain models.

Furthermore, we gathered insights on possible challenges during such a development process and proposed solutions to overcome them. We identified two challenges for which we did not find any existing literature that already presents them. The first challenge is deciding on the optimal design to incorporate machine learning functionality. This challenge occurred due to the vast number of possible investment strategies, creating many different use cases for machine learning functionality. Although this is a highly domain-specific challenge, we expect that there are many similar development processes with such high flexibility, resulting in the same challenge. The second challenge concerns the available data for the deployed system. In our case, the API planned to be used by the deployed system to retrieve stock data changed its API limits during the development process, thereby limiting the available data for the deployed system. Although such sudden API changes that occur during development might be rare, there are other cases where such a challenge can occur. For example, autonomous cars suffer from the problem that the images they detect and use to make predictions have a lower quality in wet conditions [116]. Nevertheless, it is still desired that the deployed model is capable of making accurate predictions even with only low-quality input data available.

Lastly, we collected experience in using several proposed tools to develop machine learning systems. We analyzed how helpful these tools are and how they affect the resulting software. For each tool except Pandas, we have proven that it can be helpful during such a development process. We demonstrated that Pandas is not a useful tool in some development processes since, for example, NumPy is a better choice in some cases. Nonetheless, we assume that in many cases, Pandas can be a helpful tool. Therefore, due to the high amount of uncertainty during such a development, we recommend not committing to a single data analysis and manipulation tool, and instead one
should learn how to use the most proposed tools and select the most suited tool based on the circumstances. Furthermore, we have pointed out that for the tools Optunity and Git, there are several alternatives that are even more helpful during such a development process. In addition, we have shown that in some cases, by implementing a machine learning algorithm instead of using a tool such as scikit-learn, better results can be achieved due to higher flexibility. However, on the other hand, using a scikit-learn model massively simplifies the development. Thus, we still recommend considering using scikit-learn since we assume that, in most cases, the flexibility offered by scikit-learn is sufficient.

6.2 Limitations

Since in this case study we only analyzed a single case, the general validity cannot be assured. We are certain that in different development processes, the effects of the practices and tools analyzed in this study vary. Nonetheless, we anticipate that the practices proposed in this study will also be effective in most other cases. Furthermore, we assume that in other development processes, other challenges might occur that were not covered in this study. For example, development processes with larger teams may experience additional communication challenges. Nevertheless, our work can be helpful in forecasting the challenges that might occur during an AI development process, enabling development teams to prepare for them beforehand or even avoid them completely. In addition, we expect that especially the corresponding domain of a development process affects what practices and tools are effective and what challenges occur. For example, in less complicated domains, it might be unnecessary to collaborate with a domain expert. Lastly, the perceived effects during this development process may be subject to personal bias. As a result, more research is needed to collect data in different domains and from more developers.

6.3 Outlook

A variety of additional research is possible in the field of AI engineering. So far, little data has been collected on many proposed practices and tools. In addition, more and more tools are released to support the development of machine learning systems, resulting in many new research possibilities. Nonetheless, we hope that our study improves the way machine learning systems are developed. We have shown that applying best practices and tools can increase software quality and simplify the development process. Therefore, we expect that our work can be helpful advice for development teams, especially since some of the practices we propose are rarely applied during AI development processes, such as testing the feature extraction code and automating hyper-parameter optimization and model selection [11].

This additionally raises the question of why many proposed AI engineering practices are not adopted during development processes. Such a question could be subject to future research to increase the adoption rates of these practices. Furthermore, we expect that the experiences collected during our development process can be used to apply those proposed practices more effectively. Lastly, we hope that the insights accumulated on possible challenges during such a development will assist developers in overcoming or even avoiding them.
Bibliography


Bibliography


List of Figures

2.1 Candlestick-chart of the Tesla stock ........................................ 11
2.2 OHLC candlesticks ............................................................. 11
4.1 Flowchart of the planned development process .......................... 16
4.2 Business Process Model and Notation diagram of the system’s workflow .......................... 17
5.1 Confusion matrix ................................................................... 26
5.2 Comparison between the scored precision when using cross-validation and the scored precision when not using cross-validation for k-nearest-neighbor models ......................................................... 29
5.3 Training script ...................................................................... 31
5.4 Snippet of the received OHLC stock data for Nutex Health Inc. .......................................................... 40
5.5 12th received CSV file of the Liberty Sirius XM Group Series B stock ......................................................... 42
5.6 Snippet of the Tesla stock daily file .......................................... 44
5.7 Feature importance chart ......................................................... 45
5.8 Candlestick chart of the NetEase stock .................................. 45
5.9 Volume chart of the NetEase stock .......................................... 46
List of Tables

4.1 Proposed Practices .................................................. 20
4.2 Effective moments for each practice .......................... 22
5.1 Challenges occurred during the case study ................. 48
Artifacts

1. Source code of the developed autonomous trading system, DOI: 10.5281/zenodo.7121357
2. Source code of the data retrieval script, DOI: 10.5281/zenodo.7130025
3. During the study accumulated field notes on applying practices and tools, DOI: 10.5281/zenodo.7121411
4. During the study accumulated field notes on occurred challenges and applied solutions to overcome them, DOI: 10.5281/zenodo.7121410