Exploring and researching the possibilities of the tools of AI for TECHNIA

Daniel Jonsson — danne_jonsson@hotmail.com — Linkedin.com/in/Daniel-Jonsson-0

TECHNIA Kista Stockholm — Week 29 - 33, 2019

1 Introduction

This report is written by me, Daniel Jonsson, as a part of the summer work being concluded at TECHNIA at Kista, Stockholm. I worked with the TECHNIA Value Components-team (TVC) and their part of the TECHNIA-system in particular.

This work was to investigate the possibilities of implementing the powerful tools of AI and machine learning into TECHNIA's already existing parts. Focus of the work which was agreed upon by both parties was put on personalization. This to ease the use and navigation of the programs TECHNIA are developing for the user. Several other possible applications were also researched and investigated. During the work much focus was put on what exists right now, what could be done (both now and in the future) and what it would take for this to be possible. This instead of just coding the final implementations as they would be useless without proper preparatory work and research. This is due to this area of work being in such an early stage for TECHNIA as well as the scope of the work.

For Technia to create a functional system, several steps are needed. First having the logistics fixed — such as where the data is collected and what that is saved — is required. Then comes the most work consuming and vital part because you need to figure out all the data that needs to be collected in order to achieve all the functionality in the future. In most cases more is better than less, not just in the quantity but also different kinds of data. This would require input from people experienced with the system as well as what people want to utilize the data for. Then the data can be collected, which usually takes months for it to be sufficient. Later a actual model can be trained, which requires many models tested and evaluated. This could then be implemented in a appropriate way for the users and further work needs to be taken to monitor the result and tweak if needed. All these points are thoroughly discussed in this paper, with other papers, code examples and data extractions in order for the result of this work to be concrete.

How data collection could be changed and work in the future is discussed with concrete suggestions on what to collect, as well as where the specific information could be needed. The work was done three stages; analyzing and researching the tools at TECHNIA and similar work, theorizing and testing what is possible and how it could be done and at last summarizing for the work to be of use in the future.

The report includes an introduction to AI and machine learning. This is followed by a thorough discussion and analysis on what things could be implemented with what is being collected right now in terms of logs and other data, and what could be done in future work and what steps would be needed. Programs implemented during the work are discussed and added to the appendix, as well as in a linked GitHub account[1]. The report then presents and discuss other related work to this area and how this could be linked to applications for TECHNIA. The report finishes with a short conclusion.

If questions arise about my work — either this report, some program here or on GitHub — don't be afraid to contact me on my mail in the beginning of the paper or on LinkedIn. It is also worth mentioning one more time that all material as well as similar work is on the GitHub. Most images in this report are also vector based, meaning you should be able to zoom in to see more details (not the attached images of course).

Keywords:*TECHNIA Value Component (TVC), Product Lifecycle Management (PLM), AI, Machine Learning, User Oriented Machine Learning, Features, Classification, Data Collection, Data Filtering*

2 Introduction to Artificial Intelligence

Artificial intelligence (AI) is a diverse field, held together by common goals and similar methods. The general aim is to improve performance on some task, and the general approach involves using a machine to find and exploit regularities in training data, i.e. learning using repetitive learning and discovery to — at an unprecedented speed to a human — gain knowledge in terms of experience. AI is swiftly rising in popularity and capability in almost all conceivable fields - from helping a farmer predict weather to maximize crop yield to a stock-broker on wall street using AI to anticipate stock value in order to make further profit.

An AI system could also be as simple as a floor vacuum cleaner that moves around in your house, it does not have to be learning through its lifespan, which is why AI is such a broad area where every artificial system can be placed if it only has some kind of intelligence, hence the name. There are three main types of learning algorithms; *Supervised learning*, *unsupervised learning* and *reinforcement learning*. I wont go through these in this paper because there are such good sources on the web[2][3]. This is done throughout this chapter, in order to save time for me and the reader when there are such good information on the web already (and yes, Wikipedia is a reliable source 99% of the time for these kinds of topics). I also recommend to read the articles referenced in the paper as they are often short and concise. The papers are also of course recommended to read but a more specialized and hard to read.

This rise in popularity is mainly as a result of the the dramatic increases in computation power as well as the massive amount of data that is being created every day[4]. Nowadays, the same amount of data is created in two days as was created from the dawn of time until 2003; which illustrates the current massive transition [**?**]. Many studies state that companies that don't follow on this AI train will in the long run lose, which is why it is vital to at least investigate and evaluate the possibilities.

[1] https://github.com/DanelBanel/technia_summer_work.git

[2] https://en.wikipedia.org/wiki/Machine_learning#Approaches

[3] https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know

There are some key parts of a typical AI project work flow, from data preparation to model build to evaluation and deployment, which is discussed below. These can be introduced to companies like TECHNIA through a so called AI platform, which is discussed in Chapter 3.5.1. The actual data flow in general AI can be illustrated as: A user requests e.g. a site from the server and also sends

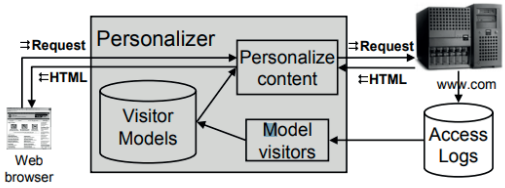


Figure 4.1: **PROTEUS architecture.**

Figure 1: *Flow of data in general AI [6]*

the appropriate data with it (such as username etc.) which returns the page, the AI-system uses the models and the data to personalize the content. This is the actual site that is returned. This data is saved to the access logs which the system can use to update the model.

One key important thing is that I decided early was that it was, due to time and experience limitations more important to find and learn how the data should be collected to be useful, and what new implementations were possible with this data. The implementation with modeling and deployment was, due to it being in such a early state, not as big of a priority.

The area that was to be put in focus that was agreed upon both parties was personalization, and with that we meant to personalize the tools that are sold to TECHNIA's customers in order for their users to more easily and effectively use the tools i.e. develop more personalized and better customer experiences. Therefore *User Oriented Machine Learning* is a suitable term for this work. An AI system or platform is often able to be implemented rather fluently into existing system if this work is done separately, as discussed in Chapter 3.2.4. When collecting the correct data tweaking and implementing changes can be done without much changes in the current system, enabling a smooth transition to the new functionality.

2.1 Data collection

For every system desiring to utilize any kind of artificial intelligence, one thing is essential for the system to be functional; data. Furthermore, even if experienced professionals implement a system for a customer it possibly won't work as expected due to the lack of data that is *good* data, and **lots** of it. You could in most cases say that a system implemented by a beginner who uses Google to find code examples will certainly outperform a system implemented by professionals if it just has significantly more and better data. This is because more good data leads to a more general model, which will lead to more truthful predictions. With good data I mean data that can be used to differentiate users and the behaviour these users have, which is why it is important to look at what data that is possible to gather, and what data that is needed for the specifics of the system sought, which can be derived by filtration and classification.
The concept data mining is vital for this. Data mining is the tool used in order to extract patterns or information out of the (hopefully) mountains of collected data. This data can then be used to detect characteristics and patterns in the data which can help you with e.g. predicting a customer clicks a link or customer retention, a customer leaving (which is discussed more in Chapter 3.6).
How data usually is collected and used for modelling in a system — as well as how I propose TECHNIA could implement this — is discussed more in depth in Chapter 3.2.4

There was also expressed interest from TECHNIA in the more finite area of AI, data analysis. This was to be able to find patterns and what their collected data actually showed, which was not done (from my understanding) before. This would enable not just the desired ultimate goal, of having a system that automatically makes the appropriate changes, but also program changes done manually in the system. This is — in more cases than actually many believe — a more efficient and reliable way of making changes in the system, which is discussed in Chapter 3.5. Therefore several programs of code were written to compile the available data in order to analyze and fit models. These are presented and explained in Chapter 3.1.3. More discussion about data analysis and the things it can enable is discussed in Chapter 3.6.1.

2.1.1 But how much data [is needed for what]?

It also sometimes underestimated actually how much data is needed for a different applications. For the Hotel Recommendation system written by Susan Li she used available data from hotels around the world and used only 1% of this for the system to be more local, but this still equalled circa one million training samples and 2.50 million entries in the test data set [11]. Example for Tesla, the company with arguably most prominent and advanced self driving system in the world, **much more** data is required. For achieving no more than level 2 autonomous driving (not to undervalue their work, but its still a long way left) out of 5, they have used 1 billion miles driven collecting thousands, if not even tens of thousands different parameters for every single mile, to train their deep learning network to be able to recognize other vehicles and traffic signs and more [17]. Even though the applications I suggest in this paper won't require (but as said, more data is generally better) nearly this much data, but it still worth to put into perspective. This also means that when this data is available, a well-equipped computer is needed for training and computation, but that is a topic for other work.

This much data is not collected today at TECHNIA (or at least made available). The number of users is sufficient and a lot of data is collected (I had $2.3M \cdot 3 \cdot 5$ entries logged by the server available) but as later discussed, not containing the actual information that is needed to make some implementations. You might also think that the gathering of data is enough but it is the opposite. In every AI projects, classifying and labeling data sets takes most of our time , especially data sets accurate enough to reflect a realistic vision of the market/world [8].

2.2 Modelling

When a system is making a prediction it is using its trained model, which order the probabilities of the next thing that will occur i.e. the classification it makes, from what it has learned. This model can be created and designed in thousand of different ways, but some common examples are neural networks that uses layers of neutrons with weights (example discussed in 3.1.3) and biases[4] (which can be seen in Figure 2), tree structures that are built with leaves and branches and determines its classification by traversing the tree[5],

[4] https://en.wikipedia.org/wiki/Artificial_neural_network

[5] https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248

which can be seen in Figure 3.

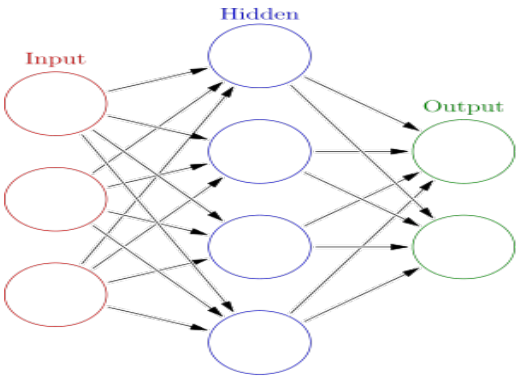


Figure 2: *Example of a neural network*

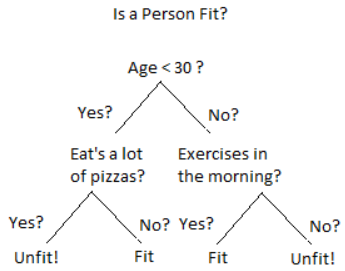


Figure 3: *Example of a classification tree*

For image classification in Deep learning the most common is Convolutional Neural Networks (CNN) which uses a filter to "scan" a image, then multiple layers are use to determine e.g. what number the image represents[6]. An example can be shown in Figure 4 and the code for a similar program and network classifying hand drawn numbers is discussed in Chapter 3.1.3 and attached in B and on GitHub. What model to use when is discussed in Chapter 3.3.

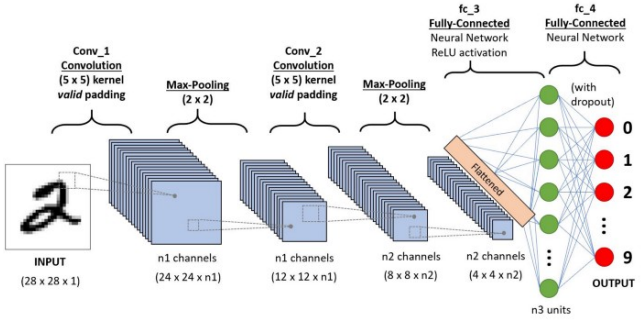


Figure 4: *Example of a convolutional neural network*

2.2.1 Training

The first and most important step of creating the model is the training. It is here that you create the foundation for the whole system, and as with building houses; A great house needs a even greater foundation. As discussed earlier in Chapter 2.1 more training data is always better. The datasets often consist of pairs of an input vector and the corresponding output vector, which is commonly denoted as the label [2]. This is the actual dataset that we use to train the model (weights and biases in the case of Neural Network), which are tweaked according to the system's prediction and the corresponding output label. The model *sees* and *learns* from this data [16], by giving more weight to nodes that lead to a faulty prediction and decreasing the weight for correct ones (or the other way around if using a maximization problem instead of minimization). Because of this, it is important that this data is collected in a way (often either manually or semi-automatic) so that the input is associated with the expected output

When creating your model you often take the majority of data as training data, and split the other part in to the validation- and testing set, as can be seen in Figure 5. How exactly you divide the data into the different sets depend on what model is used and what data is available, but one common partition is 50/25/25%.

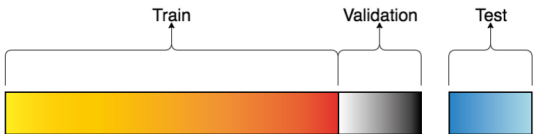


Figure 5: *The relative splitting of data into the different datasets*

In order to illustrate how the data may look and what the modelling process actually does I have constructed a simple example with a small data set. With only 201 data entries(which in almost all cases an extremely small, insufficient amount of entries) I want to predict what sex the crabs are from the other features of the crab. Two different data entries may look like this:

Table 1: *Two examples of data entries in the crab data*

species	sex	index	FL	RW	CL
Blue	Male	1	8.1	6.7	16.1
Orange	Female	47	23.1	20.2	46.2

where RW stands for rear width, CL for carapace length. These are the ones that has been deemed most important by analyzing the data.

By choosing a model, in this case a generalized linear model (GLM), which uses logistic regression and the data sets are only training data (for simplicity). I train the model by trying to find a line that can be put between the different crabs' feature *sex*. This means that the desired output for new data is the crab's sex and the other features are input. Because this is a linear problem a linear model was chosen this is possible. The model will place this line in order to make the biggest distinction between the two classes as possible, which may look like this:

This gives an misclassification rate by 0.035. The color of the dots are the predicted sexes and the green line is to illustrate how close the classification is to linear. Both the code in *R* and result for this is attached in Appendix G.

[6] https://en.wikipedia.org/wiki/Convolutional_neural_network

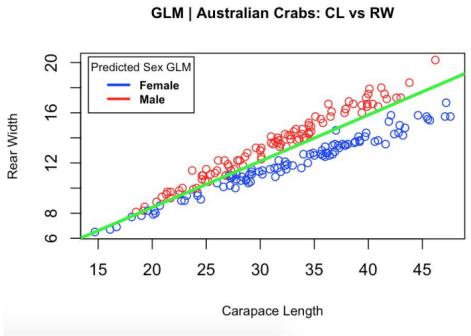


Figure 6: *The classification of the sexes of the crabs from RW and CL.*

2.2.2 Validation

A validation dataset is not needed for all applications, but needed when doing frequent evaluation to fine-tune the parameters and get a better model. Here you just look at your models and select the best performing approach using the validation data. Maybe the models that tested were two completely different model types, or the difference may be number of neutrons in the model. The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is used to select the final model, hence why both a validation dataset and a test dataset is often used. The main goal of the validation set is to check that the model is not trained too much, and therefore *over fitted*. This would imply that it work very good on the training data, but not as good on new data because the model in not a general.

2.2.3 Testing

The actual evaluation of the final model is done last, when the model is complete and you want to know how correct it predicts towards *new*, *never-before-seen* data. Here you simply use the input part of the data as input to the trained model, and evaluate how good it performs against the output part of the data. This is done in order to get an unbiased result of the accuracy of the model.

2.2.4 Using the model

The fact that makes AI such a useful and particularly scalable technique, is the aspect that when the chosen model is trained, this is all you have to save (i.e. the weights and biases of the neutrons for example). The — often huge amount of — training data needed to get a accurate and generalizable model is often hard to collect and filtrate, but afterwards it is neither needed to be saved or computed again (at least for the same model). This is what makes it possible — both in terms of computation and memory usage — to create several different models for various tasks in a system, which often is required for efficiency. When a user changes its behaviour the changed data will affect the model and therefore make appropriate changes. It may need human input to ensure it follows the right path and actually confirming the predictions (and therefore the new data the model is trained on) is correct, but otherwise it will try to follow the users behaviour in order make it dynamic and adaptable. These is not the case for all implementations of model as it makes it more complex, in some cases the trained model is used as it is until a completely new model is created.

If we recall the example about the crabs introduced above, we now can use the test dataset to test our model on new, never-before-seen data. If we get a new data entry we can see where it will end up in in the graph and therefore what it will be classified as. As said, this is extremely simplified.

2.3 Machine learning

ML is a subset of AI of AI based on algorithms that detect patterns – learning how to make predictions and recommendations by processing data and experiences, rather than by receiving explicit programming instruction. The algorithms can adapt in response to new data and experiences to improve predictions and recommendations over time, which is the thing that differentiates machine learning systems from regular AI. Machine learning could be described as the ability of statistical models to develop capabilities and improve their performance over time without the need to follow explicitly programmed instructions.

2.4 Deep Learning

Deep learning (DL), the most significant new frontier of AI and a subset of machine learning, takes advantage of recent advancements in artificial neural networks to process a wider range of data types. It requires significantly less data preprocessing and upfront feature engineering by data scientists and often produces better results than traditional machine learning approaches – although it typically requires a larger amount of data to do so. The name *Deep Learning* is based on that multiple layers of neurons are used, in contrast to only machine learning, that is traversed through by the system in order to make a prediction. A significant advantage of deep learning, especially for the more complex problem types, is that the networks themselves deter-mine the features (such as *width* and *color*), rather than requiring intensive human-led data science and domain-knowledge input. Deep learning models are excellent for image and speech recognition but are difficult or impossible for humans to interpret. You need lots of data to train deep learning models because they learn directly from the data. The more data you can feed them, the more accurate they become.

2.5 Other technologies

Other cognitive technologies, such as machine learning and deep learning, that are worth mentioning are [15]:

- **Computer Vision** — The ability to extract meaning and intent out of visual elements, whether characters (in the case of document digitization), or the categorization of content in images such as faces, objects, scenes, and activities
- **Natural language processing/generation** — The ability to extract or generate meaning and intent from text in a readable, stylistically natural, and grammatically correct form.
- **Speech recognition** — The ability to automatically and accurately recognize and transcribe human speech and vocal emotions.
- **Physical robots** — The broader field of robotics is embracing cognitive technologies to create robots that can work alongside, interact with, assist, or entertain people. Such robots can perform many different tasks in unpredictable environments, often in collaboration with human workers.

Some of these and their area of use are discussed later i Chapter 3.6.

3 Discussion

3.1 What work I have done

What I have done through my 5 weeks at TECHNIA at Kista is hard to completely compile, but can be summarized in three succeeding areas; *analyze and research, theorize and test, summarize*.

Much of the work in the first weeks was to get familiar with the environment, the programs that TECHNIA's develops (actually the target of my work) and especially *analyze* the data available. The programs were explored by simply navigating relevant parts and looking and the different request it did with the help of the *inspect element* feature in Google Chrome. These were then compared and searched for in the data available in order to get a sense of what the data actually contained. The data was available in different forms, such as *.log* and *.json* which meant that it had to be collected to be actually useful and capable to being analyzed, this with various programs written in both Python and R, some presented in Appendix D, F, E and discussed above. These files also contained different kind of data which was important to understand and explore, especially when some were system logs that were possible to modify if needed and some data was logged from the server, and thus not able to be easily alternated by the TVC-team. Therefore different efforts was put into different data. But as explained in the introduction, the data is what makes a model great so a lot of work was put into trying to gather and compile this data. One key important thing is, as mentioned earlier, that I decided early was that it was, due to time and experience limitations more important to find and learn how the data should be collected to be useful, and what new implementations were possible with this data.

The implementation with modeling and deployment was, due to it being in such a early state and with limited experience in building a complete machine learning system, not as big of a priority(there are also a lot of other approaches to do this when having the data, such as various AI platforms as discussed in Chapter 3.5.1, or hiring a consultant for a longer period of time than I had available). How the work was planned and has been carried out can be shown in Figure 7 and 8, respectively.

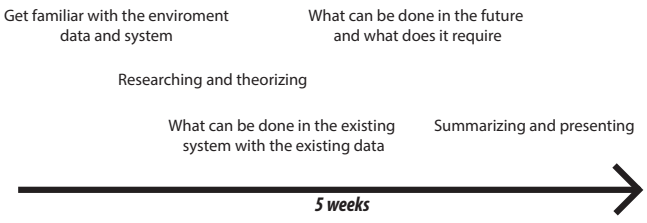


Figure 7: *Timeline for the work being carried out over the five weeks*

As you can see in last figure the flow goes from what exists now, what can exists and how can you use this for future work. The question "what data can exist?" is placed on the line as I try to not include things in this report that are not possible.

When having a somewhat clear idea of what the program and data was, much work (that naturally mostly goes unreported) went into *theorizing* about what work could be done and *researching* what other similar work had been done. A lot of testing was also done, especially on the data. Many papers were read and summarized, some referenced in this report, which gave some ideas to what could be done when applying these techniques to TECHNIA's business. Here I found related works as well as data that was saved that could

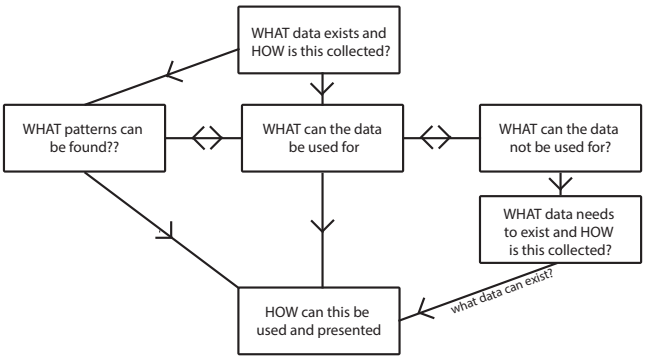


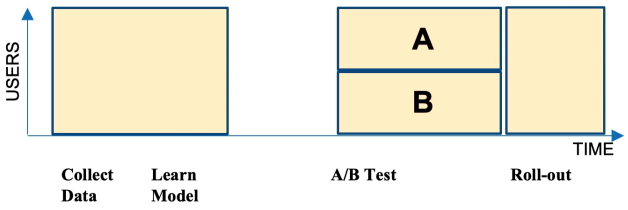
Figure 8: *Flow of the different parts that has been worked on*

be useful for such applications as shortcuts, Netflix-like home page and edit form suggestions(which could lead to prevention of validation errors). All these are discussed thoroughly later. These were things that were on both parties agreed upon before the work begun to be put in focus. Why these things were not fully implemented were due to some restrictions and uncertainties in the data, which possibly needs to be changed in order to get enough data, and good data for it to be workable. Some interesting videos and papers not referenced in this report is listed in Appendix A

In order to make the work done in these 5 weeks to hold for future work, carried out either by me at some other occasion, a consultant or TECHNIA themselves, I needed some way of *summarizing and presenting* what was completed and what is worth continuing working on. Two obvious things are this report and the presentation that is carried out in the last week, but it also needed some results and not just theorizing and reporting. That's why I have written several program that either compile data in different ways, from just changing from e.g. *.log* to *.csv* in order to be more easily handle to small functions that can filter the data in almost all conceivable ways. Programs that look at the data on try to find patterns were also written (here the lack of good data for some applications need to be mentioned) and their result presented in Chapter **??**. The programs may not be used in an actual system but they give some hints to what could be done and how, which is why they are presented in the appendix.

It is worth noting that there has not been perfect communications between the two parties due to some unpreventable conditions such as holidays, which has influenced the work done to some unknown degree, but both parties has tried to work around this as much as possible.

One other thing that is worth considering is that when implementing new things like these mentioned they need, as TECHNIA probably has a lot of experience with, to be introduce in a subtle but accessible way [6]. When introducing an AI-component you often follow this structure



where you collect data (often in months) you learn a model, and find a way of implementing it into the system. Then a popular approach is to do a A/B Test where you introduce the changes to half your user base and not to the other half and watch the result. After some appropriate tweaking you should finally come to the stage where you roll out the change [9]. A more advanced system is called Online Learning, but that is too complex for this scope of work. If you are introducing new things like homepages or shortcuts it has to fulfill some various things;
Actually be helpful, a reasonable trained model has to be working as expected and ease the use of the tool for the user.
A user has be able to reject the changes if it is not wanted for the user e.g. if it is intrusive. This can be done with a small pop-up window introducing the change, and simply explaining that the new implementation can be disabled in a settings menu.
Another thing that should be introduced to every new implementation is the user being able to update the choices presented to her/him if its not what is wanted. From this the model will also learn and lower the probability for these choices, enhancing the trained model. These implementations are added to the figures in the following chapters.

It is important not to elude important parts or rearrange so the navigation becomes more complex and too dynamic. This is especially important for TECHNIA's business where many of their users are experienced in the tools. This is why I chose not to put much effort on similar techniques, and only focused on things that meant additional information to to user in relevant place. Adding a shortcut or a skip-able homepage won't make the already existing content more complex but rather easier. Here it is also worth to mention not add too many options or information, especially in a short time span or on the same place in the tool. This will work in a contradicting way and give the user more information than desired, making it useless.

When not only considering experienced workers, but also ways to ease the learning of the tool for beginners its also important to understand that it is not (now at least) possible to differentiate between the two in TECHNIA's system and therefore not possible to personalize the content for the two different groups.

To summarize: Don't elude or rearrange content, instead add new information in a subtle but accessible way. Don't overflow the users in new information and options, and enable these to be disabled in the settings. If something is catered for experienced users vs newbies is also an important aspect that needs to be considered when weighing if it should be implemented or not. From these aspects I have come up with things that could be further researched for suitable implementations.

3.1.1 Netflix-like home page

A Netflix-like home page, also called a web portal or montage [6] is a page (often presented as the first page for the user) with links and content presented for the viewer for easier accessibility to different sites. A montage is more than "just a list of favorites," for three reasons. One reason is that remote pages may be embedded in the montage, aggregating content from multiple sites. Another is that the content depends on the user's context of browsing (e.g., the topic of recently viewed pages, or the time of day, etc.), so the content is relevant to the task at hand. Third, MONTAGE automatically selects and manages the appropriate content, without the need for explicit input from the user[6].

This can be illustrated by Figure 9, where the system predicts the users most probable sites it will visit next and orders them according to this, where the top left is the most possible. This can be links to a connecting page or shortcuts i.e. links between previously non-linked pages that help lead visitors to their web destinations. They can come in two forms, or a combination of them, links and some box that illustrates what the link leads to. Links give a more compact view, but can be overwhelming if too many are presented for the users. Although shortcuts have a large potential benefit, in practise, achieving this is difficult. If too many shortcuts, or inappropriate shortcuts are presented to a user, then the effort involved to use such shortcuts quickly outweighs their benefits[7]. Content can be hard to implement in a functional way, but is helpful for the user, especially for less experienced ones.

Figure 9: *An example of a home page with shortcuts in form of links and content*

There are several different models that could be used, both simple and complex, for this purpose. The two most common and simple models — and the ones that are being shifted out for smarter nowadays — are *Most Frequently Used* (MFU) and *Most Recently Used* (MRU) [5]. MFU and MRU work as expected by the name, where the most probable site is depended on an ordered list based on how frequently the site is visited or how recently it was visited, respectively. In many cases such simple models are preferred in many ways versus overly complex ones, due to implementation cost but also computation and memory wise. When for example suggesting search terms for a user when searching these may be the models to use, as it is often recent or frequent sites you look for. No need to over design, as these solutions would require one list (per user or a joint one, which would be stupid) and you even could combine the two for a dual suggestion list.

There are also some more complex models that are being used for this purpose. [6] experimented with 4 different models; unconditional, Naive, Markov och Mixture of Markov. The later was the most efficient for their task. Other possibilities are the combination of models that [5] presented in their paper, or the ones tried in my version of the recommendation system C.

Data entries from the log that are useful for example for shortcuts or Netflix-like home page (as these essentially do the same thing, in different formations) are the ones that contain information especially about where the users are navigating and how, for example: It does not need to contain a username if total global modelling

is chosen (which I don't suggest for TECHNIA's purpose) but is needed otherwise. This user could then be linked with other users from their previous behaviour and therefore clusters with models of similar people are created. If the user's context — i.e. what tasks and navigation behaviour the user has right now — is added to the model (which I suggest in some cases, this may not be one because it is in the early stages of doing a task) this is also something that needs to be added to the logs. The username, context and request (what site he visits, sometimes situated in the params-field which makes for confusion) are the data that needs to be sent to where the model is situated for it to compute the shortcuts. Something that I have found need to be addressed for this to be possible (beyond the filtration of unnecessary data) is where the information of a page being visted is presented. Currently I have seen data logs

that do not fully explain what site is actually visited, but maybe a parent-page to that page. This makes it impossible to make a system that suggest any more sites than these, and therefore not really a helpful system. This also depends on how well you know the system, but I am certain it still needs some refinement and improvement.

Additional data is recommended to be logged also, thus data is needed for first and foremost training the models and later updating (also called boost) the model. Some things I have found are useful are also , but not limited to, the current site, previous site(s), user, context, time, day, of active users, region, location, company, search history (long and/or short). Where some are more important than others which depends on the application and the company's needs. How and where this data collection and modelling could be done is discussed in Chapter3.2.4. What is wanted and needed is very individual for each company and most easily set up by someone experienced with the system, customers and company. Note that all this data does not need to be collected exactly at this point, but somewhere suitable in the system e.g. the users role in the company can be saved when creating a profile. Some things that are static with the user, like role, does not need to be sent due to it already being in the model that can be achieved when having the username.

This could be used due to it containing two things, first a request REQUEST and a username USERNAME. First of all, the request could be use to analyze behaviour patterns by looking at how, when and why a site or similar is requested. By looking at previous data entries it is possible to extract a pattern and how the users navigate. As discussed in Chapter 3.4 the user name information could be used to create a specific behaviour for a single user, or using cluster or people to have various models for different roles, location and work tasks. As mentioned earlier, here it is important to label the features appropriately and thereby enabling the model to more easily learn from the data.

Even though I have written a small test program for this (Chapter 3.1.3, Appendix C), which works to some degree it will not work as expected if I would choose to use the data available at TECHNIA right now .It is pretty obvious there is still not enough data that is being collected for this to be a properly functional prototype. Much information about how users navigate in the program and system goes unused simply because enough proper data is being collected. Unfortunately this data is logged by the server and therefore the TVC-team at TECHNIA does not have the same capabilities of changing what is logged, which complicates things to unknown degree.

3.1.2 Shortcuts

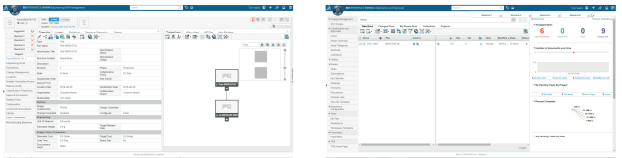
The way I am thinking of shortcuts is presenting small icons (With small icons I suggest e.g. a hyperlink) in the program that will give the user a shortcut to a site (that is not reachable from the current site) if it predicts the user want to end up here, or at least a site on the path to the end goal. This can be illustrated by Figure 10 where if a user often goes to the EBOM+ tab when choosing a part in MyParts by first navigating to the part's first page and then scrolling and clicking the EBOM+ link, a shortcut to jump directly to that site can be presented. The left most image, number 1, shows where the user currently is. By pressing the Part link he/she will be taken to number 2 where it is possible to press EBOM+ in order to get to the last image. If instead the user often follows this routine and the system predicts this would be done, a shortcut could be presented in the first image as is illustrated with red text and [EBOM+] (in order to make it clear it is a shortcut created by the system) it will

Figure 10: *Shortcut presented as a small icon and where it would take the user*

jump directly to the EBOM+ page of the corresponding part. This is just a short example that jumps one page and save one click, but the more pages that are excluded the more clicks and ease is saved for the used.

This could introduce either a more intrusive or annoying part of the system than a home page, or a less one depending on the implementation. If done as presented in the Figure above I believe it would actually be a better option, as it adds new information but not too much for the user to understand or misinterpret (which is important as discussed in Chapter 3.1). I have also perceive that, due to both experience and the learning material available, users in the program often have a pretty clear idea of where they want to navigate and how they do it, but this would just help a little bit on the way. If the shortcut leads to a page further along the trail, then the savings is the number of links skipped (we subtract one because the visitor must still follow a link—the shortcut link). If the shortcut leads elsewhere, then it offers no savings. On the other hand, a home page may be more illustrative and helpful for new users.

There are several other ways this could be implemented. You could have a small menu list placed that contain the most probable sites for the user to navigate right now. This could be placed in various positions on the page, as depicted in Figure 11, where 11a shows the menu in the already existing left menu of the program. This becomes a sub part of the menu in order for it to be clear for the user what is proposed by the system and what is fixed. Figure 11b shows the shortcuts presented in other position of the page. It could also be placed as a menu in the bottom of the page for a accessible, but yet not too intrusive for the user. Of the alternatives I believe the first option, using small icons, is the best as it is while being apparent what content is dynamic.



(a) *Shortcuts in existing menu* **(b)** *Shortcuts in top of page*

Figure 11: *One different positions for a menu of proposed shortcuts*

These examples could also be seen as a home page but is constricted in space on the page to increase accessibility and sites to offer.

Both the simple and the complex models mentioned in Chapter 3.1.1 would work the same here, with the difference being the limited choices you could present in comparison to a full page. If using small icons for the short cuts, only one alternative can be presented, and in a menu no more than say five (amount of choices presented to the viewer is discussed later). This makes it more important for the models to be useful with limited choices, and therefore needed to be more stable and accurate.

This would also require similar data as the home page.

3.1.3 Programs implemented

Several programs were implemented to different degree during the investigation in order to get a more concrete idea of many things. How the data could be compiled in a more accessible way that a log file, what the data consisted of and what you could interpret from it, and how implementations in the program could look as prototypes were some things that looked into. These program are discussed below. Many are also attached in the Appendix and added to the GitHub, and those are completed with comments about output, input and the different functions. Some datasets used are also on GitHub. Some related work and their code examples are also added to the Appendix with a short discussion in Chapter 4. Some interesting data sets are also discussed and referenced. Other similar material from courses around the AI-area as well as other work (e.g. computer vision) are also available on the GitHub.

As mentioned earlier the data more interesting was in the form of *.json* or *.log* files, where the the TVC-team had more control over the files. A program was written to read both these files and saving to a *.csv* file which is in my experience easier to work with when analyzing and computing. Almost all parts of the code very divided into help functions. This was done for future work where the function are easily callable and therefore more useful. The program was also split into 3 parts; one for help functions, one for the .log code and one for .json for easier readability. The program is also split into two additional files *read_mcs_txt.py* and *read_mcs_json.py*. The program should be restarted if run on all available files because of possible memory errors. As there are several inconsistencies in the data available the data was tweaked for it to be possible to use it. This tweaking took very much work and this also means the data is not 100% correct. This may be a sign that the data collection done today needs some updating.

I, as mentioned above, rewrote the Hotel recommendation system by [12] [11] for it to recommend shortcuts for a particular user instead of hotels by looking at the previous sites collected. This was done by using the data program above in order to create a filtered data set in a *.csv* file. This dataset is on Github. The data set is available on Kaggle[7]. A similar program could work in TECHNIA's system, with the difference being the data that is saved and the model trained on.

A simple program for image recognition was also added in Appendix B to be able to see how those differentiate to other machine learning programs. The training data is hand written numbers and the goal for the model is to guess what number is written on never-before-seen data. You could even write a number yourself on a piece of paper (or in like Paint) and try. The program has much training data and the program is simple but it behaves like expected, with some edge cases not being accurate. Here you can see an example of a neural network (or even a CNN) being created starting with the row $model.add(Conv2D(32, kernel\_size = (3,3), activation = "relu", input_shape = input\_shape))$ and the following rows adds even more layers.

Neural network program, introduction

3.2 What work could be done

There were many both existing and new things that utilize the power that could be implemented into TECHNIA's business and system. The ones I have focused on, are as earlier mentioned, are those that ease the use of the tools for the users i.e. *personalization*. Other things that are excluded in this chapter, but still could be looked into as it has substantial benefits for previously tested companies are discussed in Chapter 3.6.

The main reasons for the things discussed in this chapter are not in the previous chapter are; *shortcoming of time*, and *limited resources* which is based on the data that is collected. Lack of time for doing things was not only due to it being five weeks available, but also as discussed earlier how the focus was put on what could be done and what would be needed for it to, instead of ignoring the first steps and jumping right to the implementation.

With the data currently available I don't refer to the amount of data — because much data was accessible — but instead what data was actual usable for what applications. If no data is collected about what data users fill in the forms, then it's impossible to guess what they will write next, which is self explanatory. As it was a parable made by TECHNIA employees, and a reasonable one too, you could try to identify what data Netflix — most probably are because it is not official — is collecting. For every single user in their network they are presumably collecting age, gender, family members (or other people that share the account, which have their own data), what its family members watch, their searches, what their searches lead too and of course the movies/series watched at what time and in what context. One can theorize that for one single user it may know all, but not limited to, the following things about him/her;

- **Name:** Per Person
- **Age:** 27
- **Shares account with:** Hon Person, Han Person
- **Watched movies:**
  Valentine's Day (2010)
  - ⋆ Week_day:
  - ⋆ Time_of_day:
  - ⋆ Watched_before:
  - ⋆ Searched_before:
  - ⋆ Nr_of_searches:
  - ⋆ Play_time:
  - ⋆ Run_time:

  The A-Team (2010)
  - ⋆ Week_day:
  - ⋆ Time_of_day:
  - ⋆ Watched_before:
  - ⋆ Searched_before:
  - ⋆ Nr_of_searches:
  - ⋆ Play_time:
  - ⋆ Run_time:
- **Search history:** Romantic comedy, ... , The A team
- **Search led to:** *Valentine's Day* (2010), ... , *The A-Team* (2010)
- **Rank of search alternative chosen:** 4, ... , 1
- *Probably much more...*

Much of this data is probably also collected either in short or long term, which is weighing what a user is doing and interested in now versus generally. Worth noting that most of this data is most likely encrypted for personal and legal reasons, this was just an interpretation. As could be seen in Chapter 2.2.1 often several more features are collected than actually used for a particular task. This is because different applications need different data.

This data is collected not only because large companies love data in order to either sell it, use it for personalized ads or just saving it for future use, but also because it is needed for making personalized content and experience for the user. Why so much data is needed to just give some movie recommendations may be hard to comprehend, but with that many users it is often needed for a truly functioning personalized system. More data is always better than less data. Because of this some data may be redundant now but

[7] https://www.kaggle.com/c/expedia-hotel-recommendations/data

needed in some years. What data and how much is needed may also depend on how the personalization is modeled. Some examples of personalization that Netflix is using this data is for; Personalized - Page Generation, Promotion, Image Selection, Messaging, Marketing, Life Time Value as well as Personalized Content and finally, Learning Collaborative Search [9]. Thus instead of just one product, a video streaming application, they have millions of products specially made for the single user.

If clustering of users is used, where you assign every user to a cluster with user similar information and behaviour, when more users are in the system more information is needed to make more specified clusters if the clusters get too large. More of this is discussed in Chapter 3.3 and on[8]. It could also be the case, but not as likely, that they create and update a model on the data but only save the model, which is more memory efficient as discussed in Chapter 2.2.4.

Even if I had a complete program day 1, enough data would not be collected under the five weeks for it to be worth just focusing on the implementation. Instead it was investigated on what data would have to be collected on how this would be done.

The two main things that were researched were *edit form suggestion* and *prevention of validation error*, as these seemed to plausibly fit into the current system as well as they correspond with the personalization aspect which was agreed upon.

3.2.1 Edit form suggestion

A decent amount of the work being carried out by users of TECHNIA's system is filling out forms, and they are often fairly similar to each other. Because of this, a system that enables smart auto completion for the user would be useful. This could mean one click to fill in one or multiple boxes, or even the complete form instead of having to write all information time and time again. The goal of a predictive model of form filling is to reduce the amount of time a user spends filling out a form in these scenarios, by offering the user predictions for values of each target field on a form via suggestion lists [5]. More formally, a predictive model of form filling is required to be able to answer queries about the probability of values for a field given the values of other fields, history, and some initial text entered by the user in the target field. This could also depend on the user's role in the company, what kind of form it is and globally/locally.

As discussed in Chapter 3.5 it may not always be worth or even useful to implement AI where its not needed, and instead use simpler models, such as MRU and MFU. But I reckon that here it may be useful to have a more complex model which takes a user's context (e.g. what form is being filled, what form was the previous one and how accurate were the predictions), a user's history (short and long term) and role in the business into consideration when predicting what information will be written, instead of just standard auto-completion (which often just looks at and individual user's recent history). If a complex model actually is needed is something that needs more testing and evaluating. Some models, both simple and complex are briefly described below.

One important aspect to take into consideration is not only what you want to suggest for the user, but also how you do it. As with the number of shortcuts presented for the viewer in the previous chapter, also here it is needed to find a optimal number of choices that is presented to the viewer. Of course is it more probable you suggest the right option with many choices, but too many also overflows the user with information and therefore lowers the usefulness. This exist no gold standard for this, and have to be examined and tested. This may also depend on what kind of field it is and in what context. It was expressed by TECHNIA employees that one click was desired to fill in a field, and therefore scrolling is precluded. Therefore would my guess be 2-3 suggestions for the user, which can be seen in the following figure: In the figure three suggestions

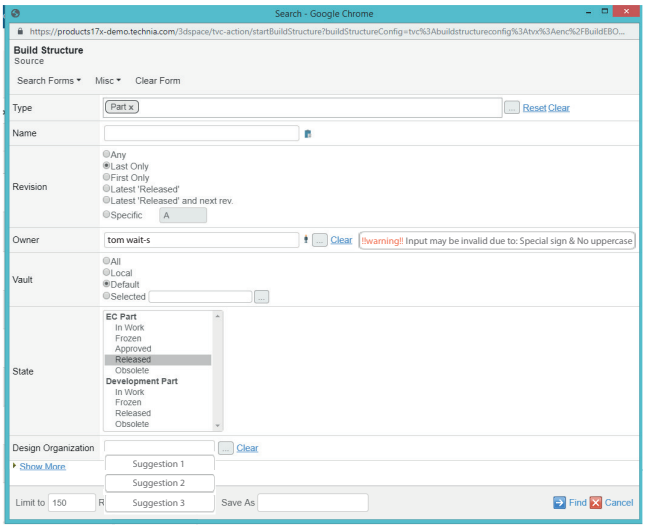


Figure 12: *Suggestions for input in edit form*

are presented for the user in the design organization field. This could also work the way that the number one suggestion is already filled in. A warning box of wrong input is also visible because the system saw that the sign "-" probably should not be there. If doing a mix between models (either complex with local and global modelling or a mix between a simple such as MFU and a complex one) the suggestion box should be separated like this for transparency.

Similarly to when suggesting shortcuts, also here additional data needs to be collected, both in real time to tell the model what to predict from and data needed for training the model. Some things are the type of field, type of form, user, context, input in other fields, input in current field, limitations on the form and the field (especially useful for prevention of validation errors, Chapter 3.2.3), time, day, of active users, region, location, company, search history (long and/or short). All data does not need to saved (e.g. input in current field) and all data does not need to be sent when editing a form (e.g. role in company), as well as some data obviously being more important than other. How and where this could be done is discussed in Chapter3.2.4.

With the correct data collected a similar program to one in Appendix C would be able to make a model with suggestions for a form, using different data (and therefore in some terms a completely different model of course).

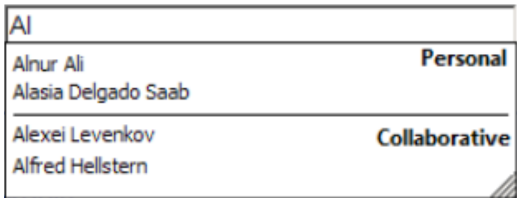


Figure 13: *Showing transparent difference between the two types of generated suggestions*

[8] https://www.geeksforgeeks.org/clustering-in-machine-learning/

3.2.2 Suggestions of other sorts

There are many different aspects and places where a prediction made for a user to do something can help the user tremendously. I will not go into too much detail on these topics as they are many, and some are similar to the earlier discussed. There are also always new things you could do to improve the use of a system, but it also comes to a point of discussion if its worth it, for the work it requires versus what benefit it gives. Some examples I found when navigating the TVC-part of the system are, where most of the pros and cons discussed earlier (e.g. eliding content) are precluded (Most of these can be either recommendations or things the system choose automatically);

- Order and set of columns in various parts of the system
- Order and set of icons in various parts of the system
- What page/tab/column is selected when switching to a new page (similar to Figure 10 but automatically) [9]
- What column you sort after e.g. MyParts [10]
- Recommending what to add to the EBOM+ page depending on the type of part [11][12]
- Color[13], size and position[14] of the different gadgets depending on what gadget and user.
- Recommendations on search terms both before anything is written and adaptive afterwards.

The footnotes show what request are being sent when performing this task. Most of this data is not logged and many parts would need to be extended to be able to collect and use this data (For example the color code when changing color of a gadget is not saved, which makes it impossible to predict this). This is future work.

3.2.3 Prevention of validation errors

When having a reliable system for suggesting to users what to what they will fill in you could use this data for one more thing; preventing that incorrect data is put into the system. Both data that gives errors for the user (if a month is filled in with 13) and that doesn't give errors (small first letter in a name) could be prevented by warning the user if the input does not match the model created from previous data. When having a lot of good data from forms filling (done as discussed above) it could be easy to just look at some different characteristics for each type of input field and look if the input either matches some requirements created by the model but also just looking at the majority (which often is sufficient too a degree). This could be introduced by a popup window warning the user, which makes the system learn if the input is changed after the warning was given (the model probably predicted correctly) or if the warning was ignored and the input still gave no error (the model probably predicted incorrectly) as shown in Figure 12.

[9] `https://products17x-demo.technia.com/3dspace/tvc-action/menuBasedTabPage?menu=tvc:menu:tvx:misc/MyDocsTabs.xml&suiteKey=&widgetId=null`

[10] `https://products17x-demo.technia.com/3dspace/tvc-action/sort?object=tvc-0000000000000002&colName=Desc&asc=true&row=4294967369&scrollY=1&scrollX=0`

[11] `https://products17x-demo.technia.com/3dspace/tvc-action/json/gr_Dashboard/addGadget`

[12] `https://products17x-demo.technia.com/3dspace/tvc-action/json/gr_Dashboard/removeGadget`

[13] `https://products17x-demo.technia.com/3dspace/tvc-action/json/gr_Dashboard/updateGadget`

[14] `https://products17x-demo.technia.com/3dspace/tvc-action/json/gr_Dashboard/rearrangeGadgets`

3.2.4 How and where to collect the data and create the model

There are some proposed steps for implementing AI to a already existing problem and because this is a list made as generic as possible, as well as a sales pitch for AI companies and experts, this list has to be taken with a grain of salt [13].

1. Get Familiar With AI
2. Identify the problems you want AI to solve and prioritize concrete value
3. Acknowledge the Internal Capability Gap
    - *If needed:* Bring in Experts and set up a pilot projekt
    - *Else:* Form a taskforce to integrate data and start working
        - ⋆ *If needed:* Invest in a created solution like AI platform
        - ⋆ *Else:* Write code completely for TECHNIA's needs
4. Start small and Build with balance

This is further discussed in Chapter 3.5.1 Another thing that is important to consider is — when I propose concrete ways of collecting data by alternating what is logged in the current system — this is not the only way to collect data. Alternatively another server can be set up that filters the data and thus only logs exactly what is needed for the model. This would eliminate the filtration that I have been required to do in the current logs, as well as don't mess with the existing parts of the system, computational and memory wise but also security and reliability wise. The concrete propositions of what data is needed mentioned earlier are applicable on this approach too.

As discussed in previous chapters, additional data is required for a more functioning system. Therefore an additional server and database would be able to be tuned during the development of this separate system. By having this externally this is the place where the computation would reasonably be too. By having a separate system with the data and models saved it is both easier to implement the changes untroubled and keeping the current system active at the same time, as you introduce small changes as they are ready. I have personally not full insight in what information the system actually knows about the user (beyond obvious things as User name, time etc.) but in this paper I try to only propose things I think are possible to collect. I have neither put work into how caching works on existing systems and how this would be dealt with, as that is future work. When having the logistics fixed, data needs to be collected in a right way. This would require input from people experienced with the system as well as what people wanted to utilize the data for. Then extensive work would be needed to introduce labels because as previously said, you might also think that the gathering of data is enough but it is the opposite. In every AI projects, classifying and labeling data sets takes most of our time , especially data sets accurate enough to reflect a realistic vision of the market/world [8]. This requires meticulous research and planning before collecting the first data entry, which would be one of the first steps for TECHNIA to take towards an AI solution.

If the approach instead is to use the existing log-capabilities you instead would need to collect the data by extending the internal information in e.g. the *request- params-field*. Such a data entry could include additional fields such as; type of field, type of form etc. This means that this approach is possible to take, but creates a system not separated from the current system which can cause problems.

For an example the complete data flow will work something like this; A user is filling in a form, the system sends the appropriate information to the model. The computation is done and a prediction is sent back. The system reacts to the return information and executes different functions that gives the user a suggest and maybe

changes the xml/java code in order for a warning popup window to be visible (Changing the code in the existing system coherent with the AI-part was done in [6]).

3.3 What model to use where?

As could be seen in the recommendation code in Appendix C we can also see that it is hard to know what model to actually use before trying it out, which is why it is often recommended to look at many different and choosing the one that fits your problem, because they are so different. There are groups of models you could try on a problem available on the internet[15]. What model to use where is a question that is very hard to know the answer if not plenty information comes with it. This is because what type of model to use differs extremely with different applications, and the configuration of that model also depend on several different things. There are some more general models, such as Neural Networks and CNN for images. There are also some tools that fixes fix some things you would otherwise need to test and evaluate yourself, such as pytorch, tensorflow, ONNX. All these are listed in Appendix A with short explanations.

3.4 Global versus local modelling

When modeling you have to consider what approach fits best for what application, either global which means all users' data is used to make the model or local which uses only a group of users' data (or even one individual user). Such things as suggestions in Edit forms may be more individual and what sites are often visited may be a more global pattern. When considering local modelling, clus-

Dimension	Range of values
Scope of target audience	Whole Web Group of related users Individual user
Scope (in time) of personalization	Web session Many site-sessions Single site-session Single page view
Location of user modeling	Client Intermediate proxy Server
Location of personalizer	Client Intermediate proxy Server
Visitor's goal of web interaction	Browsing for long-term interests Browsing for short-term interests Goal-directed browsing ("information sortie")

Figure 14: *Space of personalization [6]*

ters can be used to separate the users into different groups. Here you need to look at the different collected parameters for all the users and find pattern. AI is extremely good at doing this to a degree not possible for people. Not only can it do this fast but also find patterns not thinkable otherwise.

3.5 Where are AI solutions actually helpful and needed?

The available machine learning tools are currently or not even in the future needed for some applications. Sometimes it is simply better to use a MRU or MFU- model. There are also several cases where there is not enough benefit with regards to the work implementing and maintaining a ML-solution, for it to be worth the benefits received. This is a really broad topic that needs much research and depends on the current system, which is also important to take into consideration, especially when weighing what improvements need an machine learning or AI solution first. One thing I suppose can be improved with AI but is not top priority is input suggestion in a log in screen. Here it is just simply more reasonable to use the MRU-model (or MFU).

3.5.1 AI platforms and other ways of implementation

As a result of the rise in popularity of AI and machine learning, several companies work with providing a AI platform to companies that due not have the competence or time to implement the entire system by themselves. You can then either buy or rent a service which functions as the separate AI-part of the system as explained in Chapter 3.2.4. You research what data you want to collect, collect this data then just call the service where ever it is appropriate and receive an response according to the model chosen. This would still mean most of the work but transfer the most AI-specific part of the system to a more reliable and knowledgeable source. Some cons with this approach is that it may be unneccessarily expenseive for a company with enough knowledge to implement this themselves, or at least research and learn how to do it. This approach may not cover all aspired functionality either. Peltarion is such a platform that offers an operational AI platform is an end-to-end, cloud-based platform with an intuitive graphical interface[4]. Runway ML is a similar tool, linked in Appendix A. These platform may or may not be suitable or scalable for TECHNIA, this is to be investigated further, but many more exist.

Another approach is to hire a machine learning expert as a consultant for this task. They would be able to cover all parts of the implementations, from data collection to deployment, but still need insight and consultation from the employees more experienced with the tool. A lot of freelancers and other people working with these kinds of things should be available.

The last and most obvious approach would just to do the complete implementation yourselves. Form a small team that will work on this, either part time in parallel with the ordinary work or full time. This would, as all the other approaches, require consultation from most of the workers for the system to be functional on many parts of the system.

3.6 Other applications

[15].

Beyond the possible implementations as discussed previously there are several more things AI and machine learning could be used for in TECHNIA's system. Some things that were brainstormed in the beginning of the work were and some ideas that arose during the work was the following;

- **Personalization and recommendations** as discussed in Chapter 3.1.2, 3.1.1, 3.2.2.
- **Give recommendations in cases where user otherwise might get validation errors** as discussed in Chapter 3.2.3.
- **Predict locks/crashes** by trying to find patterns in what causes failures in the system. *Where how* and *when* are things that possibly could be found by analyzing the data. This is how the system could identify a problem close to occurring and do the appropriate quick fix for avoiding it. Here I advise Technia
- **Identify problematic user patterns/recurring searches**
- **Image based recognition** (Den önskade funktionalitet är troligvis inte idag möjlig, tex för att identifiera 10 siffror

[15] https://towardsdatascience.com/which-machine-learning-model-to-use-db5fdf37f3dd

behövdes 60000 bilder och då funkar den inte ens i närheten av felfritt)

- **Active work hours? Activity per component?**
- **JIRA data** (stories sprints etc)
- **Customer Relationship Management (CRM)**
- Sales

As discussed earlier, the two first points where the focus of this work. The others are possible and could also be useful to different degree. The ordering of the items in the table represent how I personally rank them in value for TECHNIA, with regards to time, knowledge and resources needed. The ranking done in the early stages of the work, before getting to know the system and data, was different.

3.6.1 Data analysis

There are several other ways to use to tools of AI than just machine learning systems. As is often desired by companies is to be able to analyze the data, hopefully enabling patterns and finding, and this strengths and weaknesses. This information could then be used for changes in the system done manually by e.g. changing the arrangement of a site or deleting a redundant page. There are several tools available for this on the web, as well as some added to the GitHub. This can enable to make changes where AI solutions as unnecessarily complex or wasted. This also requires a lot of data and seperated data for it to be as useful as possible. Outliers can easily be found by looking at errors and for example what time they occured versus request. Kabana[16] is a tool used by TECHNIA that can do some data analytics.

4 Related works

It is both hard and easy to find good sources and information of related work, or even machine learning in general. It is a extremely popular area which is only rising with more and more techniques being researched which makes it easy to read about what others did, but not how they actually did it in detail. How the code is written is a common misconception in machine learning, or AI in general. A paper may say; ”we used the X data set and trained a Markov model using cross validation in order to achieve this”. This could sound complex, but with the tools and resources on the web nowadays this statement could be achieved using less than 10 lines of code, which maybe can be seen in some places in the programs written by me, like the image classification example[3] or the recommendation code example in Appendix C where 90-95% of the code is to set up the data properly. This makes it both easy to write code if you know what you want, but also makes it harder to follow papers when no code is available to try and revamp their work. With limited experience in building a complete machine learning system, and the limited time span, more emphasize and work was put on what could be done and what steps were needed for this to be possible.
Many papers were old, up to 20 years old, with outdated models and solutions and therefore too old to follow unconditionally. Though, these were read and referenced in this paper due to the techniques were the ones being requested. They gave some explanation (unfortunately only in text and no code examples) on what they wanted, how they did it and what the result was. The reason for why many old research papers were referenced was also because some things that were to be researched actually were a problem that long ago. By that I mean that these things were popular back then, but now follow the same implementation but only with better models. This meant unfortunately that newer papers with examples of models they used for these techniques were not found. But, the main difference should more often than not only be the kind of model trained.

A Machine Learning Approach to Web Personalization by Corin R. Anderson presents a study of three different machine learning solutions to the problem of presenting a personalized experience for a user[6]. The techniques introduced were a montage, a shortcut solution and mix between the two.

They also state that ”we are also more concerned with personalizing content for an individual visitor, as opposed to making adaptations generally useful for a site’s entire web audience, although our approach is relevant to both targets”, which also match this work.

They concluded that eliding, erasing and moving content in the desire to filter a users content to the content that is desired often is unwanted. Instead, adding shortcut links to a page is frequently useful for visitors, helping them reach their destinations more quickly, and rarely decreases the usability of a page. If you chose a model that changes the content anyway, one way to make it more accessible is to only do this in specified areas, know by Corin R. Anderson as content blocks.

This paper is especially useful to read when researching the implementation of shortcuts (Chapter 3.1.1, 3.1.2). Some pseudo code from the paper is rewritten for TECHNIA’s purpose and added to Appendix **??**. A lot of emphasize put on the user’s context is mostly precluded because regular patterns and behaviour was found more interesting for TECHNIA. Neither scroll behaviour is not as important initially as from my research most pages’ content fit into one page.

[6] also talk about how problems occurs when trying to predict behaviors on seldom- or never-before-visited pages, and how they used a clustering approach to solve it. This problem is called the cold start problem. The cold start problem is a well known and well researched problem for recommender systems, where system is not able to recommend items to users. due to three different situation i.e. for new users, for new products and for new websites. This could be solved with users that are behaving similarly on similar sites are grouped together in clusters which then can be used to make a reliable prediction on how the cluster will behave, even though just this particular user haven’t visited this site before[17] [18].

Susan Li writes a short article and provides a program[19] on how to solve this problem with so called content-based filtering[12]. Her implemented system first uses the metadata of new products when creating recommendations, while visitor action is secondary for a certain period of time. The systems is then able recommend a product to a user based upon the category and description of the product. Content-based recommendation systems may be used in a variety of domains ranging from recommending web pages, news articles, restaurants, television programs, and hotels. The advantage of content-based filtering is that it does not have a cold-start problem. If you just start out a new website, or any new products can be recommended right away. This is why this is relevant to TECHNIA and their work. By labeling different parts and sites in the system both new features are easier to find for users, but also navigating in the system as a new user is easier. By removing stop words and looking at the most popular bigrams and trigrams of 152

[16] https://www.elastic.co/products/kibana

[17] https://en.wikipedia.org/wiki/Cold_start_(computing)

[18] https://en.wikipedia.org/wiki/Recommender_system

[19] https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Hotel%20recommendation.ipynb?

hotel descriptions it was possible to sort these hotels and therefore propose to the user without the need of proper user data. As presented in Chapter 3.1.3 I created a similar program, which is more reasonable to discuss there.

One paper that went into heavy details about predictive form filling was written by Alnur Ali and Christopher Meek [5]. They performed a study on how machine learning could give suggestions to what the user would want to edit, which gave a favourable result even though some limitations was put on the process. It was assumed that the user will only fill in the form from top to bottom and only text fields were allowed, but this could also be applied to drop down menus and more.

The most common approach to ease a user when filling out a form is to generate a list of likely values for the field the user is attempting to fill out, a so called suggestion list. Something that can make this more advanced is to take into consideration the other fields in the form, so that you don't exclude the naturally occurring dependencies between fields. Such as that between a field intended to capture the name of the form filler and a field intended to capture the filler's address.

This is a good paper to look at when implementing edit form suggestions, suggestion of other sorts but also prevention of validation errors (Chapter 3.2.1, 3.2.2 and 3.2.3). The

5 Conclusion

References

[1] Technia documentation, mcad optimizer - administration guide. https://products.technia.com/app/docs/tvc-documentation-2019.3.0/tvc/adminguide/mcadoptimizer/index.html.

[2] Training, validation, and test sets. https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets.

[3] What does ai code look like? https://www.quora.com/What-does-AI-code-look-like, https://github.com/mnielsen/neural-networks-and-deep-learning.

[4] Moving from research to reality for deep learning and AI. https://resources.peltarion.com/c/operational-ai-white, 2019.

[5] A. Ali and C. Meek. Predictive models of form filling. 2009.

[6] C. R. Anderson. *A machine learning approach to web personalization*. Citeseer, 2002.

[7] R. Bridle and E. McCreath. Inducing shortcuts on a mobile phone interface. In *Proceedings of the 11th international conference on Intelligent user interfaces*, pages 327–329. ACM, 2006.

[8] A. Gonfalonieri. How to build a data set for your machine learning project. https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-project-5b3b871881ac, 2019.

[9] T. Jebara. Machine learning for personalization - youtube video.

[10] D. Jonsson. Github. https://github.com/DanelBanel/.

[11] S. Li. A machine learning approach — building a hotel recommendation engine. https://towardsdatascience.com/a-machine-learning-approach-building-a-hotel-recomm 2018.

[12] S. Li. Building a content based recommender system for hotels in seattle. https://towardsdatascience.com/building-a-content-based-recommender-system-for-hot 2019.

[13] R. Marvin. 10 steps to adopting artificial intelligence in your business. https://uk.pcmag.com/business/87232/10-steps-to-adopting-artificial-intelligence-in-you 2018.

[14] V. Reavie. Do you know the difference between data analytics and ai machine learning? https://www.forbes.com/sites/forbesagencycouncil/2018/08/01/do-you-know-the-difference-between-data-analytics- 2018.

[15] D. R. . P. Sallomi. Cognitive technologies survey. https://www2.deloitte.com/us/en/pages/deloitte-analytics/articles/cognitive-technology-adoption-survey.html.

[16] T. Shah. About train, validation and test sets in machine learning. https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7, 2017.

[17] Z. Shahan. Tesla autopilot miles soaring. https://cleantechnica.com/2019/01/05/tesla-autopilot-miles-soaring/, 2019.

Appendices

A Appendix A

LIST OF USEFUL VIDEOS AND PAPERS AND TOOLS

A.1 Programs and code

https://github.com/facebookresearch/dlrm/blob/master/dlrm_s_caffe2.py
https://github.com/kailashahirwar/cheatsheets-ai
https://github.com/mnielsen/neural-networks-and-deep-learning

A.2 Papers and articles

https://towardsdatascience.com/@actsusanli — A creator that focuses on machine learning solutions
E-Learning personalization based on hybrid recommendation strategy and learning style identification http://www.sudskavestacenja.com/wp-content/uploads/2014/11/2011-computers-and-education-Boban-Vesin.pdf
Application of data mining techniques in customer relationship management: A literature review and classification http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.6973rep=rep1type=pdf
Search Personalization using Machine Learning https://pdfs.semanticscholar.org/5f5a/5ed03af48842b054e220f8207128ca8bfcdd.pdf
Automatic Web Content Personalization Through Reinforcement Learning https://www.researchgate.net/profile/Silvia$_M irri/publication/294425705_A uto$
$Web - Content - Personalization - Through - Reinforcement - Learning.pdf$
$IntelligentTechniquesforWebPersonalizationhttps://www.researchgate.net/profile/Bamshad_M obasher/publication/227311174_I ntell$
$DataMiningforWebPersonalizationhttps://www.researchgate.net/profile/Bamshad_M obasher/publication/200121028_D ata_M ining_f or$

A.3 Tools

https://peltarion.com/signup — Peltarion tool as discussed in the paper, used as a AI platform
https://runwayml.com/
https://www.ibm.com/analytics/cloud-pak-for-data
https://www.tensorflow.org/overview/?hl=sv
Models (pytorch, ONNX, osvosvosv)

A.4 Videos

https://www.youtube.com/watch?v=bebNv-wSqt4 — Netflix conference about personalizing their content with machine learning. Very interesting to watch.
https://www.youtube.com/watch?v=FWOZmmIUqHg —
https://www.youtube.com/watch?v=32o0DnuRjfg —
https://www.youtube.com/watch?v=u6sahb7Hmog —
https://www.youtube.com/watch?v=7zmClpSZuxo —
https://resources.peltarion.com/c/webinar-how-to-build —

B Appendix B

Code snippet for image classification of numbers using a deep neural network (.ipynb file on GitHub)

C Appendix C

Python code snippet for Rewritten Hotel Recommendation written by me [12] [11](.ipynb file on GitHub)

D Appendix D

Python code for reading the .log data from folders and then writing it to a CSV-file in the correct format

```
'''                  .LOG CODE                '''

```

```
path = "C:\summerwork\_ELK_DATA_LOGS\MCS\srv14220\\1\\" #Path of files
write_path = "C:\Program1\\" #Path to write
filename_log = path + "plmprod-1.access.2018-08-22.log" #Only used when only reading 1 file, use
    createListFromFile
suffix_log = ".log" #To seperate from .log and .json
csv_file = write_path + "test1_log.csv" #Path to outputfile + outputfile name
counter = 1 #Used to only write header for the first file in folder
total = 0 #Total rows of information

'''
Takes a folder of files, writes all .log files to a csv file with the appropriate seperation of the data
'''

fout = open(csv_file, 'w')
filenames =  getFileNamesFromFolder(path)
for filename_log in filenames:
    filename_log = path + filename_log
    #Creates a list to with all the data. Either from one file (createListFromFile) or all files in a
    folder (createListFromFolder)
    data_log = createListFromFile(filename_log, suffix_log)

    #Filter away unnecessary data, by looking at the requests made.
    filter_array = ["/3dspace/collaboration/communication", "dynaTraceMonitor", "/3dspace/tvc-action/lazy
    ", "/3dspace/tvc-action/collabAvatar/", "/3dspace/tvc-action/collabProfilePicture/", "/servlet/
    MatrixXMLServlet", ]
    #print("Length before filtration", len(data_log))

    data_log_filtered = [row for row in data_log if not any(filter_string in row for filter_string in
    filter_array)]
    print("Length after filtration",len(data_log_filtered))
    data_log = data_log_filtered
    total = total + len(data_log)

    #Creates a dictionary from the list for easier access and filtration
    #List of all columns in .log. Taken by backwards engineering from Kabana
    tag_array = ["IP", "Time", "Method", "Request","Params","HTTP-type","Response", "?", "??", "Referer",
     "Useragent", "Username", "???"]
    d_log = createDictFromList(data_log, tag_array)

    #Writes dict_data to a csv file
    counter = WriteDictToOpenCSV(fout,d_log, counter)
print("Total length of csv-file: ", total)
fout.close()
```

E Appendix E

Python code for reading the .json data from folders and then writing it to a CSV-file in the correct format

```
'''                  JSON CODE                        '''

path = "C:\summerwork\_ELK_DATA_LOGS\MCS\srv14220\\1\\" #Path of files
write_path = "C:\Program1\\" #Path to write
filename_json = path + "Castor-3dspace-plmprod.2018-08-22.json"  #Only used when only reading 1 file, use
     createListFromFile
suffix_json = ".json" #To seperate from .log and .json
csv_file = write_path + "test1_json.csv"#Path to outputfile + outputfile name
counter = 1 #Used to only write header for the first file in folder
total = 0 #Total rows of information

'''
Takes a folder of files, writes all .json files to a csv file with the appropriate seperation of the data
'''

fout = open(csv_file, 'w', encoding="utf8")
```

```
fieldnames = ['timeMillis', 'thread', 'level', 'loggerName', 'message', 'thrown', 'endOfBatch', '
    loggerFqcn', 'contextMap']
filenames =  getFileNamesFromFolder(path)
for filename in filenames:
    filename_json = path + filename
    json_list = createListFromFile(filename_json, suffix_json)
    total = total + len(json_list)
    print("Length after filtration",len(json_list))
    if(len(json_list) != 0):
        try:
            writer = csv.DictWriter(fout, fieldnames=fieldnames)
            if counter == 1: # only write header for the first file
                writer.writeheader()
                counter += 1
            for json_object in json_list:
                 writer.writerow(json_object)
        except Exception as e:
            print("Exception", e)
print("Total length of csv-file: ", total)
fout.close()

'''
    Example of what can be done with the different help functions
'''
'''
timeMillis_array = []
thread_array = []
level_array = []
loggerName_array = []
message_array = []
endOfBatch_array = []
loggerFqcn_array = []
contextMap_array = []
json_list = createListFromFile(filename_json, suffix_json)

##Arrays for saving all of the different json-key-information in different arrays
keys = getAllKeys(json_list)

##Fill all array e.g. timeMillis_array with corresponding values
fillAllKeyArrays(json_list)

#Filter the timeMillis_array on a number, calculate Avg min and max on the filtrated data(if filtration
    is done)
timeMillis_filter = 10
timeMillis_array_filtered = filter_timeMillis(timeMillis_array, timeMillis_filter)
print("Length of Timemillis before filtration: %d and after %d" % (len(timeMillis_array), len(timeMillis_
    array_filtered)))
print("TimeMillis: Avg - ", calculateAverageOfArray(timeMillis_array_filtered), " Min - ", min(timeMillis
    _array_filtered), " Max - ", max(timeMillis_array_filtered), " Difference is - ",  max(timeMillis_
    array_filtered)- min(timeMillis_array_filtered))

unique_tags = findAllUniqueTags('level', json_list)
#print("UNIQUE TAGS", unique_tags)

d = findAllUniqueTagsReturnArraysAsDict('level', json_list)
#print("All rows with level:INFO", d['INFO'])

substring = "HM Product Development"
d_info_filtered = filterArrayOnSubString(d['INFO'], substring)
#print("All rows with substring" ,d_info_filtered)
'''
```

F Appendix F

Python code for the help functions used in Appendix E and Appendix D

```
import json
import numpy as np
import os
import re
import csv
from collections import defaultdict
path = "C:\summerwork\_ELK_DATA_LOGS\MCS\srv14220\\1\\"
write_path = "C:\Program1\\"
filename_json = path + "Castor-3dspace-plmprod.2018-08-22.json"
filename_log = path + "plmprod-1.access.2018-08-22.log"
suffix_json = ".json" #Used in createListFromFile
suffix_log = ".log"

'''            Help functions             '''

def createListFromFolder(folder_path, suffix):
    '''
    Creates a list from all rows in the files in a folder (.json or .log)

    Input:
        folder_path: Path to folder with files
        suffix: Read .log or .json files?
    Output:
        list with all unseperated information
    '''
    file_name_array = getFileNamesFromFolder(folder_path)
    complete_data_log = createListFromFile(folder_path+file_name_array[0], suffix)
    for file_idx in range(1,len(file_name_array)):
        addListFromFileName(complete_data_log, folder_path+file_name_array[file_idx], suffix)
    return complete_data_log


def createListFromFile(filename,suffix):
    '''
        Creates a list from all rows in a file (.json or .log)
    Input:
        filename: Path to file + filename
        suffix: Read .log or .json files?
    Output:
        list with all unseperated information
    '''
    array = []
    try:
        with open(filename, 'rt', encoding="utf8") as f:
            if filename.endswith(suffix_log) and suffix == suffix_log:
                array = f.readlines()
                print("Reading .log: ", filename)
            elif filename.endswith(suffix_json) and suffix == suffix_json:
                array = [json.loads(line) for line in f]
                print("Reading .json: ", filename)
    except Exception as e:
        print("Failed to load. May be corrupt file or no ", suffix, "-file. Exception: ", e)
    return array

def addListFromFileName(l, filename, suffix):
    '''
        Extends existing list with new information from new file
    Input:
        l: existing list
        filename: Path to file + filename
        suffix: Read .log or .json files?
    Output:
        extended list
    '''
    return l.extend(createListFromFile(filename, suffix))
```

```
66
67  def getAllKeys(json_object):
68      '''
69          Returns a list with all keys from a json_object
70      Input:
71          json_object: dict
72      Output:
73          A list of all keys in dict
74
75      '''
76      keys = []
77      for key in json_object:
78          keys.append(key)
79      return keys
80
81  def fillAllKeyArrays(json_list):
82      '''
83          Fill all arrays with the appropriate information from the list of dicts
84      Input:
85          json_list: list of json_object
86      Output:
87          -
88      '''
89      timeMillis_array = []
90      thread_array = []
91      level_array = []
92      loggerName_array = []
93      message_array = []
94      endOfBatch_array = []
95      loggerFqcn_array = []
96      contextMap_array = []
97      keys = getAllKeys(json_list)
98      for row_index in range(len(json_list)):
99          timeMillis_array.append(json_list[row_index][keys[0]])
100         thread_array.append(json_list[row_index][keys[1]])
101         level_array.append(json_list[row_index][keys[2]])
102         loggerName_array.append(json_list[row_index][keys[3]])
103         message_array.append(json_list[row_index][keys[4]])
104         endOfBatch_array.append(json_list[row_index][keys[5]])
105         loggerFqcn_array.append(json_list[row_index][keys[6]])
106         contextMap_array.append(json_list[row_index][keys[7]])
107
108
109 def findAllUniqueTags(key, json_list):
110     '''
111         Find unique information for a key, return list
112     Input:
113         key: key searched for
114         json_list: list of json_object
115     Output:
116         List of unique tags (tex. ['ERROR', 'INFO', 'WARN', 'FATAL'] for key = "level")
117     '''
118     filter_array = []
119     unique_tags = []
120     for row_index, row in enumerate(json_list):
121         if not any(filter_string in json_list[row_index][key] for filter_string in filter_array):
122             unique_tags.append(json_list[row_index][key])
123             filter_array.append(json_list[row_index][key])
124     return unique_tags
125
126 def findAllUniqueTagsReturnArraysAsDict(key, json_list):
127     '''
128         Find unique information for a key, return as dict
129    Input:
130         key: key searched for
131         json_list: list of json_object
132     Output:
133         Dict of unique tags
134     '''
135     unique_tags = findAllUniqueTags(key, json_list)
```

```
    d = defaultdict(list)
    for row_index, row in enumerate(json_list):
        for tag in unique_tags:
            if(json_list[row_index][key] == tag):
                d[json_list[row_index][key]].append(row)
    return d

#Filters all element in array on a substring
def filterArrayOnSubString(list_objects, substring):
    '''
        Filter away elements in list that do not contain the substring
    '''
    list_filtered = []
    for d in list_objects:
        string_dict = json.dumps(d)
        if substring in string_dict:
            list_filtered.append(d)
    return list_filtered

def calculateAverageOfArray(array):
    '''
        Calculates average of all elements in array (used for tex. timeMillis), return average
    '''
    if (len(array) == 0):
        return 0
    return sum(array)/len(array)

#Filter timeMillis on a value and make to a new array. Done by converting to numpy-array for speed. Dont
    know if numpy is heavily used at TECHNIA which is why not generally used
def filter_timeMillis(timeMillis_array,  timeMillis_filter = 1534930000000):
    '''
        Filter away elements that do no match the filter-value, return filtered list
    '''
    timeMillis_array_filtered =  np.array(timeMillis_array)[np.array(timeMillis_array) > timeMillis_
    filter]
    return timeMillis_array_filtered

def getFileNamesFromFolder(folder_path):
    '''
        Get a list of all filenames from a folderpath
    '''
    return os.listdir(folder_path)

def filterUserName(string, filter_string):
    '''
        Search if filter_string is in string
    '''
    return re.search(filter_string, string).group(1)

def WriteDictToCSV(csv_file, d):
    '''
        Opens a csv-file and writes the header and all lists of information to that file
    Input:
        csv_file: path to outputfile + outputfile name
        d: new information as dict
    Output:
        -
    '''
    if len(d) != 1:
        try:
            with open(csv_file, 'w') as file:
                 writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
                 writer.writerow(d.keys())
                 for idx in range(len(d[list(d.keys())[0]])):
                     writer.writerow([d[key][idx] for key in d.keys()])
        except Exception as e:
                print("Exception", e)
    return

def WriteDictToOpenCSV(csv_file, d, counter):
```

```
    '''
        Takes a open csv_file and concatenates the new information to that file.
    Input:
        csv_file: open csv_file
        d: new information as dict
        counter: used to only write header-columns for the first file
    Output:
        counter: adds and return so no header is written for the second file
    '''
    if len(d) != 1:
        try:
             writer = csv.writer(csv_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
             if counter == 1:
                 counter += 1
                 writer.writerow(d.keys())
             for idx in range(len(d[list(d.keys())[0]])):
                 writer.writerow([d[key][idx] for key in d.keys()])
        except Exception as e:
                print("Exception", e)
    return counter

def createDictFromList(data_log, tag_array = ["IP", "Time", "Method", "Request","Params","HTTP-type","
    Response", "?", "??", "Referer", "Useragent", "Username", "???"], split_c = "|"):
    '''
        Takes a list and splits it into the appropriate dict-parts using a tag_array with keys and a
    split_character

    Input:
        data_log: List of information
        tag_array: Keys to split into, instansiated by .log files
        split_c: character to split on, instansiated as "|" from .log files
    Output:
        dict: the full dict
    '''
    d_log = defaultdict(list)
    d_log.fromkeys(tag_array)

    for string in data_log:
        string_split = string.split(split_c)
        for tag_idx, tag in enumerate(tag_array):
            d_log[tag].append(string_split[tag_idx])
    d_log['Total'] = data_log
    return d_log

def filterUserNamesDict(d_log):
    '''
         Filter data to only include data entries with a attached username
    Input:
        d_log: dict
    Output:
        filtered dict
    '''
    filter_string_tracking_id = "userName=(.*)&X-Atmosphere-tracking-id="
    filter_string_transport = "userName=(.*)&X-Atmosphere-Transport=close&X"
    d_log['User'] = d_log['Params']
    for row_idx, row in enumerate(d_log["User"]):
        if ("-" not in d_log["Username"][row_idx] and "/" not in d_log["Username"][row_idx] and "." not
    in d_log["Username"][row_idx]): #Filter away inconsistencies in the data such as useragent or HTTP-
    type
            d_log["User"][row_idx] = d_log["Username"][row_idx]
        else:
            if("userName=" in row and "X-Atmosphere-Transport=close&X" in row):
                d_log["User"][row_idx]  =  filterUserName(d_log["User"][row_idx], filter_string_transport
    )
            elif("userName=" in row and "X-Atmosphere-tracking-id=" in row):
                d_log["User"][row_idx]  =  filterUserName(d_log["User"][row_idx], filter_string_tracking_
    id)
    d_log["User"] = [user for user in d_log["User"] if '"' not in user and "?" not in user]
    return d_log

```

```
def getUniqueUsers(d_log):
    '''
        Converts list to set and therefore only keep unique users, return set
    '''
    return set(d_log["User"])
```

G Appendix G

R code for doing a linear regression classification on crabs

```
setwd("~/Lab2")
library(readxl)
library(MASS)
crab_data <- read.csv("~/Desktop/australian-crabs.csv")
set.seed(12345)
#2.1.1
cl <- crab_data\$CL
rw <- crab_data\$RW
sex <- crab_data\$sex

#2.1.4
glm_analysis <- glm(sex~ cl + rw, data=crab_data, family=binomial())
glm_analysis.p <- (predict(glm_analysis)>0)
ct_3 <- table(prediction=glm_analysis.p, data=sex)
sum(diag(prop.table(ct_3)))
# Missclass GLM = 1 - 0.965 = 0.035
plot(cl, rw, main='GLM | Australian Crabs: CL vs RW', cex.main=0.9, cex.lab=0.8,
xlab='Carapace Length', ylab='Rear Width', col=c('red','blue')[glm_analysis.p+1])
legend("topleft", title='Predicted Sex GLM', cex=0.7, text.font=2,inset=.02,c("Female",
"Male"), lwd=c(2.5,2.5),col=c('blue','red'))
coeffs <- glm_analysis\$coefficients
par(new=TRUE)
abline(-coeffs[1]/coeffs[3], -coeffs[2]/coeffs[3], col="green", lwd = 3)
```