DAT410/DIT728

Group 54
Module 3: AI Tools

Jonatan Hellgren, 960727-7010
Matematikprogrammet GU
gusheljot@student.gu.se

Ankita Rahavachari, 19960709-4605
MPDSC Chalmers University
ankita@student.chalmers.se

January 2021

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.

Hours spent working

Jonatan	Ankita Rahavachari
20 hours	10

1 Reading and reflection

1.1 Jonatan

2 Summary

Machine learning models are very senitive to small changes, both in the data which they are trained on and the values for the hyper parameters chosen when creating the model. This is know as the CACE principle: Chagning Anything Changes Everything. A strategy to mitigate this is to ensemble multiple strategies that gives uncorrelated errors, but this increases the complexity even more and changing the parameter values might have a even more drastic consequence.

In ML data dependencies are often created, where one model takes as input the output of another model. This can become an issue due to the fact that some outputs can be unstable since we do not know how small changes inte tha model will affect the output. A common mitigation for this is to keep versioned copies of the models used for dependencies. A thing to look out for when using other models as dependencies is to continuously remove unnecessary dependencies that give small value to the model.

A common thing with ML programing is that a significant amount of the code actually doesn't have anything to do with machine learning, in the aritcle they say that 95% of the code doesn't. What is instead included is refered to as "glue code" which is written to combine things that aren't made to directly work together or is combined in the wrong way. Another thing they mention is a pipeline jungle where to many preprocessing steps are included. In the article they also mention that changes in the real world can cause the models efficiency to decrease.

All of the things described above are some examples of things that one should be cautious of when creating ML models. If not it may result in technical depth where the code has to many lines which makes it unnecessarily hard to work with and may easily cause bugs when worked with as well as having an unreasonable runtime. So measuring the dept and attempting to mitigate it is a crucial process when doing ML.

3 Take-Aways

My take-aways from this is that I should be more cautious when linking multiple models with each other since the effect of that can be hard to interpret, also that going through the code and trying to make it simpler is always a good idea. Another thing to keep in mind is the sensitivity of the models both on the hyper parameters and the data, the models we create is only as good as the person that implements them, one can not expect the algorithms to do the job for them.

3.1 Ankita Rhavachari

3.1.1 Summary

Technical Debt: It is a metaphor introduced by Ward Cunnigham to help us understand the long term costs incurred by transitioning to new technologies.
The primary goal of maintaining a machine learning model is not to add extra functionality to it but to enable future improvements, correct the errors and improve the maintenance. It is difficult to detect a debt that exists at system level rather than code level. Therefore there has to be few methods to solve these issues instead of using the traditional code level solutions. It is not possible to express the desired behaviour of the system using software logic. In these cases use of ML algorithms is necessary.

Complex models erode boundaries

- Whether a model is retrained fully in a batch or allowed to adapt, adding new features can cause removal of the other i.e., no feature is independent. This principle is known as CACE: Changing Anything Changes Everything. Instead of relying on the combination of components, in many cases using an ensemble might work well because the errors in the component models are uncorrelated. Next thing is to focus on the changes in prediction behaviour as they occur.
- There are situations when we can create a slightly different model from the existing one but this creates a system dependency which makes it more expensive to analyse improvements for the model. This can also create an improvement deadlock.
- The Undeclared Consumers are also known as Visibility Debt. The predictions made by one system are accessible to the other therefore it can be consumed by the other systems. It is expensive as well as causes tight coupling of the model

Data dependencies Cost more than Code Dependencies
There are few packages which are mostly unneeded. These packages are input signals that provide incremental modelling benefit but this can make an ML system unnecessarily vulnerable to change. The occurrence of such underutilized data dependencies are due to legacy features, bundled features etc.
Feedback Loops
There are two different kinds of feedback loops which end up influencing their own behaviour if they update over time. Direct feedback loops show statistical challenges which the researchers might easily find. They are costly to analyse. Hidden feedback loops happens when two system influence each other indirectly
ML system Anti-Patterns
There are several system design anti-patterns that can surface in ml systems which should be avoided. Few of them are Glue code, Pipeline Jungles, Dead experimental code paths, abstraction debt and common smells.
Configuration Debt
Configuration of machine learning systems can accumulate potentially more debt than we imagine. The number of lines of configuration may far exceed the number of lines in a traditional code. Mistakes in configuration can be costly, leading to serious loss of time, waste of computing resources or production issues.

3.1.2 Take-Aways

An ML engineer should be aware of the design choices they make and the model should be designed in such a way that it can be subjected to future changes without incurring massive cost in the future analysis and during maintenance. The engineer is engaged with the ML machine not only during the development-deployment phase but also after the deployment. This acts as an important period in terms of predicting the changes the systems undergoes and also to curate possible ways to enable future improvements and make the system flawless. Also while making a design change it is essential to keep in mind that the small design change should not increase the complexity and the cost of computation because this will eventually slow down the performance of the system.

4 Implementation

To implement a k-NN classifier we defined a class in python named knnClassifier, this class contains 4 functions: `__init__`, `fit`, `predict` and `score`.

4.1 Brief explanation of the functions

The `__init__` function is just used to initialize the classifier, this function is called when the following line of code is passed:

```
knn = knnClassifier(k = 10, distance='euclidian'),
```

here we can choose what value for k we want and also what metric we want to use to measure the distance. Now at the moment it is possible to choose between Euclidean and Hamiltonian distance, this is only to show how easy it is to switch distance function with this design, also adding more is also simple.

The `fit` function purpose it to store the observations positions in the vector space and their label. So basically it stores the neighbors that we later will measure the distance to. To use the `fit` function we use the following line of code:

`knn.fit(X,y)`,

where `X` is the matrix that contains the position for the observations in our vector space and y is their corresponding labels.

With the `predict` function we are able to predict new points in our vector space. The function works by computing the distance to each observation in `X` and then returning the k closest of them, with these k observations a majority vote is executed where the predicted label of the new point will be the most common label in the group that is it's closest neighbors. To use the predict function we will use the following line of code:

`knn.predict(Z)`,

where `Z` is the matix which we want to predict (important that `Z` has the same number of rows as the matrix `X` that we passed in `fit`). This function returns a vector of the prediction for each row in `Z`.

Finally the `score` function is used to evaluate how well the function performed. It does this by computing the accuracy between the predicted labels and the true labels. To use this function we will need to use the following line of code:

`knn.score(Ypred, Ylab)`.

The system classifier works pretty well, it is easy to use and make correct predictions. However it is quite slow compared to scikit-learns k-NN function. This may be due to the fact that we haven't spend much time trying to optimize it yet. However with some tweaks to the code and possibly utilizing paralellization the compute time could be decreased drastically.

4.2 How well did the system work?

To see how well our model is able to predict the data that was given we begin with testing which value for k is the best choice. We do this with a regular train-test split of the data from the two cities Beijing and Shenyang, where the model is trained on the train data and later the accuracy for the train and the test data is computed, the results for this can be seen in Figure 1. As we can see in the figure when $k = 17$ we get the highest test accuracy, which is about 0.8. So the value for k we will try on the validation data from the two cities Guangzhou and Shanghai.

When fitting the model on the validation data we get an accuracy of 0.737. This accuracy is not that great, it is however better that a random guess so that is atleast something.

Some things that could be done to increase the performance will now be discussed, however since the lack of time and the fact that we focused on the code this will not be implemented. Firstly the variables should be standardized before fitted, this is to prevent certain features having higher influence on the prediction then others just because they have higher numerical values. Secondly we have some categorical features that doesn't really suit this model. For example the seasons, they are categorised from 1 to 4 but since they are periodic taking the euclidean distance between them doesn't make sense. Further

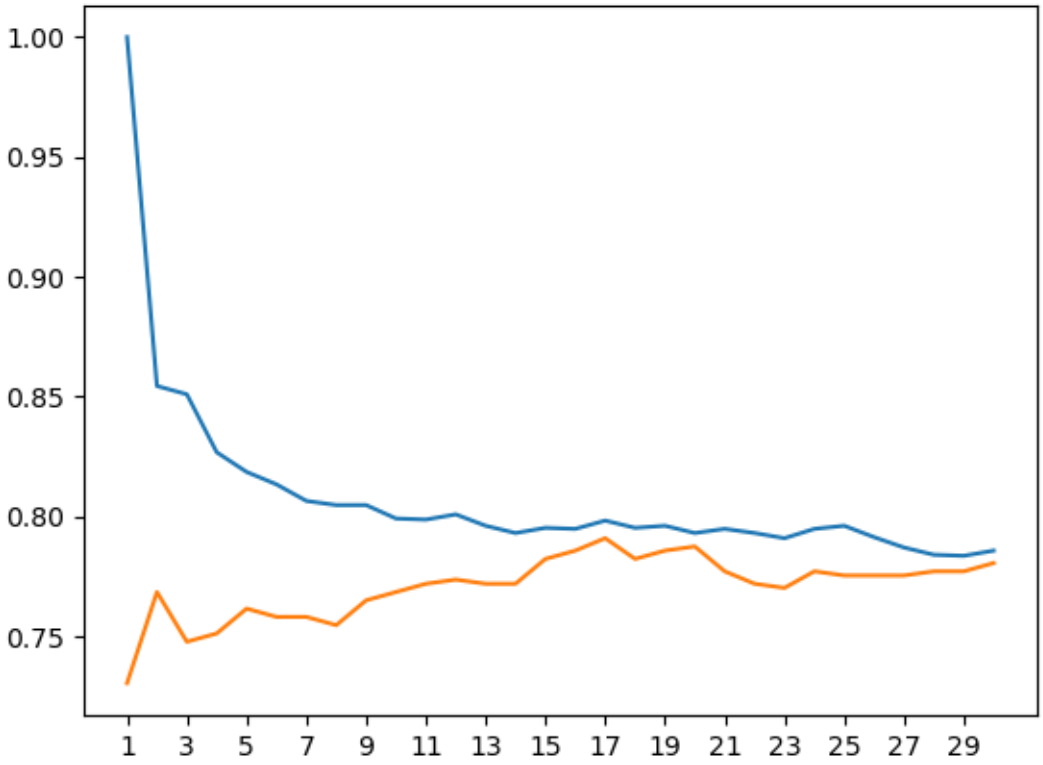

Figure 1: Here we can see how the train and test accuracy changes depending on the value chosen for k. The blue line represents the train accuracy and the orange represents the test accuracy.

tweaking of the model would be necessary to achieve a better fit. And lastly cross-validation would have been a better choice rather then a simple train-test split, since it would yield more information about the model.

5 Discussion

One assumption made during the development of the system is that the training set would be static i.e., the machine learning model is trained based on the initial set of data that is available but this might not be the case after deployment of the system. The training data set might incur small changes or it will no more be considered due to modification in the objective. So this eventually affects the performance when the system gets deployed.

When a system is built, there will be inter-dependencies between the modules.This have to be taken into account while making changes in the module because a minor change made in one module can cause deadlock in the main system and affect the accuracy. Even if the quality of the system is good, retraining of the system is required because of the unexpected changes that can happen in between the development and the deployment phase.

Once the system gets deployed, the ML engineers should stay on with it to know the feedback in order to ensure that the machine stays live and accurate. This means that there has to be frequent accuracy management. Also after the production of the machine learning system, there is no use for the training data as the system has to make predictions using the real data. Therefore the growth of this real world data would be exponential allowing the engineer to build and improve their model after they are in the public platform.

6 Summary and Reflection

6.1 Jonatan

6.1.1 Summary of the Lecture: AI Tools

To easily create AI systems certain tools are required, according to kaggle an online machinge learning competition website python is the most common to use. In python we have a few module that are very useful for these tasks, the names of three common ones are:

- NumPy - Basically a linear algebra library, also the basis for pandas and sklearn
- Pandas - A dataframe library that makes data handling easier then standard NumPy
- Scikit-learn - A library that contains many machine learning tools that are easy to use

However creating AI systems often also require the use of databases, optimization and graphs.

How we write the code that executes the model is critical for making it easier to understand and make it possible for other people to use your code. One thing to think about when doing this is to avoid the beginner mistake of writing one long script, where the code is basically only able to do one thing and lots of rewriting is needed to be done if someone would want to change the objective of the code. A tactic to avoid this is to write programs using the fit, predict, score pattern, which has been implemented in this assignment. This pattern is used in scikit-learn. However it isn't always the best pattern for writing a model, for example in reinforcement learning this might be a bad choice.

A common strategy to use when fitting an AI/ML model to data is to use empirical risk minimization (ERM). ERM tries to minimize a loss function by choosing the right parameter values. It does this usually with gradient descent (GD) and it finds an local minimum of the loss function. Gradient descent is a

useful strategy since it is can be easily be computed with the chain rule aswell as it is easily paralellizable when training bigger models for example in back propagation.

6.1.2 Summary of the lecture: Follow up Recommender systems

The major point that was covered in this lecture was that minimizing the error on the data we have isn't the best choice for an objective, the seasoning behind this is that it can lead to a bad recommender system since we are not taking in to account the future predictions of the system which is basically the point when creating a recommender system. Another thing to think about is that this i a quite dynamic system, because the recommendation will change the future data and thus the new ratings will be affected, this is called a distibutional shift.

A quite common strategy is called A/B testing, here we have two different systems A and B that is distibuted to the users wither randomly or according to some strategy. The purpose of having two different systems is to see which one has the best performance and evalutating changes that way. With this stategy we can either do it online or offline, if we do it online we use the two systems side-by-side and save the data and later evaluate it, if we instead do it offline we take the data collected from the original systems and then evaluate it with our new system we want to try out.

6.1.3 Reflection on the Recommendation System Module

In the previous module we learned a lot about recommender systems, we also got to try implementing one for our selves. Choosing an objective for the recommender system always obvious and evaluating one isn't an easy task. This combined with the fact that they keep generating new data makes the task of recommendation a difficult task. However it gave an good insight in how the systems can be constructed and evaluated.

6.2 Ankita Rahavachari

6.2.1 Summary of the Lecture: AI Tools

- Lifecycle of AI: Process, Represent, Build, Evaluate and deploy Focus on python programming language and basic understanding of different types of data and how to interact with them using machine learning libraries like Numpy, pandas etc.
- A traditional machine learning data is stored in matrices/ Vectors
- Databases are highly relevant for AI industry
- Skeleton of a ML model involves: Fit, Predict, Score pattern in order to avoid the one-long script method.
- But the above model doesn't fit well for reinforcement learning. It relies on roll-outs rather than single predictions
- It is important to preprocess the data to standardize the features to have mean 0 and standard deviation to 1
- Scikit-learn implements data preprocessing in transforms. They can used for standardization, discretization, missing-value etc.
- An example of back propagation is deep learning
- Optimization in general: Gradient descent give local optima in unconstrained, differentiable problems. In convex problems it leads to global optima. Examples of Optimization tools: CVX and Gurobi

6.2.2 Reflection on the Recommendation System Module

The reading and reflection part of the module provided basic understanding of how the combination of two machine learning techniques can complement each technique's drawback and work accordingly. It was insightful to know that adding extra data other than the user's review proved to improve the performance and accuracy of the system. Also insights gained from lecture and the reading part helped to implement a recommendation system in the second part of the assignment. Finally the discussion session gave an overview of what might actually cause the system to degrade in time and produce low accuracy during evaluation.