GROUP 21

Hanna Olsson - 35 hrs

Johan Östling - 35 hrs

We had to spend quite a lot of time on this assignment as it was hard to know exactly how much you should do for the implementation.

Reading and reflection - Hanna

"Hidden Technical Debt in Machine Learning Systems" by D. Sculley et al. highlights the importance of considering technical debt in the development and maintenance of machine learning systems. The paper defines technical debt as "the implicit trade-off between short-term gains in speed, cost, or ease of development and long-term consequences in terms of decreased maintainability, increased development costs, and reduced ability to make changes."

The authors argue that technical debt in machine learning systems is often hidden and can accumulate over time, leading to decreased performance, increased maintenance costs, and reduced ability to make updates. They also highlight the difficulty in detecting and managing technical debt in machine learning systems, due to the complexity and ever-evolving nature of these systems.

This paper really showed me why it is important to have a good design on your AI systems since it can be very costly in the future otherwise. Writing code can be very fast but this paper suggests that the process should be very carefully executed, with things such as documentation reviews and implementing continuous testing practices. Since this makes the process slower it might be hard to implement at first but it is important to create an understanding about the future winnings of these actions.

Reading and reflection - Johan

The article investigates ML systems and how the development of such systems has a bigger chance of increased technical debt, since they have already traditional coding problems plus additional problems that occurs with specific ML-issues. This why it is important to know how to handle the development of ML-systems in a way to minimize technical debt along the process of developing. This article provides guidelines to do this, by for example documentation of the system design, reviewing the codebase and planning for future upgrades. And goes in to further concrete advice on how this could be done. In conclusion this article is relevant for anyone working the the development of ML systems as technical

debt could turn out to be a big problem if you are not aware or know ways to deal with technical debt.

My biggest takeaways was to think for future projects about the issue of technical debt to avoid unnecessary problems. I think many of the points made in this article was sort of obvious, like planning for future upgrades as an example. Or even follow industry-standard practices in the software. These are quite obvious, but I think that since I never really have thought about it as a huge issue to not do these things, this article has gave me perspective on the seriousness of the problem of technical debt for ML systems.

Implementation

Using own KMeans clustering methods

When building the classifier using our own implementation of K-means we used two classes, one for clusters (class Cluster) and one for the classification (class KM_classifier) that uses the cluster class.

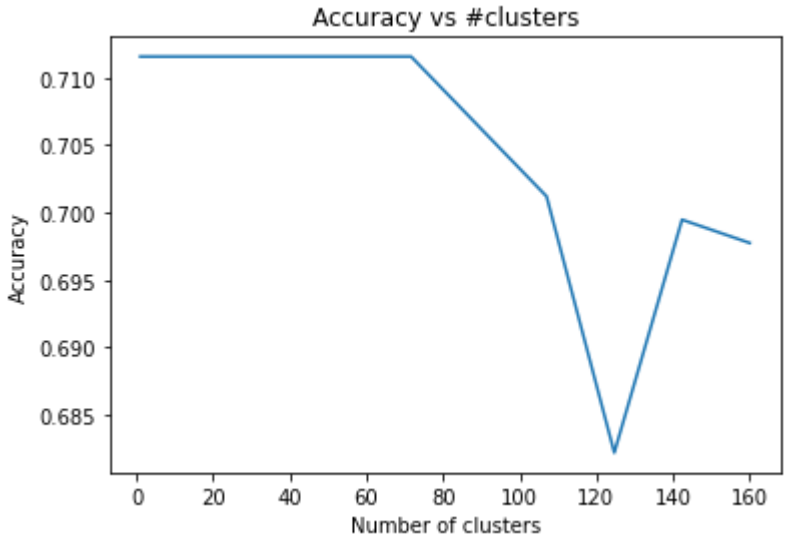
The class Cluster represents instances of a cluster with attributes describing the centre of the cluster, the data points that are in the cluster and what label the cluster has. The label of the cluster is determined by the majority label of the data points that are in the cluster.

The class KM_classifier takes in the desired number of clusters as an argument and every classifier has a list of clusters of type Cluster. The **fit method** takes data points X and corresponding labels Y as arguments. First random clusters are created by letting random data points be the centres. Then the method iterates over all the points in X and assigns them to the cluster that is the closest by computing the Euclidean distance. When all data points have been assigned to a cluster the label of the cluster is updated according to the majority of the labels in the cluster. These iterations continue until the average change in cluster positions is small, which in our case is 0.01. The **predict method** takes in data points X as an argument and simply assigns a label to each data point by taking the label of the closest cluster (Euclidean distance). The **accuracy method** takes in predicted labels and true labels as arguments and returns the fraction of correctly predicted labels. The **precision method** also takes in predicted labels and true labels but instead returns the fraction of 1s that are correctly predicted.

The accuracy is dependent on the number of clusters and to determine what the optimal number of clusters were we made a plot over the relationship between the two using the validation data. The plot shows that about 70 clusters give the highest accuracy.

For training and validation accuracy we get roughly 71% which might seem good but when looking closer at the predictions we notice that almost all labels are predicted to 0 and since the training and validation data consists of about 70% of 0 and 30% of 1 we get an accuracy accordingly. This is why we also choose to measure the precision of the prediction which

indicates that the model fails to predict the 1-labels that exist. We believe that this is due to the fact that the training data is heavily biassed to 0-labels and will thus be much better at predicting these. This is a big flaw with our model. When testing the data separately on Guangzhou and Shanghai we get a very high accuracy of slightly above 90% which is simply due to the fact that these data sets consist of about 90% 0-labels.



This plot shows that the accuracy is constant up to about 70 clusters. This is due to the fact that our model only predicts zeros up to this point. When it starts predicting ones the accuracy drops since the model is so bad at predicting ones so that it predicts some zeros to ones which causes the drop in accuracy.

Using built in methods

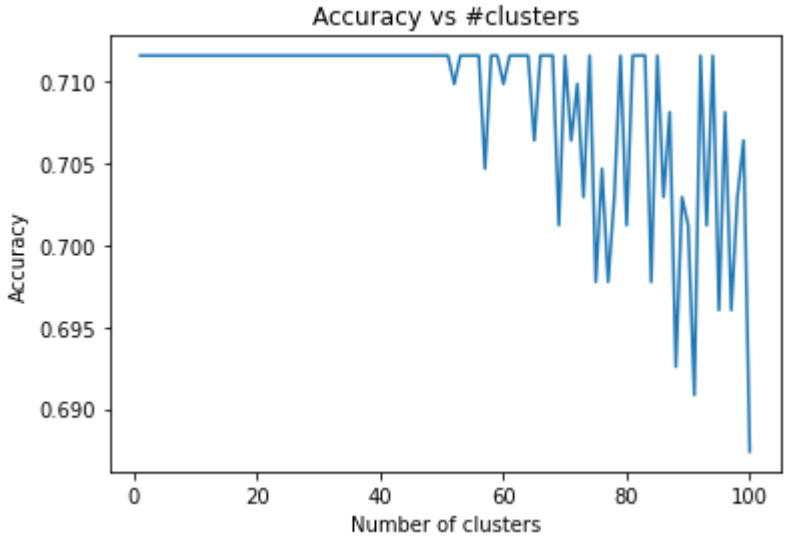
For using built in methods we wanted to compare the accuracy of our own KMean model with a model were we used built in methods from Sickit learn. We implementet fit(), predict() and score within the class Classifier().

For fit() we take in X and Y as arguments, were X is the training data and Y is the test data. fit() returns a list of cluster centers and a list of cluster labels, were the corresponding values at the same index in both lists represents the same cluster. In fit() we use Kmeans().fit(X) wich returns cluster objects. These has attributes such as cluster centers and labels. The problem with only using this built in method is that it does not return the wanted labels. Because we wanted to define a label for a cluster as the majority of value of PM_high for all datapoints in that cluster. Hence we implemented our own algorithm for this which go though each cluster and checks the value of each label for each points and then saves the majority of these labels in a list we call self.cluster_labels.

The predict() method takes in X as parameter, were X is the dataset which we want to predict labels to. predict() returns a list of predicted labels for each datapoint in X. We use Kmeans.predict() method within predict() which returns the index of the cluster each datapoint in X belongs to. Then we use this index to get our desired label (PM_high) from self.cluster_label and adds this to our list of predictions.

The score() methods takes in predicted_labels and true_labels as parameters, the name represents what they stand for. This method is very simple and gives the number of right predictions divided by all predictions and therefore returns an accuracy for our model.

We trained our model on Beijing and Shenyang by combining the data from these cities and then splitting this data into training set and test set (80/20 split). First we examined the training accuracy by training and test the training set. This gave the accuracy of 73% with 70 clusters. Which is the highest we found. We then validated our model by predicting the labels of the test set using our model trained on the training set, we got the following accuracy for 1 to 100 clusters:



This results show that the accuracy does not get higher than 71%.

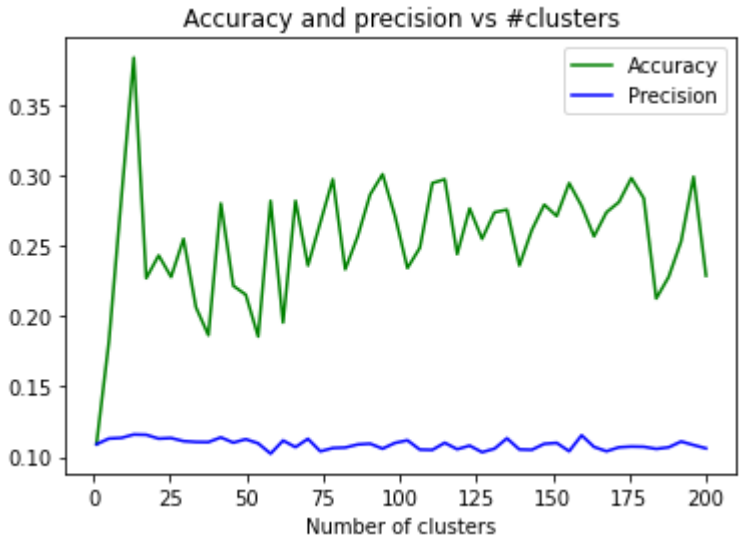
We also tested our trained model on the cities Guangzhou and Shanghai with 70 clusters and for Gouangzhou it gace 89% accuracy and Shanghai it gave 94% accuracy. One interesting takeaway is that these are the same numbers as we got when we uses our own implementation of Kmeans clustering, which may indicate that we did it correctly.

Discussion

The accuracy during development is often higher than the accuracy during deployment since there is a great risk that the model is overfitted to the training data. This means that the model memorises the training data which lead to a high accuracy, but is very bad at generalising which gives low accuracy for new unseen data. You could absolutely argue that our model was overfitted since it basically learned that most labels are 0 and thus predicts 0 for most

data points without learning the real relationship between the data points and their labels. This did however not result in low accuracy for the predictions for the test data since these data sets had a very high rate of 0's compared to 1's which made the accuracy very high instead. But looking at the precision instead shows how bad the model is at generalising since the precision is almost most clusters.

Since we believed that the poor generalisation was due to the imbalanced data, we tried oversampling from the minority label (1) to balance the data. When we fitted the classifier to this data the training accuracy increased to nearly 80% and the precision was no longer 0. However when we tested the model on the test data we got a lower accuracy than before, but the precision still increased. This can be seen in the figure below.



Since neither of our models give sufficiently good accuracy and precision we believe that this data might not be suitable for clustering with K-means. Another reason for the poor model fitting could be that the method for assigning labels to clusters by the majority label in the cluster is suboptimal. We did however not try out another method for this.

Summary of the previous lectures - Hanna

Lecture "AI-tools" 31/1

AI tools are computer programs and software packages used in AI system development and implementation. They come in different forms but can be broadly categorised into two groups: statistical learning and symbolic/knowledge-based. In the lecture, the focus was on statistical learning using machine learning algorithms and prototyping, rather than deployment. Data representation is a crucial aspect of AI tools, with data usually represented in matrix or vector form, but it can also be in the form of time series, 3D im ge data, or

graphs. Computers work best with matrices and vectors as they are heavily based on linear algebra. However, not all data is best represented in matrix form, and other methods such as decision trees can be used. In machine learning, the data used for training and prediction is often assumed to be complete, but this is not always the case, and missing data must be handled through imputation or informative missingness. Model development is another critical aspect, with reusable, differentiable models and optimization libraries used to develop models that can be applied to different datasets. Optimization is important in model development, with gradient descent used to find a local minimum. In conclusion, AI tools are vital for AI system implementation and machine learning algorithms are used in model development and deployment. Data representation, model development, and optimization must be carefully considered for accurate results.

Follow up 2 3/2

During the follow up we talked about minimising the generalisation error, which means that the model should be as good as possible at predicting for new data as it is for predicting the training data. For recommendation systems the data set is commonly very sparse, meaning that every user has rated far from every movie. What is quite unique for recommendation systems is that this data is not random since users are more likely to engage with movies that they think they will like. This means that the model should take into account that movies that are not rated may not have been engaged with for a reason. Another difficulty with recommendation systems is that the distribution for old ratings is not the same as for new ratings since our system changes as for example new users and new movies appear. This is called prediction under distributional shift. We also discussed A/B tests that compare two systems (A and B) at the same taks. There are online and offline A/B tests. For online tests the new system is used in production along with the old system. This is less sensitive to selection bias but can however be costly. For offline tests the old system is used to predict behaviour under the new one. This is what we did with our recommendation system as we only used old data to make predictions and this instead has to compensate for selection bias.

Summary of the previous lectures - Johan

Lecture “AI-tools” 31/1

AI tools are something that helps you improve AI systems. This lecture will focus on statistical machine learning and is most relevant for prototyping.

1. Data representation and processing

Typical ML data is stored in matrices/vectors. For example table of features X and labels Y. There are other types of data that do not have natural tabular representations, for example, time series, image data, and graphs.

But to summarise, you should try to represent data in matrices and vectors, because then it is easier for the computer to make calculations using linear algebra theories. There are expectations to this though, for example, decision trees.

A common problem with data representation in matrices/vectors is that a cell misses a value. We could represent missing values in a matrix X by a missingness mask M. This missingness mask is a matrix that has 0 in a cell where there are values in the matrix X and 1 where the cell misses a value. Two common solutions to missing values are imputation (reconstruct X and predict Y with the reconstructed X) and informative missingness (predicting Y using X and M).

2. Modelling

This part went through reusable models, differentiable models and optimization libraries.

One common beginner mistake in ML programming is the one-long-script tendency. Development in ML has the flowing standardizes common ML workflow:

fit(x,y):	Train model
predict(x):	Predict y from x
score(x,y):	Evaluate model on data

Instead of having one long script, fit, predict and score should be fitted into object-oriented programs. This is what makes the code reusable.

Each time we fit an ML algorithm on a dataset we solve an optimization problem. Many AI tools are based on ERM (Empirical Risk Management) in differential systems. Which is the concept of minimizing loss in the model by finding a function that makes the model as accurate as possible. One strategy to minimize loss is to use the function R (Risk function) and minimize this function by finding a local minimum with the gradient ascent approach. The difficulty of this approach is to differentiate the function R.

3. Learning/optimisation

One example is fair ML, where the goal is for different groups to have equal opportunity. This may be defined as the constraint of having an equal probability of predicted success given true success. For example, that if two applicants for a job has the same probability of succeeding in the job, they should have the same probability of getting the job.

The point is that optimisation is a much broader topic than GD. Because in some cases the model may not be differentiable, discrete or constrained etc.

4. Evaluation

Classical evaluation tools are: Mean squared error (MSE), Accuracy, Cross validations (CV)

For larger systems, you can evaluate with experiment tracking, where you can evaluate the model continuously.

Follow up 2 3/2

For this follow up lecture evaluating systems and problems with recommender systems in general was discussed. The evaluation part of the lecture was about minmizing the empirical risk of predicing ratings and that a natural approach to this is to calculate the empirical risk of predicing ratings and then think about what caused errors in the prediction. Then train/test splits was discussed and then the importance of independent samples. Meaning that the the samples you choose from the training and test portion of the data should be independent. This is because it prevents overfitting by ensuring that the model you test can predict for generalized and unseen data.

Another important subject that was discussed was the difference between correlation and causation. The important takeaway was that even though there is a correlation between two variables, this does not necessary mean that the variables directly effect each other. And that there could be underlying variables that connect these two correlated variables. Causation on the other hand is a relationship between two variables were they directly effect each other. This is important in recommender systems since the recommendations will directly effect how users rate movies, which is also a weakness with recommender systems.

The last subject was A/B tests, which is a way of evaluating which is better of two different solutions, and hence is a powerful tool for data-driven decision making.

Reflection on the previous module - Hanna

It was very interesting working with recommendation systems and one thing that has become very clear is that it comes with many challenges. We only worked with content based filtering during the assignment but it would have been interesting to also make a recommendation system using collaborative filtering to get a better understanding of the differences between the two. Therefore it might have been a better assignment to develop two different recommendation systems.

It would also have been interesting to discuss the possible consequences of bad recommendation systems, like what effect this might have on turnover or customer satisfaction. A bad recommendation system can have far-reaching consequences, negatively impacting both users and the platform itself. A recommendation system that consistently suggests irrelevant or uninteresting items can lead to decreased user satisfaction. This can result in users losing trust in the platform and becoming less engaged, potentially leading to decreased revenue for the platform. Additionally, poor recommendations can result in missed

business opportunities, as users may not discover products or services they would otherwise have been interested in. The fact that there are many negative consequences to bad recommendation systems and the many challenges with developing a good one is very interesting.

Reflection on the previous module - Johan

The previous module was fun to work with, even though one frustrating thing about the module was that how the recommendation system was modelled was up to each group. So it was hard to find an algorithm you would use for your recommender system. I tried to calculate a preference matrix by using linear regression, but I found out that this was more difficult than expected. Because I wanted to, for each user, do a linear connection between what genres makes them rate a movie high, and what makes them rate a movie low. But this was very difficult because several users rated for example action movies both high and low. We then settled to use content based filtering, based on a method called cosine distance. I had never seen that method, but it was easy to understand why it worked. So I am happy that I've learned this new method of finding similar features in data sets.