Module 3 - AI Tools:

Group: 12
Anton Claesson, 971104-3217, MPDSC, canton@student.chalmers.se, 14~h
Lukas Martinsson : 980203-9678, MPDSC, malukas@student.chalmers.se, ~14h

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

Reflection about the paper: Anton

The paper discusses the subject of “hidden technical debt” in real-world machine learning systems, which comes in the form of massive ongoing maintenance costs. The maintenance costs in turn come from several risk-factors that often are overlooked when designing ML systems.

One of the main risk factors that are discussed is related to how changes in one part of an ML system can easily lead to downstream side effects, i.e. changing one thing (such as adding or removing a feature or a changing data dependency) often leads to many required changes in other later parts of the system. An example of this is when using a previously trained model or its output for a new task, as this could create a dependency on the previous model, so any change to the previous model would likely result in required change of the new model as well.

The code architecture of a ML system might lead to significant ongoing costs in the form of glue code if not carefully planned. Otherwise, a pipeline jungle is often gradually developed with multiple intermediate file outputs and expensive testing procedures as new signals and information sources are added to the system. One could also add additional abstraction layers to code by wrapping black-box packages into common API’s, such that the code becomes more reusable for other projects as well as making packages easier to be changed. However, it is often not easy to identify where and how to introduce abstraction into ML code.

There are quite a lot more examples of technical debt and potential solutions discussed in this paper, for example when adding configuration options to the system and when the external world changes. The main take-away for me is that one needs to be very considerate of how one designs a ML system in order to avoid later maintenance costs, which are often much more expensive than the creation of the initial system.

Reflection about the paper: Lukas

The paper's main focus was the discussion of a term called *technical debt*. It explained what it is, how it can occur in different parts of the code and ideas on how to resolve it. To briefly explain *technical debt* it is the cost of “moving too quickly in software engineering”. What this means for ML systems could vary a bit compared to traditional systems since it interacts with the real world in a more direct way, for example through data that continuously can change throughout its lifetime. Hence it’s important to understand how “erosion of boundaries” could increase the technical debt. An example of this is entanglement, which is explained through the CACE principle. Basically when an input vector changes in some way, be it just a single feature or something larger, everything changes.

Further, the paper explained how real world application of ML systems only has a small part of “ML code”. There are several aspects around it that have to be considered such as data collection, feature extraction, monitoring, and so on (described as plumbing in the paper). Therefore the *technical debt* is not only limited to the ML specific code but also to the whole system around it. Plumbing could also suffer from bad patterns that could impede the systems main functions, be it through *glue code* that could make alternative testing considerably difficult or through *pipeline jungles* that are paths that get added as more data signals get added. Identifying these and refactoring them could, especially in the long term, bring significant benefits.

There are several other areas that could bring additional *technical debt* that the paper further explained, among them data dependencies and configuration debts. What the paper made me realize was mainly two things. Firstly, how only a fraction of the whole code is dedicated to the “prediction”, i.e the part the our assignments are focused on. Secondly, when developing a system, if it is possible, develop it slowly so that the maintenance and further development of the system could be considered carefully. This way the *technical debt* could be reduced which in the long term would cost less than if the system was deployed and developed too quickly.

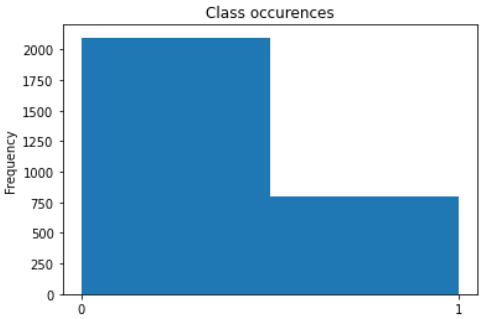
Implementation of KNN classifier

Dataset exploration

We start by inspecting the dataset in order to understand a bit what we are working with. This will help us when evaluating our model and discussing the results.
First, we load and merge the data of Beijing and Shenyang, which will serve as training and validation examples. We also load the data for Guangzhou and Shanghai, which will be separately tested after training.
We note the following properties of the dataset:

- Categorical variables are already dealt with (one-hot encoded).
  Season could perhaps be considered categorical but since season is cyclic keeping it as a numerical variable is probably better.
- There is a large class imbalance both in the training cities and in the testing cities data.

Fig. 1: Training cities target class occurrences



- For Guangzhou we have [1266, 86] class occurrences, meaning that you could get 0.9364 in accuracy by predicting only 0's.
- For Shanghai we have [1218, 133] class occurrences, meaning that you could get 0.9016 in accuracy by predicting only 0's.
- Using box plots of features grouped by the target variable, we see that some features seem more important than others. In particular, 'season', 'HUMI', 'Iws' seem to be most important, while it is hard to say anything regarding 'cbwd' from the box plots as it is categorical. See notebook for figures and more detail.

Implementation

We created our KNN classifier class using the fit, predict, score-pattern as told. For the KNN-classifier, the fit function simply takes the input features and labels and stores them as instances of the class.

The predict function does most of the heavy lifting here, as the KNN-classifier algorithm requires us to find the k nearest number of points for all query points that we wish to classify. We chose to measure the euclidean distance between points. We made a vectorized implementation of the KNN classification-algorithm with NumPy, although there is a for loop for each query point which we think is required. The score function first calls the predict function and compares the predictions to the provided ground truth labels. We then used Scikit-learn to calculate precision, recall, f1 and accuracy metrics.

Evaluation

First, we merge and then randomly split the training data (from Beijing and Shenyang) into a training set and validation set with a 75% / 25% split. This resulted in 2171 training examples and 724 validation examples.

Before fitting the KNN classifier, we standardize the data to zero mean and unit variance in order to get a fair representation of each feature when calculating distance between neighbors. The standardization parameters were calculated from the training set data, after which we then apply the same standardization to the test cities data.

For each odd number of neighbors between 1 and 47 we then fit our KNN classifier with the training data and calculate a score using the validation data.

We choose to evaluate our model with a weighted f1-score. This metric is the harmonic mean between precision and recall, and will provide a fair representation considering the class imbalance. We obtained the best validation score for k=29, with an f1-score of 0.781 (in terms of accuracy this was also the best model with 0.793).

Next, using this model with k=29 that had been fitted on the training data, we tested the model on the two testing cities separately. These are the results:

- Guangzhou: F1 = 0.823, Accuracy = 0.774
- Shanghai: F1 = 0.794, Accuracy = 0.764

These scores are more or less similar to our validation scores, indicating that we have not overfitted. This also makes sense since we are using quite many neighbors which reduces variance. Note that it is possible to predict only 0's and obtain a much higher accuracy (>90%).

Improving the model

Now, as noted when performing data exploration, some features seem more important than others. Moreover, simpler models are always preferred whenever

possible. We therefore drop all features except 'season', 'HUMI', 'lws' and the different ‘cbdw’ features. Performing the same training and evaluations steps as before we obtain the following results:

- Best model from training: 15 neighbors, F1 = 0.791, Accuracy = 0.798
- Guangzhou test: F1 = 0.822, Accuracy = 0.771
- Shanghai test: F1 = 0.796, Accuracy = 0.766

Overall, this is pretty much the same performance as compared to using all features. This indicates at least some of the dropped features mostly contain noisy data when it comes to predictive power and are unnecessary.

Discussion

In our case, we obtain a test accuracy that is similar to our training accuracy thanks to the help of a validation set. However, once the model is deployed problems such as distributional drift might occur, where the underlying dynamics we are basing our predictions on changes. In this case, this might be caused by something like climate change. It could therefore be good to continually evaluate how the deployed system performs to detect if any changes are required.

On an additional note, we believe a KNN classifier might not be suitable for this problem for a few different reasons. First, class imbalance in the data is very high, and we might obtain better performance if we use a classifier which allows us to minimize a loss function such that the imbalance can be accounted for better. Secondly, the feature ‘season’, which from the box plot seems to be important, is not very well handled by a KNN classifier. This is because seasons are cyclical, i.e. winter->spring->summer->-autumn->winter…
Since the KNN classifier compares absolute distances between points, it will consider the season indicated by ‘1’ as being further away from season ‘4’ when in reality it is season ‘1’ and ‘3’ that have the largest distance. This would be equivalent to saying that winter and spring are less similar than winter and summer. A solution to this in this case might be to convert this variable into a categorical one, but we never found the time to test this.

Lastly, we note that the performance in terms of accuracy is quite a bit lower for our model compared to the dummy classifier. However, the dummy classifier is in essence completely useless while our KNN classifier is at least somewhat modeling the underlying distribution of the data, so if the class imbalance was handled our model would achieve a much better performance.

Individual Summary and Reflection: Anton

Lecture summary (AI Tools)

The lecture considered the topic of how some common AI tools are used when developing AI system prototypes in statistical machine learning. First, data representation was covered; most ML data is represented in matrices and vectors. Different kinds of data is often handled by different kinds of systems, but this is changing as multi-modal ML is becoming more common.
Next, the lecture focused on the problem of missing data and different ways to deal with it. Some times, the fact that data is missing could be informative in and of itself and should be utilized (for example by using a missingness mask); other times the missing values could need to be imputed.
There are different ways of performing imputation. One popular method is called MICE which is an iterative algorithm which gradually refines the imputed values by utilizing random sampling (it is some form of a markov chain monte carlo method). One also needs to consider that imputation is not always necessary or better.

Next, the lecture covered development of ml models, from preprocessing to design patterns. Then, it was briefly discussed how one minimizes empirical risk through gradient descent and how the chain rule is utilized in order to automate it with computational graphs and symbolic differentiation frameworks. Other optimization tools were introduced for solving problems where the objective function is not differentiable or is constrained.

Reflection and summary of the previous module (Recommendation systems)

For our recommendation system approach we implemented a content based linear regression approach as covered in the lectures. In our case this worked out quite well as we had data with quite good features as well as users with quite many ratings. In the real world however I expect that collaborative filtering approaches would be more useful, as finding groups of similar users and utilizing this knowledge when making recommendations seems as an approach that would generally be more robust and work better than relying on having sufficient item data.

It is also important to consider the ethical aspects of recommendation systems. How does one design systems that are based on real-world data, where unwanted negative biases are present, and at the same time avoid reinforcing these biases? Spread of misinformation and conspiracy theories might be further worsened when these things are not considered, and from what I know this is an area of active research that is hard to solve.

The main problem that was discussed in the follow up is how to evaluate recommendation systems. Before releasing our models we can obtain certain indicators of performance, but these might not be indicative of the accuracy of new recommendations we make, for example due to distributional drift and the fact that

our models are only based on samples where we have data and that the data is sampled from the real “test” distribution. In the example of movie recommendations, there might be very good movies that have not been rated yet, which might then inaccurately get a low recommended score.
We talked about AB testing in both an online and offline form, and how this can help when comparing the performance of two systems in the real world.

Individual Summary and Reflection: Lukas

Lecture summary (Recommendation system follow up)

The follow up lecture firstly covered with concrete formulas what the assignments goals were and how to calculate them, as well as discussing distributional assumptions. Since just picking a movie in of itself is not random the recommender system could be considered a bit flawed. Furthermore the difference of causation and correlation was explained to understand this point even further. Lastly the A/B testing that was briefly mentioned in the previous lecture was expanded upon and MAR and its different types which was a bit difficult to wrap one's head around was explained.

Lecture summary (AI Tools)

The focus of this lecture was AI tools. Firstly what it is varies depending on the system, whether its statistical learning or symbolic/knowledge-based and also the different types within them. Because of the scope of the course and time frame only the statistical AI tools and small selections within them that exist in python were explained.

The difference of “typical” ML data where every data point is stored in matrices or vectors compared to more general data like time series, image data, and graphs was explained. Then the difficulty of using real world data, i.e. not taken from kaggle or similar websites was explained. Mainly how it could be difficult to interpret and process the data since usually it is not intended for the use of machine learning. This means that the data could be both hard read and also sometimes it could be incomplete for a ML model.

Next it was explained how missing values in data could be solved and taken into account. For example through impute missing values or assuming that the missing values in themself is a sign of something. An example of how to account for it through imputation was the MICE method.

Next, the lecture focused on how to develop a model properly. Since writing a long code were everything from preprocessing to fitting the model to predicting it would be hard to follow and change later on it was explained that creating a class with different functions in it, i.e. fit, predict, and score was much better.

Differentiable system and gradient descent was explained and how as long as every function in the model is differentiable the whole model could through the chain rule be differentiable through the chain rule. Lastly some optimization tools for constrained and non- different problems were touched upon,

Reflection on the previous module (Recommendation systems)

The previous module covered recommendation systems. I learned several things in how to implement such a system. Firstly to choose between two main areas, content based or collaborative filtering. Determining what to choose depends upon a multitude of things, but the main two aspects I would look at when determining what to choose is firstly how the data sample looks, i.e. how many reviews each user give on average as well as how many features each movie have and what the goal of the system is. If in the dataset each user tends to review a moderate amount of movies and all movies have several features (like in our assignment) I would choose content based. However if the goal was different this could change, and if the features in each movie were limited I would definitely choose collaborative filtering. However as with most ML models a mixture could probably be better then each model on their own.

As with the covid false positive example, where a lower accuracy could be better than a baseline model that always predicts negatives it is hard to evaluate how good a recommender system is. Even if we assume maximum profits are always the goal of a company it is hard to find what goal maximizes this. By developing a model that gets as many reviews as possible the model itself will probably improve but it will not be as beneficial to the user base. On the flip side, trying to make the users as content as possible, through for example only recommending movies that they will probably rate high, the model in itself will not improve. Hence the first model would have been better for predicting movies in the future but with a smaller user base and vice versa.

Finally is the issue with bisases, whether they are intentional or not. With the example that the recommender system shows different movie pictures depending on the user. One could argue since ML models do not have any inherent biases this is not racism in any form. However one could also say that the long term result of this could be that a “white” audience never sees any movies from a different ethnicity and thus are not exposed to it, which could make them unintentionally more drawn to movies with “white” actors when in reality, if these biases in the model never existed they would have a much broader preference.