DAT410 - Assignment 3

Eric Johansson & Max Sonnelid

February 8, 2021

Group number: 70	
Module number: 3	
Eric Johansson	**Max Sonnelid**
970429-2854	960528-5379
MPDSC	MPDSC
johaeric@student.chalmers.se	max.sonnelid@gmail.com
25 hours spent	25 hours spent

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.

Introduction

This assignment aims to develop knowledge about the kNN classifier in more depth by building it from scratch. This project has been executed by Eric Johansson and Max Sonnelid. Summaries of associated lectures toghether with personal reflections will be attached in the end of this document.

1 Reflections about "Hidden Technical Debt in Machine Learning System "

1.1 Max Sonnelid

The article *Hidden Technical Debt in Machine Learning Systems* revolves around the concept *technical debt*, which can be defined as the costs occurring by moving too fast in software engineering, which causes errors and make future improvements and maintenance hard. Machine Learning-based software has its own unique aspects of technical debt and two interesting such aspects are discussed in this short article review.

Data dependencies is a major and often hidden contributor to technical debt, which can for example be characterized by keeping underutilized data dependencies in the code which are not really needed. This is something that s easily recognizable in my personal coding behavior. Adding functionality to already existing code is often done faster by adding new, separate input data to not risk disturbing the already working functionality. Repeating this several times results in a code which is difficult to understand and vulnerable to changes in the input data. Instead, it would be better to spend more time on minimizing the data dependencies. Even though this is a time-consuming task, one is rewarded in the long run with a more stable and easily maintainable code.

Another contributor to technical debt is *sensitivity to changes in the external world.* Many machine learning-based systems operates with manually set decision thresholds for performing some action, for example marking an email as spam or not spam. With new data available, these thresholds might need to be updated, which is a time-consuming task to do manually if many thresholds should be updated. Instead, it is probably a better solution if these thresholds are learned by the machine learning algorithm. As senders of spam emails constantly need to evolve the design of emails to not get caught by automatic detectors, this is an excellent example of a system where the external data changes quickly and thresholds need to be updated often in an efficient way.

1.2 Eric Johansson

In the article *Hidden Technical Debt in Machine Learning Systems*, Sculley et al. (2014), highlights critical aspects of Machine Learning. Since ML is increasingly used in all kinds of sectors, the drawbacks of using ML needs to be emphasized, since they often are ignored. The term *technical debt* is introduced as word for the long term costs which comes from moving to fast in software development. This dept needs to be "repaid" some time in the future. Repaying the debt could be refactoring or deleting code, or rebuilding entire models since they no longer work.

The article highlights seven categories of technical debt in different areas of Machine Learning as well as some measures that can be taken in order to mitigate these debts. However, even though it is important to be aware of technical debt, it is hard to quantify it - and thus measure it. To get some feeling of the level of debt that a certain system has, the authors provide some good questions to think about.

- How easily can an entirely new algorithmic approach be tested at full scale?
- What is the transitive closure of all data dependencies?
- How precisely can the impact of a new change to the system be measured?
- Does improving one model or signal degrade others?
- How quickly can new members of the team be brought up to speed?

By going through these question, one can definitely get a feeling about if there exist a lot or no debt in their system.

The biggest take-away reading this article was that it is important to separate assignments that is being done to learn about IT and systems that are being developed to solve an actual problem. Since most students only encounter the first mentioned problem, it is easy to think that their solutions are easy to scale up or use for a longer period - even if this is not the case. Thus, it can be a good exercise to try to develop code or systems that would work in a longer term in order to create a mindset that is durable in "real life".

2 Creating a kNN Classifier

Provide a summary of your evaluation of the system. Describe briefly your implementations of fit, predict, score. How well did the system work? What was the training accuracy, what was the test accuracy?

2.1 Data Preprocessing

In order to train the data on two multiple data sets, we first needed to combine these. This was done by appending the data sets and then re-index them, in order to keep track of their associated label. The model were supposed to be tested on two separate sets, and therefore, we left them unappended.

Since our data sets consists of many different features, there was a high risk of inaccurate predictions, due to that our ML-model is sensitive to the representation and especially scale of input data. To prevent this, we used `preprocessing` from skLearn in order to standardize the data. By doing this, all rows had a mean of 0 and standard deviation 1. Since the scaling function returns an array, we then converted the set back to a dataFrame and added back all the feature names in order to facilitate interpretation of each step.

To learn even more about the data sets we wanted to see which features that is important to make accurate predictions and which are not. Thus, we used `Permutation feature importance` from scikit learn. This method loops through the data sets n times, where n is the number of features. Each time, one feature is left out. In that way, the resulting accuracy can be compared to each other and thus it is possible to rate features based on their importance. This can be really time consuming, since the model must do n predictions. Later down in this report, the speed of our model will be discussed. Since it is quite slow, we choose to use scikit learns knn classifier to calculate the importance, in order to save time. After running the method, only four feature turned out to be of any importance. Therefore, the other were left out.

2.2 `get_neighbours` method

This method takes a data point (a row from the dataset), a dataset (where to retrieve the neighbours from) and a labelset (where to retrieve the neighbours' target values from) as input parameters. Then, for each data point in the input data set, the euclidean distance is calculated between that data point and the specific data point parameter, and is saved in an array. This array is then sorted in ascending order with respect to the distance. For the n first neighbours (n is set when initializing the classifier object), their respective target values are stored in another array, which is eventually returned.

2.3 Fit method

The fit method was simply implemented by setting two instance variables (`trainData` resp. `trainLabels`) to two datasets (data and target values), which are parameters for the method.

2.4 Predict method

This part was the trickiest according to us since we want to model to stick to the standardized fit, predict and score design pattern. Therefore, we only had the `x_test` data set as input. What we then did was to loop through that set, and make a prediction for each row. This was done by first calculating the nearest neighbors for each row, which was done by using our help method `get_neighbours` which returns an array of the respective labels to the k-nearest neighbours (from the training set). This array only contains ones and zeros (the classifications of the neighbors) and therefore we could use `bincount` from numPy to count which of the two classes that were most common, i.e. we weighted all the neighbors uniformly. The class that was most common amongst the neighbors was then our final prediction. This procedure was repeated for all rows in our test set. Finally, a list containing all the predictions were returned.

2.5 Score

The score method was implemented by looping through the output from the predict method and comparing each prediction with the corresponding target values. If the predicted value corresponds with the target value, 1 is added to a sum with start value 0. If not corresponding, nothing happens. The accuracy score is calculated by dividing the final sum by the total number of predicted values.

2.6 Results and discussion

When executing the model, the result turned out mostly as expected. The highest score was yielded when testing on the training data (81,2% ; which indicates some overfitting), though only slightly better compared to the test sets. The results when predicting on Guangzhou and Shanghai were 75,4% resp. 72.3%. To get a feeling of whether this was a good or bad result, we also compared this to scikit-learn's own kNN classifier. Using scikit-learn's own classifier resulted in 77% and 73% in accuracy which is very close to our own classifier. The clear difference between our classifier and scikit-learn's classifier is the time it takes to make predictions with them. Our own model took around 25 minutes to compute all three predictions whereas the model made by scikit-learn yielded an answer in a tenth of a second. We put a lot of effort into minimizing the computing time of our model, but it seems like there still exists some bug in our model that possibly leads to inefficient calculations. What is quite interesting is that the baseline comparison classifier (DummyClassifier with a most frequent strategy) yielded, by far, the best result (92%), which suggests that accuracy score is probably not the best metrics for evaluating the model performance in this task.

The target values in the data set consists of a big majority of 0 values. Therefore, a model which is predicting 0 values for all data points would in many cases give a very good accuracy score compared to the kNN Classifier implemented above, which is what the DummyClassifier has done.

A better evaluation method for this task would be to create a confusion matrix and pay more attention to the number of true 1s, false 1s and false 0s. Using the standard accuracy score, implicitly, attention is only paid to true 0s as these are significantly more numerous than the other categories in the confusion matrix. Looking at the number of false 1s and false 0s and aim to decrease them is therefore a more suitable evaluation method for this specific task.

Another way of handling the problem mentioned problem would be to re-sample the data, by either under-sampling it or oversampling it. The first approach is done by removing samples from the majority class and the second is done by adding more examples. In this way, the classes 0 and 1 can be split within the data set. By doing this, our dummy classifier would only get a score of 0.5. Figure 1 shows two examples of resampling.

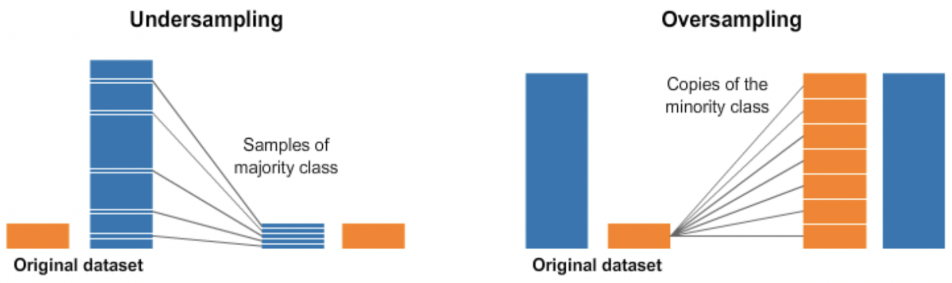

Figure 1: Visualization of undersampling and oversampling. Retrieved from: Kaggle

Cross validation could be a way to measure the training accuracy in a better way and to optimize the hyper-parameters. However, as mentioned above, our model is really slow, and using cross validation to find an optimal k-value would take several hours to compile. One way to get around this could be to find an optimal k using scikit-learn's kNN-classifier and then use that certain k on our own model. This could have been done if the goal of this assignment was to yield the highest accuracy, but since we interpreted the goal of the assignment as learning about how to build our own classifier we choose not to find an optimal k in the above-mentioned way.

As suggested above, a confusion matrix for the predictions on the Guangzhou data set (Figure 2) was also created, where it is possible to calculate that 78 % of the actual 0s are correctly classified, while only 35 % of the actual 1s are correctly classified.

As the aim with this task is to predict when the pollution level is higher than 100 ug/m$\hat{3}$ (represented by a 1), this model can not be considered having a rather good performance as it is only successful 30 % of the times when there is an actual 1.

As too high pollution levels can be expected to cause health risks and other serious issues, it is probably better to create a model that is more prone to falsely predict 1s than falsely predict 0s. Such a model lets authorities implement measures for mitigating these risks sometimes in vain (if 1s are falsely predicted), compared to not getting a warning and missing implementing these measures (if 0s are falsely predicted). The latter could in this case for example lead to people dying from the high pollution levels. One can hope that the authorities wants to prioritize to minimize number of deaths rather than minimizing costs in this case and thus wants a model slightly more prone to falsely predict 1s rather than the opposite.

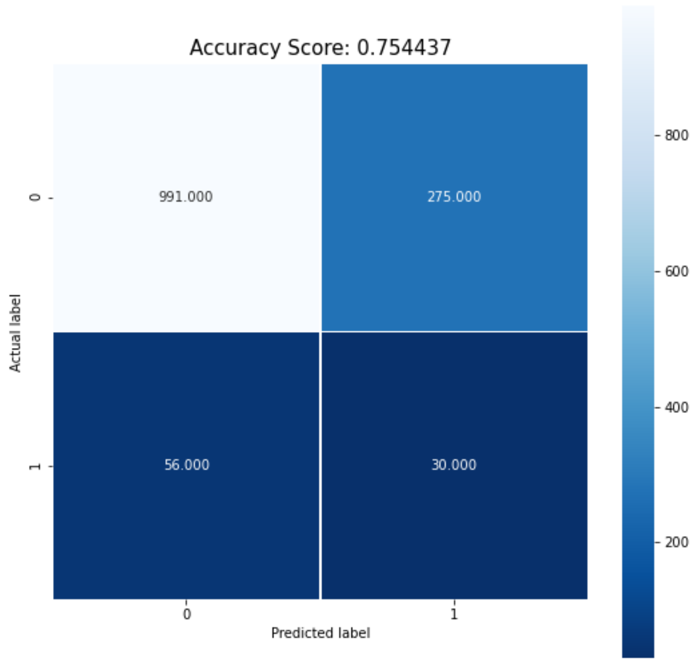

Figure 2: Confusion matrix for the predictions on the Guangzhou data set.

3 Discussion

In most cases, the accuracy of a system when deployed is lower than the accuracy measured during development (training). What are possible reasons for this? Discuss which assumptions are made during development that may not hold, and what could be done to improve the system.

As mentioned in the question, most models performs better on the set that they were trained on compared to a test set. One reason for this is that the original training set might not be an optimal representation of the entire sample space. It is also hard for a model to draw general conclusions by only using a small data set. It is also common that a model learns too much about the training set and thus becomes overfitted.

A practical example of this would be if a system is developed with the aim of distinguishing whether an image contains a cat or a dog. One subcontractor has the responsibility to provide training data images containing cats, while another subcontractor has the responsibility to provide training data images containing dogs. The system developer has not given proper instructions to the subcontractors about the photographing and thus one subcontractor chooses a dark grey background, while another subcontractor chooses a light grey background. As the backgrounds are similar, the system developer does not reflect about this fact when training the system. However, when deploying the system and using it on other kinds of cat and dog pictures, it seems that the system has a strong tendency to falsely predict a cat if the background is darker respectively falsely predict a dog if the background is lighter. In this case, it can easily be concluded from the training data that the background is the most important feature and thus the model becomes overfitted in this respect. The system developer should have done a better job of assuring that the cat and dog pictures, including the backgrounds, are representative for any real-life picture of cats and dogs and therefore include a bigger variety of backgrounds in all pictures.

Even though overfitting seems to be the most common cause of bad results on test set, the opposite (known as underfitting) can also cause issues. This happens when to model has too many constraints, such as when too much weight is put on the regularizing part of the minimizing function. To further explain, the picture below shows a common minimizing function where the first term represent to data and predictions from the training data and the second term is a regularizer (a term to prevent overfitting). With λ set to 0, there is a high change of overfitting, which is not good, but if λ is set to a very large number, to first term will be ignored. That leads to a Θ that only contains zeroes, which also will yield inaccurate results.

$$\boxed{\min_{\theta} \frac{1}{2} \sum_{j=1}^{m} \sum_{i:r(i,j)=1} \left(\theta_i x_j - y_{ij}\right)^2 + \lambda \sum_{i=1}^{m} (\theta_i)^2}$$

Figure 3: Minimizing function from a recommender system. Retrieved from: Canvas

4 Summary of lectures

4.1 Max Sonnelid

4.1.1 Lecture 3

- An AI tool is something that helps you implement AI systems. Some specific tasks that could require AI tools are data collection, optimization and probabilistic programming.
- One of the most frequent used AI tools are programming tools, e.g. Python libraries as Scikit-learn, Keras and TensorFlow. For interactive computing and prototyping, Jupyter Notebook is an useful programming tool.
- Traditionally, Machine Learning data is stored in matrices/vectors, while image data is stored in so called tensors ("3D matrices" where each depth dimension represents a different color). A consequence of the wide use of matrices is a heavy dependence on linear algebra in ML.
- Data frames add indices and names to matrices, similar to a spreadsheet, which makes the data feel more like a database. For very large data sets, which is often the fact for AI implementations in industry, it is often appropriate to store the data in structured databases.
- The standardized common ML workflow is fit, predict and score, which is a useful pattern for dividing the model into mutually exclusive parts. However, this workflow does not fit all kind of AI models, e.g. not reinforcement learning. Still, for all kinds of AI models, it is important to implement standardized input/output patterns to improve reusability and error minimization.
- A common problem is that input data have different scales that could decrease the accuracy of models. A way to mitigate this issue is to standardize the data to mean 0 and standard deviation 1.
- Empirical Risk Minimization (ERM) is by far the most common AI/ML strategy for optimizing models, where gradient descent is a tool for finding the minimal error. It is then necessary that the model is differentiable in order to calculate the gradient.
- Optimization in general is another frequent used AI tool as the answer to a query often is to find the optimal data point by some definition. More specific branches of optimization are optimization with constraints or optimization with decision trees (which are implemented in e.g. CVX and Gurobi).

4.1.2 Reflection lecture

- The main theme of this lecture is to discuss different ways for evaluating the performance of a recommendation system.
- Whether the system is based on collaborative filtering or content, one way to evaluate the system is to minimize the *empirical risk of predicting ratings*. However, this is only possible to do on the training data, and the real objective is to minimize *expected risk* on the test data.

The difference between empirical risk and expected risk is *generalization error* and should be low in an optimal system.

- However, recommendation system are constantly evolving and the real objective for a recommendation system is to minimize the expected risk of the new recommendation system, which is called *distributional shift* and means that we want to minimize the risk based on the new distribution of x and y.
- *A/B tests* is a way for comparing how successful two different systems are at the same task. When users are visiting, for example, a website, they are split into two different groups, which are showed two different version of the same website. Then it is possible to evaluate how users react differently based on the system they are in.
- The *explore/exploit trade-off* is the question whether users should be shown the item with highest predicted utility or the item most useful for improving system, and is also the trade-off between performing A/B-testing or achieving the objective of the system.
- An interesting measure in A/B-tests is the *average causal effect* (ACE) which is the difference between average rating under new system and average rating under old system. If the ACE obtains a positive value, this means that a specific item's rating has improved with the new system.
- *Online tests* are performed when the new system is used in production, which is a costly task, but less sensitive to selection bias.
- *Offline tests* are performed when knowledge from the old system is used to predict behavior under new one. In these tests it is important to compensate for biased selection.

4.2 Eric Johansson

AI tools is a wide area but in this case, AI tools is something that helps you implement AI systems. These systems can be split up into two categories, *statistical learning* and *symbolic/knowledge-based.* The entire lifecycle of an AI will be assesed (however, only on statistical machine learning).

Initially, basic programming skills where discussed and useful frameworks were presented. Thereafter, more focus was put on the development of a model.

When building an AI model, there are three factors that should be considered. Firstly, the model should be reusable (which is often not the case in the academidia). To make a model that is resusable, it is important to use standardized workflows, such as the fit, predict, score pattern. By standardizing input and output in models, a lot of bugs is avoided.

Secondly, the model should be differentiable. By having a differential model, gradient descent can be applied to optimize the model. Gradient descent is executed by starting at one place (an initial vector), choose the direction where the "downward slope" is the steepest in predetermined step length. Then iterate. The nice thing with this is that it only has one non-trivial operation. Even better, we can always get the gradient of Θ without too much problem. Even though Gradient Descent seems nice, it might not always produce the best result.

Finally, it should be optimized by using appropriate libraries such as CVX and Gurobi. Here, optimization is broader than just gradient descent. The above-mentioned librares can be used on problems that are not differentiable.

4.2.1 Summary of reflections lecture

How to evaluate system?

- Since we dont have any dedicated test set, we can only evaluate on the data that is given. However, we rather want to minimize the expected risk than the error on our given data. This is called the generalization error - which we want to be small.
- Choose a useful objective

It is important to think about if ratings and recommendation have direct causation or correlation (i.e, if we change our recommendations, what will happen to our ratings?)

A good solution is to use A/B tests, where users get recommendations from different systems. For example, 50% gets recommendations from system A and the rest from system B. The results can then be compared to each other in order to determine which system that is the superior one. Doing this can be a fruitful way of building accurate systems but can be costly. This is since there exists a trade off between exploration and exploitation. Either you learn a lot about your model or you improve it. This is also known as an online test.

When comparing to systems A and B in an online test, one can use the *average causal effect* as a measure, which basicly is the difference in rating when switching from one to the other.

5 Reflection on the previous module

5.1 Max Sonnelid

What I really liked about the previous module was the combination of putting the technical task into perspective (in the article review and discussion parts) and actually completing the technical task. This gave me a better understanding of the major role recommender systems plays in today's society and their origin. After reflecting about the article, I felt more motivated to create the recommender system as I could easily compare my own work with the submission in the Netflix Prize Challenge.

It was challenging to create a recommender system from scratch, without stated technologies to use, but also an interesting task as we then had to put the goal of the task (to recommend 5 movies each for 5 people) in relation to our own skills, the available data and the time frame for the task. We wanted to focus on an implementation, where we were able to both understand and construct all details from scratch. With respect to our skills and the timeframe, we soon realized that it was a lot simpler to only use one of the datasets in order to be able create a working end product. On the one side weighing quality and wide functionality, and on the other side weighing available resources and time, is probably something that I will have to experience many times in my future career and is the biggest lesson I have learned in this task.

5.2 Eric Johansson

The previous assignment consisted mainly of building our own recommendation system that took Netflix-users ratings as input and gave them recommendation of movies that are prone to enjoy.

A general approach was presented during the lectures, and a more in-depth approach was discussed in the related article. It was very interesting to understand the fundamentals of a recommender system as well as why they are so commonly used. However, when we started to create our own model, we soon realised that is wasn't as easy as it sounded in the article. What we did instead was to start on small scale - recommend a few movies based one movie, only using one of the two data sets. When this worked we eventually elaborated our code until could recommend N numbers of movies based on a persons ratings. What we saw here was the clear benefit of initialize a project by building a very simple model, then build a model that solves another problem until you can solve all the problems that are asked for. (also known as Divide and Conquer).

This assignment was quite different from other tasks that we have encountered in previous it-classes, since no code was given here. In that way, we first had to think about how to solve the problem in an abstract way, and then try to figure out how to code that in the most efficient way. After putting a lot hour into create a functioning model, it was very nice to go back to the article and read it once again. By doing that, we learned a lot by connecting parts of the article to our code. With the new knowledge that we have acquired during the past module, I think that we could create an even better recommender.