DAT410 Module 3 Assignment 3 – Group 26

Yahui Wu (MPMOB) (15 hrs)
yahuiw@chalmers.se
Personal number: 000617-3918
Tianshuo Xiao (MPMOB) (15 hrs)
tianshuo@chalmers.se
Personal number: 000922-7950

February 7, 2023

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

Reading and reflection

Yahui Wu

It's fast and cheap to develop and deploy ML systems but the system could sometimes have maintenance issues. Technical debt is a metaphor introduced by Ward Cunningham to help reason about the long term costs incurred by moving quickly in software engineering. Similarly, ML systems have the technical debt problem and it could be even worse since ML systems have all of the maintenance problems of traditional code plus an additional set of ML-specific issues. ML systems have erosion of boundaries issues due to the difficulties of enforcing strict abstraction boundaries. Entanglement is a typical problem ML systems facing with. There is no inputs of the system independent and changes of anything would affect not only other input signals but almost every aspects of ML systems from model building to application. One possible mitigation strategy is to isolate models and serve ensembles. A second possible strategy is to focus on detecting changes in prediction behaviour. New model built based on a previous and similar one could make it more expensive to improve and maintain the model in the future and this issue is called correction cascades. Mitigate strategies are to augment the old model to learn the corrections directly by adding features or accept the cost of creating a separate model for the coming problem to be solved. Undeclared consumers of a ML system could be expensive to get it improved and sometimes even raised dangerous situations since they create a hidden tight coupling of the initial model to other parts of the stack. Undeclared consumers may be difficult to detect unless some special systems are designed to guard against the case.
Besides code dependencies, data dependencies in ML systems could also build debt and are more difficult to detect. Fist, the data is unstable and this can happen implicitly and explicitly. One common mitigation strategy is to create a versioned copy of a given signal. Second, underutilized data can make an ML system more vulnerable to change. The underutilized data should be regularly evaluated and removed if it is unnecessary. To ensure all dependencies have the appropriate annotations and migration and deletion of the data in the system much safer, tools for static analysis could be of great use.
Another feature of ML systems is the feedback loops which leads to analysis debt. There are different forms of feedback loops and all of them are difficult to detect and address. Direct feedback loops may directly influence the future training data of the model. It is possible to mitigate direct feedback loops by using some amount of randomization or by isolating certain parts of data from being influenced by a given model. A more difficult case is hidden feedback loops, in which two systems influence each other indirectly through the world.
It's surprising that only a small part of the code in ML systems is actually devoted to learning or prediction. There are several system-design anti-patterns we should avoid in ML systems. Glue code is caused by using generic packages and it allows amount of supporting code to get data into and out. It is costly in the long term and could be solved by wrapping black-box packages into common API's. A special case of glue code is pipeline jungles which often appeas in data preparation. It's often requires expensive end-to-end integration tests. Pipeline jungles can only be avoided by thinking holistically about data collection and feature extraction.

Tianshuo Xiao

This paper explores several ML-specific risk factors that need to be accounted for in the design of the system, analyses the causes of each risk and concludes with a proposal for optimization.Here are some of my summaries and take away.
As the machine learning (ML) community has grown, there has been a destabilizing trend where ML systems have become more and more easier and quicker to deploy at an early stage, but maintenance at a later stage has become difficult and expensive. The main reason for this situation is the failure of traditional abstractions and boundaries due to the difficulty of detecting certain technical debts.

The other factor is the potential for ML models to erode abstraction boundaries and generate large amounts of glue code.
As for complex models erode boundaries, it is difficult to enforce strict abstraction boundaries for machine learning systems by specifying expected behaviors. There are several reasons for this problem. Machine learning systems mix signals together so that they become entangled. When we change the input values, the importance and weight of the remaining n-1 features are likely to change, which can lead to entanglement. One solution is to merge isolated models to serve the integration, or to perform multidimensional slicing to detect changes. In addition, when we need to correct the system, we can reduce costs by adding features to distinguish between different situations.
The main factor in technical debt is debt dependency. The main element of technical debt is dependency debt, which means that if a machine learning system is excessively dependent on data and cannot be reasonably processed for data cleaning it will increase the cost of modelling. The situation is generally encountered with unstable data dependencies, underutilized data dependencies and static analysis of data dependencies.
In addition to static analysis, one form of debt analysis is live machine learning about the impact of the system on itself, which will update over time to influence its own behavior. In addition to static analysis, one form of debt analysis is dynamic machine learning of a system's impact on itself, which will update and thus influence its own behavior as it changes over time. This creates direct/hidden feedback loops in the system, which typically using supervised algorithms.
In ML systems, there will be some anti-patterns. Such as glue code (wrap black-box packages into common API's to combat glue-code), pipeline jungles (impurity codes), dead experimental codepaths (consequence of glue code or pipeline jungles), abstraction debt and common smells (subjective indicators). Another area of debt accumulation that may be at risk is the configuration of machine learning systems. In a mature system which is being actively developed, the number of lines of configuration can far exceed the number of lines of the traditional code and each configuration line has a potential for mistakes. Besides, there are other areas of ml-related debt like data testing debt, reprehensibility debt, process management debt and cultural debt.
In conclusion, ML systems often interact directly with the external world, but the external world is rarely stable, which creates ongoing maintenance costs and technical debt. However, these problems are difficult to measure for the time being, and it is important to think about how to assess the impact of these problems and how to solve them at an algorithmic level.

Implementation

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df_bj = pd.read_csv('./Cities/Beijing_labeled.csv')
df_sy = pd.read_csv('./Cities/Shenyang_labeled.csv')
df_gz = pd.read_csv('./Cities/Guangzhou_labeled.csv')
df_sh = pd.read_csv('./Cities/Shanghai_labeled.csv')
df_train = df_bj.append(df_sy)

Ytrain = df_train['PM_HIGH']
Xtrain = df_train.drop(columns = ['PM_HIGH'])
ytrain = Ytrain.to_numpy()

Ytest_gz = df_gz['PM_HIGH']
Ytest_sh = df_sh['PM_HIGH']

Xtest_gz = df_gz.drop(columns = ['PM_HIGH'])
Xtest_sh = df_sh.drop(columns = ['PM_HIGH'])

ytest_gz = Ytest_gz.to_numpy()
ytest_sh = Ytest_sh.to_numpy()

ks = range(1,8)
dis = []
for k_e in ks:
    kmeans = KMeans(n_clusters = k_e ,random_state = 0)
    kmeans.fit(Xtrain)
    dis.append(kmeans.inertia_)
plt.plot(ks,dis)
plt.title('Elbow Curve')
plt.xlabel('k-value')
plt.ylabel('inertia')

class K_Means(object):
    def __init__(self,k):
        self.k = k
    def fit(self,x_train_df,y_train_df):
        ''' generate a mapping reflecting the label of each centroid'''

        x_train = x_train_df.to_numpy()
        self.x_train = x_train
        self.y_train = y_train_df.to_numpy()
        # use kmeans to allocate data to different clusters
        self.kmeans = KMeans(self.k)
        self.kmeans.fit(self.x_train)
        # find out which cluster each data belongs to
        centroid_labels = self.kmeans.labels_
        labels = []
        # create the mapping as a dictionary
        self.dic_ref = dict.fromkeys(range(self.k),0)
        for i in range(self.k):
            # create a list to store the labels of the data in each cluster
            labels.append([])
            # get the index of the data in each centroid
            label_value = list(np.where(centroid_labels==i)[0])
            for j in range(len(label_value)):
                labels[i].append(ytrain[label_value[j]])
            # judge what's the majority label of each centroid
            if labels[i].count(0)/len(labels[i])>0.5:
                self.dic_ref[i] = 0
            else:
                self.dic_ref[i] = 1
        self.ytrain_fit = []
        # predict the label of train data
        for i in range(len(centroid_labels)):
            for key, value in self.dic_ref.items():
                if centroid_labels[i] == key:
                    self.ytrain_fit.append(value)

    def predict(self, x_test_df, y_test_df):
        '''predict the label on test set'''
        self.x_test = x_test_df.to_numpy()
        self.y_test = y_test_df.to_numpy()
        centroid_labels_test = self.kmeans.predict(self.x_test)
        self.ytest_predict = []
        for i in range(len(centroid_labels_test)):
            for key, value in self.dic_ref.items():
                if centroid_labels_test[i] == key:
                    self.ytest_predict.append(value)

        return self.ytest_predict

    def score(self, y_guess, y_test):
        '''evaluate the prediction'''
        Y_test = list(y_test)
        count1 = 0
        count2 = 0
        for i in Y_test:
            if Y_test[count2] == y_guess[count2]:
                count1=count1+1
            count2=count2+1
        return count1/count2

    def score_train(self, y_train):
        '''evaluate the model'''
        Y_train = list(y_train)
        count1 = 0
        count2 = 0
        for i in Y_train:
            if Y_train[count2] == self.ytrain_fit[count2]:
                count1=count1+1
            count2=count2+1
        return count1/count2

    def error(self, y_train):
        '''calculate the mean squared error of the model'''
        ytrain_predict = np.array(self.ytrain_fit)
        y_error = (ytrain_predict-y_train)**2
        error = sum(y_error)/len(y_error)
        return error


clf = K_Means(4)
clf.fit(Xtrain,Ytrain)
y_guess_gz = clf.predict(Xtest_gz,Ytest_gz)
print(clf.score(y_guess_gz, ytest_gz))
print(clf.score_train(ytrain))

y_guess_sh = clf.predict(Xtest_sh,Ytest_sh)
print(clf.score(y_guess_sh, ytest_sh))
print(clf.score_train(ytrain))
```

Listing 1: Predict the PM level

First we use K-means to select the appropriate value of k. The image is shown below and by Elbow Method, we select k = 4.

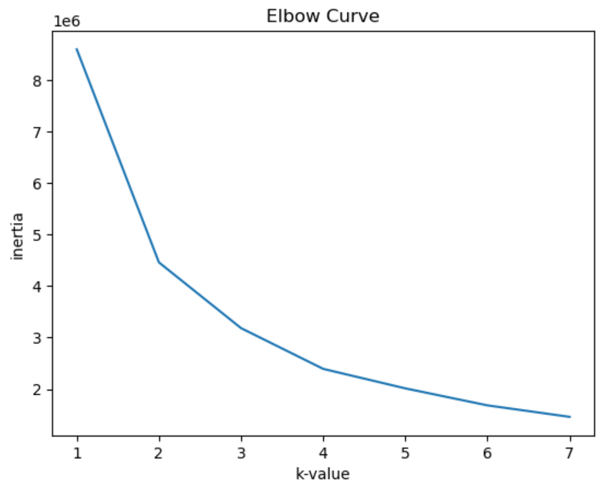

Figure 1: Elbow curve

The classifier was built based on the fit, predict, score pattern using k-means clustering. However, as an unsupervised learning approach, k-means can not help to identify what label a certain centroid belongs to. In the fit part of this classifier, each centroid should be given a label marked as 1 (high level of PM) or 0 (low level of PM) and assigned a label equals to the majority label of points in the cluster. Basically, we aimed to create a specific mapping which could reflect each centroid's label. For example, if three cluster centroids are identified by k-means from scikit-learn, we would like to find a dictionary {0: 1, 1: 0, 2: 0} where the first centroid was labeled as 1, the second was labeled as 0 and the last one was labeled as 0. We achieved this function in the fit part of our classifier. Using the mapping from the first part, we could fit the model on train set and obtain the predicted label of train set after training since each piece of training data belonging to a certain centroid could find its label from the mapping dictionary. Similarly, the label of Guangzhou and Shanghai could be predicted using the mapping given by function fit. In the scoring part, we defined the score by calculating the percentage of correct prediction after comparing the predicted label and the origin real label. The score function could be applied to data of train set as well as test set.
We found the optimal value of k which was 4 in this task by using elbow method. The training accuracy was 0.7250431778929188 and the test accuracy was 0.9363905325443787 on Guangzhou and 0.9015544041450777 on Shanghai. Viewing the test accuracy, we could say that the prediction was pretty good. However, it is not common to get better accuracy on test set than on training set. More discussions about the accuracy on training set and test set could be seen in the next page.

Discussion

In our model, sometimes the accuracy of a system when deployed is lower than the accuracy measured during training, We consider the following reasons:

1.Selection of scoring functions. Normally, we would score the training set using cross-validation, but in our model, we use an algorithm similar to *accuracy_score*, which calculates the percentage that refers to all classifications correctly. This algorithm is sometimes unscientific, especially if the positive and negative samples are unbalanced, as in the case of our weather data in this case, for reasons that will be described in the next article. Cross-validation works by repeatedly using the data, slicing the resulting sample data, and combining it into different training and test sets, using the training set to train the model and the test set to evaluate how well the model predicts. It is particularly suitable when there is less data.
2.In our weather data, especially for Shanghai and Guangzhou, PM_HIGH = 0 accounts for the majority of the data. The detailed ratios are shown in the table below, with PM_HIGH = 0 means not polluted and PM_HIGH = 1 means polluted.

Status	Train	Shanghai	Guangzhou
Polluted ratio	0.275	0.098	0.064
Not Polluted ratio	0.725	0.902	0.936

Once we have classified by K-means, we then label the different groups. Our classification is based on assigning a label to each prime equal to the majority label of the cluster.This can lead to most groups being labelled as 0 ("Not Polluted") due to the large ratio of 0 occupied by the data, which reduces accuracy.From the above table we can see that the proportion of "Not Polluted" in the training set is smaller than the proportion of "Not Polluted" in the testing set, which means that more groups are classified as "0", which results in a lower correct rate in the training set than in the testing set.

Some assumptions

Some assumptions are valid and some will lead to a decrease in accuracy. In our model, we assume that if the number of "0" (or "1") in a group is more than 50%, then we classify all the data in the group as "0" (or "1").This causes a large number of groups to be identified as 'not polluted', which decreases the accuracy of predictions.This classification may not be appropriate. When we validated the model we found that the accuracy of the calculation changed each time we ran it. It is possible that this is due to the influence of initial values and outliers in K-means, which causes the results to be unstable from time to time. And it tends to converge to a local optimum solution.
The other two assumptions are that the cluster centre is the arithmetic mean of the coordinates of all data points belonging to that cluster, and that the distance from each point of a cluster to the centre point of that cluster is shorter than the distance to the centre points of the other clusters.

Improvement

In our K-means clustering, we used Euclidean metric to calculate the distance between data points. The disadvantage of the Euclidean metric is that it treats the differences between the various dimensions of the vectors as equivalent. In our weather data, which is multidimensional, there are multiple factors that affect PM_HIGH, but these factors may affect it to different levels. If we could give different weights to these factors, it would make more sense to group them together and the accuracy of the prediction might be improved.[1]

What is more,We need to normalise the data and eliminate the effect of the magnitude on the data structure. Euclidean distances are mainly used in K-means to calculate distances between samples, and if a feature has a very large range of values[2], then the distance calculation depends mainly on this feature, and consequently on the opposite of the actual situation. It also speeds up the gradient descent to find the optimal solution. These are the points where we need to optimise in order to improve accuracy.

References

[1] Euclidean metric versus Manhattan distance. AliceWanderAI-CSDN Blog_The difference between Euclidean metric and Manhattan distance. (n.d.). Retrieved February 7, 2023, from https://blog.csdn.net/NXHYD/article/details/103887404

[2] Verma, J. (2022, August 3). Easy ways to normalize data in python. DigitalOcean. Retrieved February 7, 2023, from https://www.digitalocean.com/community/tutorials/normalize-data-in-python

Reflection on the previous module

Yahui Wu

In the last module, we implemented our own movie recommendation system. The recommendation system was built based on item-based collaborative filtering and made recommendations to five different users successfully. However, we recognized that it was difficult to evaluate our system. To optimize the objective of a designed system, it is very common to use empirical risk minimization. In our recommendation system, it is easy to evaluate movies that were rated by users but to recommend movies unrated if we believed users might like them, we want unrated movies to be accurately predicted as well. $L(\theta)$ is the expected risk in a distribution p and it's irrespective of whether a user has rated or not, We want the difference between $L(\theta)$ and $\hat{L}(\theta)$ to be smaller. In data, we only see rated movies which means the data is biased. Moreover, the distribution is not fixed because new users or new viewing patterns are introduced when the system is deployed. Another problem is that the way users engaged with the content would change when the recommendation system changed. In other words, it is very important to adjust our system properly to interventions we introduce when deploying the system. Our prediction error could be small even for a bad system where the ratings are low but it's not our goal, what we really care about is the ratings and the quality of our recommendations. Thus we want our ratings to be higher by changing our recommendation system.
Online A/B test could be a method to compare two systems at the same task. The new system is used in production, side by side with the old one. It is costly but less sensitive to selection bias. Offline tests use knowledge from the old system to predict behaviour under the new one. It could be affected by the selection bias because of the intervention of the old system so compensation for such bias is necessary.

Tianshuo Xiao

In the last module, we learned about recommendation system. First, we need to confirm our objective, such as what and how we want to get from the system. Then, proxies and motivation. Data collection is one of the most important parts of building the system, which includes users' ratings and preference. The main methods are content filtering (user-product and product-feature data) and collaborative filtering (user-product data). Content filtering which through similar products get similar scores from users to make recommendations while collaborative filtering through users with similar histories give similar scores to the same products.
After the model was built, the most important part of the process was to evaluate it. Generally, we try to minimize the empirical risk of predicting ratings. We want to be accurate for unrated item, then recommend it to users, so we are supposed to calculate the generalization error (through expected error in predictions).
We have expanded our recommendation system on A/B test, which means that we compare two systems at same task. There are two types of tests, online tests, and offline tests. Online tests combine new system and old system in production, it can estimate the effect of the new system by compare average rating. Offline tests use old system knowledge from the old system to predict behavior under new system. This results in a scenario where different factors interfere with each other. When one user is exposed to may influence others are exposed to, we call this situation as network effects. At that time, we can use content filtering to deal with it. We can assume it as an individual model or combine collaboration or content filtering to do this task. Sometimes, we may make some wrong recommendations, such as we recommend drama movie to an action movie fan. This may because there is some similarity between movies. What is more, some taste unaffected by recommendation.
We have reflected on our assignment. We did not validate the accuracy of the model, which needs to be improved. Besides, we didn't try to evaluate w.r.t. like dimension of SVD. However, we describe the modelling process in detail and the model has a relatively high precision of recommendation.

Summary of lectures

Yahui Wu

AI Tools
January 31, 2022
Last lecture mainly focused on the introductory of AI tools especially some basic machine learning procedures. Data representation and processing is a key step in machine learning. When we learn ML, data is stored in the form of matrices or vectors. Even in reality data is not always stored in a matrix, matrix still has its own ubiquity to represent the real-world data. For example, matrix or vector multiplications are fast, well-understood and almost all ML builds on linear algebra. However, inputs of machine learning algorithms are not always fully observed. It is common for some feature values to be missing in reality. Two common solutions are often to be considered when we handle the missingness of features matrix X. One way is to impute missing values, We can reconstruct X from $\widetilde{X}$ and predict Y from reconstruction $\hat{X}$. The second way is to make use of missingness matrix M and predict Y using both $\widetilde{X}$ and M. The basic idea of imputation is to predict the missing value from observed values of other varables. The imputation functions to predict the missing value can be regression on complete observations or observations with less missingness. However, it is tricky when all observations have some missingness. One of the most popular methods for imputation is MICE. Random sampling is used in multiple imputations. To deal with the issue that data is not missing at random, we can stitch a simple imputation $\hat{X}$ and M, as binary indicators, together for prediction. In ML, fit, predict, score pattern is a standardized common ML workflow. Besides the process of missing values, it is important to standardize features since ML is sensitive to input data. Scikit-learn implements data preprocessing in transformations. Optimization is another key conception we should consider when building ML systems.ERM is the most common strategy to achieve optimization. In a differentiable systems, gradient descent is a helpful method to find the optimal θ. However, gradient descent gives few guarantees of optimality and it could fail when the system is not differentiable. Luckily,there are tools for constrained and non-differentiable problems such as CVX and Gurobi. There are several methods to evaluate the model. For example, we can measure model's error using the mean squared error. Besides MSE, we can use accuracy and cross validation in different application scenarios.

Tianshuo Xiao

AI Tools
January 31, 2022
This lecture introduced the use of AI Tools and the tool libraries. AI tools are implemented in many AI systems, which depend on the nature of the system, such as Statistical learning and Symbolic/knowledge-based.
The programming language for AI is mostly based on Python. there are many types of expressions when processing data. In machine learning, data is stored in matrices or vectors. e.g., horizontal representations of features and columns of labels. However, there is no natural tabular representation for more general data, such as time series. The image data is stored in tensors, and each depth dimension in the 3D matrix represents a different color. Graphs can be represented in many ways(visual).
Matrices are important in machine learning and many machine learning models such as Linear regression, Deep neural networks and Linear programming are built from matrices because matrices or vector multiplications are fast and well-understood. When we create data as Data frames, it makes processing data easier because Data frames add indices and names. Each data type is better handled in a slightly different way in ML and then merged.
When we process data, we may meet some problem on missing data. When we using machine learning algorithms, inputs are often assumed to be fully observed. However, it is common for some feature values to be missing. We have many methods to solve it. First, we can represent missing values by a

missingness mask. We can set to 1 where data is present and 0 where data is missing. What's more, we can predict them by tabular features, such as mean imputation, Knn, remove NaN row, complete with random values and so on.
The basic idea of imputation is to predict the missing values from the observations of other variables and missing data can be learned by regression. Another way is Multiple Imputation by Chained Equations. It creates more than one sample of each missing value to account for variance without predict before. MICE give each variable a placeholder imputed value and repeat for a number of iterations. In conclusion, imputation methods may fail when data is not missing at random but it is not necessary to minimize test error.
Building a model requires the following three key steps: fit(x, y)(Train the model), predict(x), score(x, y)(Evaluate model on data x, y). At the same time, we can solve problems with different systems, such as gradient descent. This is a method of differentiation that makes the error lower by gradually decreasing the gradient of the function, which makes the prediction more accurate. Finally, model evaluation. We can measure models' error by using the mean squared error, accuracy function, cross validation, etc.