DMT Assignment 1

Bruce Ma[1[14210134]], Eva Lampret[2,3[14035227]], and Anne van der Weijden[3[12533130]]

Vrije Universiteit Amsterdam

Abstract. This paper presents a study on various techniques for time-series data analysis, including pre-processing, feature engineering, and classification. The aim of the study is to provide insights into the complexities of working with time-series data and the importance of careful data processing and model selection in achieving accurate predictions. The study examines the performance of well-known algorithms such as Random Forest and Multi-Layer Perceptron (MLP) for classification tasks, and ARIMA for numerical predictions, while investigating the differences between Mean Squared Error (MSE) and Mean Absolute Error (MAE) as evaluation metrics for the models. The study includes a data preparation process consisting of data exploration, data cleaning, and feature engineering, which leads to a cleaned and normalized dataset with less missing data and outliers. The findings of this study can be beneficial for researchers and practitioners working with time-series data in various domains.

Keywords: Data mining · Classification · Numerical predictors.

1 Introduction

Time-series data analysis is a crucial aspect of modern data mining, with applications ranging from finance and economics to weather forecasting and healthcare. In this paper, we present the results of a study exploring various techniques in time-series data analysis, including pre-processing, feature engineering, and classification. Specifically, we examine the performance of well-known algorithms such as Random Forest and Multi-Layer Perceptron (MLP) for classification tasks, and ARIMA for numerical predictions. Furthermore, we investigate the differences between Mean Squared Error (MSE) and Mean Absolute Error (MAE) as evaluation metrics for our models. Our study aims to provide insights into the complexities of working with time-series data and the importance of careful data processing and model selection in achieving accurate predictions. The findings presented in this paper may have implications for researchers and practitioners working with time-series data in various domains.

2 Task 1: Data preparation

2.1 Task 1A: Data Exploration

The original dataset contains 376912 rows and 5 columns, namely id, time, variable, value, and an unnamed column for indexing. The variable column consists of all the relevant attributes that might potentially be useful for the prediction of mood. In Table 1 you can find descriptive statistics of the variables which include the ranges of values (min and max). Looking further into the data description, we find that columns such as call and sms has either value 1 or it is missing, therefore it is logical to assume that user did not have a call/sms unless specifically indicated. It can also be observed that column appCat.entertainment and appCat.builtin contain value below zero, which is not possible due to the nature of the variable. The negative values will be removed.
Taking a deeper look at the data we can see that it contains 27 unique IDs and it is clear that we need to pivot the data if we would like to predict the mood of a user. Also, it can be seen from Figure 1 that we have a lot of missing values. An interesting fact is that in case of a missing mood value, circumplex.arousal and circumplex.valence are missing as well except for two incidents.

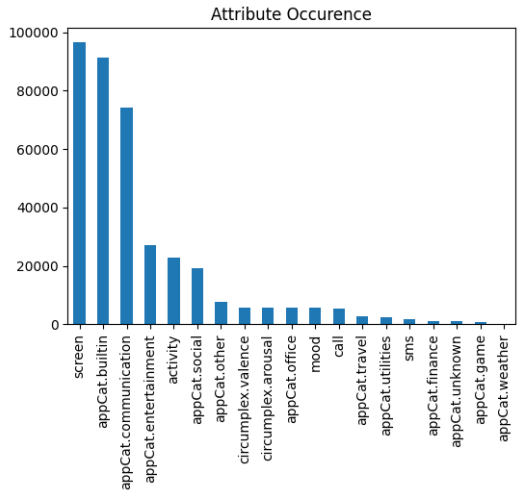

Fig. 1: Attributes count

2.2 Task 1B: Data Cleaning

Given that the goal is to predict the daily average mood of the next day, we need to first pivot the data and make a daily average per user ID. We opted with predicting the mood per individual user because the moods of the different

Variable	Count	Cean	Std	Min	25%	50%	75%	Max
activity	22965.0	0.115958	0.186946	0.000	0.00000	0.021739	0.158333	1.000
appCat.builtin	91288.0	18.538262	415.989243	-82798.871	2.02000	4.038000	9.922000	33960.246
appCat.communication	74276.0	43.343792	128.912750	0.006	5.21800	16.225500	45.475750	9830.777
appCat.entertainment	27125.0	37.576480	262.960476	-0.011	1.33400	3.391000	14.922000	32148.677
appCat.finance	939.0	21.755251	39.218361	0.131	4.07200	8.026000	20.155000	355.513
appCat.game	813.0	128.391615	327.145246	1.003	14.14800	43.168000	123.625000	5491.793
appCat.office	5642.0	22.578892	449.601382	0.003	2.00400	3.106000	8.043750	32708.818
appCat.other	7650.0	25.810839	112.781355	0.014	7.01900	10.028000	16.829250	3892.038
appCat.social	19145.0	72.401906	261.551846	0.094	9.03000	28.466000	75.372000	30000.906
appCat.travel	2846.0	45.730850	246.109307	0.080	5.08650	18.144000	47.227250	10452.615
appCat.unknown	939.0	45.553006	119.400405	0.111	5.01800	17.190000	44.430500	2239.937
appCat.utilities	2487.0	18.537552	60.959134	0.246	3.15850	8.030000	19.331000	1802.649
appCat.weather	255.0	20.148714	24.943431	1.003	8.68400	15.117000	25.349000	344.863
call	5239.0	1.000000	0.000000	1.000	1.00000	1.000000	1.000000	1.000
circumplex.arousal	5597.0	-0.098624	1.051868	-2.000	-1.00000	0.000000	1.000000	2.000
circumplex.valence	5487.0	0.687808	0.671298	-2.000	0.00000	1.000000	1.000000	2.000
mood	5641.0	6.992555	1.032769	1.000	7.00000	7.000000	8.000000	10.000
screen	96578.0	75.335206	253.822497	0.035	5.32225	20.044500	62.540250	9867.007
sms	1798.0	1.000000	0.000000	1.000	1.00000	1.000000	1.000000	1.000

Table 1: Descriptive statistics

users do not follow a similar distribution over time. This is tested by plotting the mood over time for all users.

Then we consider how to deal with missing data for the attributes outside mood. We started with filling the missing values with 0s for all the columns except for mood, circumplex.valence, and circumplex.arousal. This follows the assumption that if a datapoint is not filled, it means the user didn't open the app or didn't send a text or call respectively.

We observe that quite some data missing from mood is from the same period. Therefore instead of imputing a long period of mood, we would rather keep the data integrity and simply remove that period from the data set. This is achieved by checking for 2 missing data points in a row for each individual user. We then follow up to remove single date points in time that are far away from the majority (if it is 5 days away).

A descriptive analysis of the new data shows that it has indeed removed most of the missing values. Now we would like to detect the outliers and impute them together with the missing values. We plot the distribution for all the attributes and it is clear that they are all quite skewed. We decide to apply the MAD (Median Absolute Deviation) method to detect outliers in the columns, except for the column that is binary. The reason for choosing this method is that the outlier values are quite extreme, and MAD deal with the extreme value of outliers much better compare to standard deviation. Also needless to mention that the attribute does not follow a normal distribution. We replace the outliers with NA and impute all the missing values with medium imputation.

We have considered three different methods to impute the missing values. Namely mean imputing, medium imputing, and KNN neighborhood imputing. Unfortunately, KNN neighborhood imputing did not achieve a good result. This is due to the fact that the extreme outliers are so far from the majority and they would

form a cluster and stay undetected. We went with medium imputing instead of mean because it is in general more robust to outliers and non of the attributes are normally distributed. After trying out both methods, we see that the numerical result agrees with the theory. We end up keeping less extreme min and max values using the medium method. Also, an interesting outcome is that we end up being convinced that we should remove columns such as appCat.finance where it contains too many missing values, due to the fact that after imputation the column contains no information. Last but not least, we conclude the data cleansing part by doing a min-max normalization on the data where we would assign a common scale for all the attributes and potentially avoid a dominant attribute.

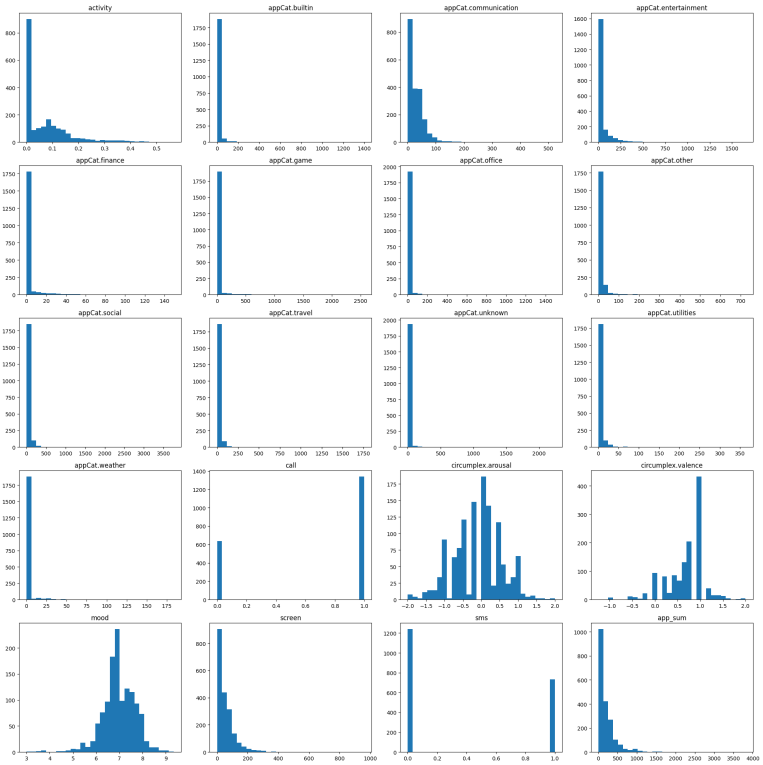

Fig. 2: Distribution of attributes after dropping missing values for mood

2.3 Task 1C: Feature engineering

The data used in this paper is temporal. However, the classical machine learning algorithms (for example decision trees) are not generally able to handle temporal data like that. To ensure we can use these algorithms on our data, we will perform feature engineering.

The process of feature engineering will result in a dataset in which values in each variable contain information from the previous days and are used to predict the next time step. By doing this you create a dataset that, for example, a decision tree can still use to make decision rules and see the next day as a target variable.

As mentioned before, feature engineering was needed to make our time-dependent data suitable for classical machine learning algorithms. We windowed the data and used the arithmetic mean of the variables (including the mood) of the 5 previous days, to predict the mood on the sixth day (target variable). It should be noted that, regarding the initial five rows, there is not enough available data to make a summary of the previous 5 days, and therefore these days are excluded from training and testing.

3 Task 2: Classification

3.1 Task 2A: Application of classification algorithms

In this section, we will discuss two algorithms used for classification. We will first focus on a random forest algorithm, and after that, we will discuss a machine-learning algorithm that is able to handle temporal data.

Random forest Random forest is a machine learning algorithm that creates a "forest" of decision trees and uses that to make predictions. A decision tree is a graphical representation of possible solutions to a decision based on certain conditions, and it can be used for both regression and classification tasks. The tree is trained on data and uses the data to make those decisions and so builds a tree. A random forest is a collection of such trees each trained on a subset of the data. The final decision of the algorithm is then made by collecting all the decisions made by all the separate trees.

A random forest is traditionally not used for time series data because it does not inherently handle the temporal dependencies between sequential data points. However, it is often used for classification and regression tasks. Therefore, if we focus on the classification tasks: of predicting integer variables between 0 and 10, we can use feature engineering to ensure that the predictions are based on the previous data times to predict the next day.

To implement random forest we first need to alter the dataset. For this, we used the feature engineering method discussed in section 2.3. Now that we have a windowed dataset, we need to transform the target variable (mood) into a category. This is done by rounding the values to the nearest integer, the integers are now the class labels.

In this paper, we used the function `RandomForestClassifier` (Pedregosa et al., 2011) as the random forest algorithm to predict the mood of the following day. This function has the hyperparameter *n_estimators*, which sets how many trees there are in the forest, which is normally set to 100. We tuned this parameter by testing difference values (ranging from 10 to 120, steps of 10) and found that using 10 yields the highest f1-score for most of the users. We then zoomed

in and did the same experiment ranging from 1 to 20 and found that 1 was the optimal number of trees in the forest. For the final classification, we thus use $n\_estimators = 1$.

Multi-layer perceptron A multi-layer perceptron (MLP) is a type of neural network that consists of multiple layers of nodes, or neurons, each connected to the next layer. The MLP is a feedforward network, meaning that the information flows in one direction, from the input layer to the output layer, without any feedback loops.

Each neuron in the MLP receives input from the neurons in the previous layer, and computes an output based on a weighted sum of those inputs, along with an activation function that introduces nonlinearity to the network. The weights of the connections between the neurons are adjusted during training, using backpropagation, to minimize the error between the predicted output and the true output.

The MLP is a powerful tool for classification tasks, as it can learn complex decision boundaries between classes. However, it requires careful tuning of hyperparameters, such as the number of hidden layers, the number of neurons in each layer, and the activation function, to achieve good performance and avoid overfitting.

In this paper, we used the function `MLPClassifier` (Pedregosa et al., 2011) for the multi-layer perceptron. We have tuned the parameter *hidden_layer_sizes* as this influences both hidden layers and the number of neurons. We have chosen the $reLU$-activation function as this is commonly used in multi-class classification, which is what we are doing here. We tuned this parameter by testing difference values (ranging from 10 to 130, steps of 10) and found that using 10 yields the highest f1-score for most of the users. We then zoomed in and did the same experiment ranging from 1 to 20 and found that there was no big difference and so we continued with 10 hidden layers.

Evaluation For evaluation, we used f1-scores. These are commonly used for classification evaluation. We did not separately calculate the precision and recall as these measures are part of the f1 score and in this paper, we are mainly focused on the overall performance of the model.

Accuracy measures the percentage of correctly classified instances out of all the instances in the dataset. It is defined as follows: Accuracy = (Number of Correctly Classified Instances) / (Total Number of Instances). Accuracy is a useful metric when the dataset is well-balanced and has an equal number of examples for each class. However, our dataset is imbalanced, and thus accuracy can be misleading as it does not take into account the relative frequencies of the different classes.

The F1 score, on the other hand, is a more robust metric that takes into account both precision and recall. Precision measures the percentage of true positive predictions out of all positive predictions, while recall measures the percentage of true positive predictions out of all actual positive instances. The

F1 score is the harmonic mean of precision and recall, and is defined as follows: F1 score = 2 * (precision * recall) / (precision + recall). The F1 score is a useful metric when the dataset is imbalanced, as it gives equal weight to both precision and recall. A high F1 score indicates that the model has both high precision and high recall, and is therefore performing well.

Results Table 2a shows the f1-scores of the classification using random forest. As you can see in the table, there is a lot of variation in the results (ranging from 0.0 to 1.0). The average f1-score is 0.45 (with a standard deviation of 0.29) for the random forest algorithm and 0.53 (with a standard deviation of 0.23) for the multi-layer perceptron. From this, we can conclude that the random forest classifier has poor performance overall.

Table 2: F1-scores of the classification methods.

(a) F1-score of the classification of the tree algorithm per user.

User id	F1-score
AS14.01	0.06
AS14.02	0.08
AS14.03	0.57
AS14.05	0.71
AS14.06	0.48
AS14.07	0.48
AS14.08	0.51
AS14.09	0.06
AS14.12	0.40
AS14.13	0.23
AS14.14	0.04
AS14.15	0.91
AS14.16	0.71
AS14.17	0.47
AS14.19	0.66
AS14.20	0.71
AS14.23	0.46
AS14.24	0.63
AS14.25	0.24
AS14.26	0.20
AS14.27	0.39
AS14.28	0.0
AS14.29	0.60
AS14.30	1.0
AS14.31	1.0
AS14.32	0.52
AS14.33	0.09

(b) F1-score of the classification of the multi-layer perceptron per user.

User id	F1-score
AS14.01	0.22
AS14.02	0.25
AS14.03	0.4
AS14.05	0.51
AS14.06	0.41
AS14.07	0.31
AS14.08	0.61
AS14.09	0.08
AS14.12	0.47
AS14.13	0.37
AS14.14	0.65
AS14.15	0.82
AS14.16	0.58
AS14.17	0.61
AS14.19	0.33
AS14.20	1.0
AS14.23	0.27
AS14.24	0.56
AS14.25	0.85
AS14.26	0.44
AS14.27	0.41
AS14.28	0.46
AS14.29	0.64
AS14.30	1.0
AS14.31	0.8
AS14.32	0.51
AS14.33	0.73

To explain the results, we initially hypothesized that the length of each user's dataset is simply too small to train and test on. To test this hypothesis, a Pearson-r correlation with the length of the dataset for each user was performed. If there were a correlation (a bigger dataset relates to a higher f1-score) might suggest that the hypothesis might be true. However, we found that correlation to be 0.21 ($p = 0.29$) for the random forest algorithm, and 0.02 ($p = 0.90$) for the multi-layer perceptron. Therefore, we cannot safely say that the length of the dataset correlates to the f1-score. This does not necessarily mean that the length of the dataset does not influence the performance of the random forest algorithm. It can still be true, meaning that the datasets of all users are simply too small to train a model on. Future studies could try and use the random forest classifier on the whole dataset.

Another reason to explain the poor results might be that mood is just simply not a categorical variable. To force it into one we had to round the data and by doing that, we threw away valuable information. Later in this paper, we will focus on numerical predictions as this would be a more fitting procedure for this dataset.

3.2 Task 2B: Winning Classification Algorithms

The TalkingData AdTracking Fraud Detection Challenge was a data science competition hosted on Kaggle in March-April 2018. The competition was sponsored by TalkingData, a Chinese big data company that specializes in mobile app analytics and marketing. The goal of the competition was to predict whether a user will download an app after clicking a mobile ad. This is a challenging problem in the field of digital advertising, as fraudulent clicks can artificially inflate ad metrics and waste an advertiser's budget. The dataset used in the competition contained approximately 184 million rows of clickstream data and was provided by TalkingData. Participants were evaluated based on the area under the ROC curve (AUC) metric, which measures the accuracy of binary classification models. The competition attracted almost 4,000 teams, winning one achieved an AUC score of 0.98343 on public board.

The dataset contained clickstream data from a mobile advertising platform. The dataset was provided by TalkingData, a Chinese big data company that specializes in mobile advertising. It has approximately 184 million rows of clickstream data from various mobile devices, such as smartphones and tablets. Each row in the dataset represents a click event on a mobile ad and includes information such as the device's IP address, its operating system version, the time of the click event, and the app or website on which the ad was displayed. The dataset also includes a binary target variable indicating whether the click event resulted in a user downloading the advertised app.

The winner of the competition was the team named ['flowlight', 'komaki'].shuffle(). LightGBM models, combined with extensive feature engineering, hyperparameter tuning using grid search and Bayesian optimization, and class imbalance techniques were used. Moreover, Their approach relied heavily on negative downsampling [1, 2]. This means that they used all of the positive examples (where

isattributed == 1) and down-sampled the negative examples during model training. They down-sampled the negative examples until their size was equal to the number of positive examples. This resulted in discarding approximately 99.8% of the negative examples, but they did not observe significant performance deterioration when they tested their initial features. Additionally, they achieved better performance when creating a submission by bagging five predictors trained on five sampled datasets created from different random seeds. This technique allowed them to utilize hundreds of features while keeping the training time for the LightGBM model to less than 30 minutes.

Their main idea of the winning approach in the TalkingData AdTracking Fraud Detection Challenge was extensive feature engineering and the use of an ensemble of machine learning models. The team focused on incorporating a wide range of time-based, device-based, and domain-based features, including click frequency, time since the last click, device type, OS version, IP address, publisher, and app ID. They also engineered interaction features between these variables. By using machine learning models, including gradient boosting machines, logistic regression, and neural networks, and blending their predictions, the team was able to create an ensemble model with high accuracy. Additionally, they employed various techniques to address the class imbalance in the dataset, including oversampling and threshold adjustment.

To conclude, in the comparison with the other teams that ranked in the top 10, they all used similar methods. However, the crucial reason that made the dinner was with employed of different techniques for addressing the class imbalance in the dataset. (*Winninng team*, n.d.)

4 Taks 3: Assosiation rules

One of the approaches that were mentioned during the lectures is hierarchical clustering. This involves creating a tree-like structure where similar items are grouped together at different levels of the tree. This specific method creates a hierarchy of clusters by recursively partitioning objects in a top-down or bottom-up manner. (Kaur, 2015) There are two types of hierarchical clustering algorithms: agglomerative, which is a bottom-up approach that starts with each data point as its own cluster and then merges them into larger clusters, and divisive, which is a top-down approach that starts with the entire dataset as one cluster and then divides it into smaller clusters. (Madhulatha, 2012)

Hierarchical clustering has several advantages.Firstly, it produces a dendrogram that visually displays the clustering structure, making it easier to interpret and understand the relationships between the data points. Secondly, it does not require the user to specify the number of clusters beforehand, allowing for a more flexible and adaptive approach to clustering. Thirdly, it allows for the identification of both global and local clusters, making it suitable for datasets with complex structures. Fourthly, it can handle various types of data, including categorical, ordinal, and continuous data. Finally, hierarchical clustering can be

used for outlier detection and can handle missing data, as it only uses pairwise distances between the data points. (Zhao, Karypis, & Fayyad, 2005)

However, the interpretability of the resulting dendrogram deteriorates heavily with increasing database size. (Achtert, Böhm, & Kröger, 2006) Another drawback of hierarchical clustering is its computational complexity, as it is a relatively slow process. It involves the computation of pairwise distances between all observations in the dataset, which can be computationally demanding, leading to a significant increase in the computational load as the size of the dataset grows. Prior to performing clustering, data rescaling may be necessary. The sensitivity level will vary depending on the distance metric utilized for calculating the distance between data points. (Ellis, 2021)

5 Task 4: Numerical prediction

5.1 Time-series machine learning: ARIMA

We would like to predict mood using ARIMA model. First we discuss the steps needed before training the model. We would first plot the mood per user ID and chyeck for Stationarity via ADF test, including the first differences. Then we would use the ACF test to determine the best parameters combination of p,d,q. After all the checks are over, then we fit and predict the data.
In practice, AUTO ARIMA is used, this is a python function that determines the best parameter combination by looping through the combination of parameters and find the best one by testing the ACF (autocorrelation function). Then the data for each individual is split into 80% training set and 20% testing set. It would also use differencing to remove the non-stationary component in the data. After that we train the model and test it and generate the MAE between the predicted value and the real value from the testing set. Due to the fact that the data set for each individuals are quite small, some extreme outliers could have quite a big impact, therefore we would like to mitigate that impact by opting to use MAE instead of MSE.
Unfortunately the performance of ARIMA is limited by the lack of data point considering the amount of attrributes presented in the sample. This is confirmed where quite some IDs are not getting any parameter optimization from Auto ARIMA which indicate that they do not present enough information to form a pattern. However if the data set is rich enough, we believe the result would be much better. In particular for ID A14.02, we can see from the figure that it predicted the user mood quite well.

6 Task 5: Evaluation

6.1 Task 5A: Characteristics of evaluation metrics

We would like to evaluate the performance of the ARIMA model using the two main techniques, namely mean absolute error (MAE) and mean squared error

(MSE). MAE is the sum of absolute errors between the predicted value and the real value divided by sample size, MSE is the sample mean of the squared error. Mathematically, let n be the sample size, Y_i be the real value and $\hat{Y}$ be the predicted value . MAE $= \frac{\sum_{i=1}^{n}|Y_i - \hat{Y}|}{n}$ and MSE $= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$. We can see that mathematically, MSE punishes extreme values a lot harder than MAE due to the fact that error is squared. This can be both an advantage and a disadvantage. On the one hand, you can detect outliers easily. If they are important to be predicted, then MSE would be quite handy. On the other hand, if you would like to mitigate the effect of the outliers and have a more robust evaluation method, MAE is a good idea as it assigns the same weight to all values of error.
A good example where MSE and MAE would give identical result would be trying to predict a variable y that has constant value A. We could be trying to predict the weather for a future day and coincidentally the historical dataset given have all temperature of A. In this case, the mean of the true value and the predicted value are both A as well. hence MSE and MAE are both 0.

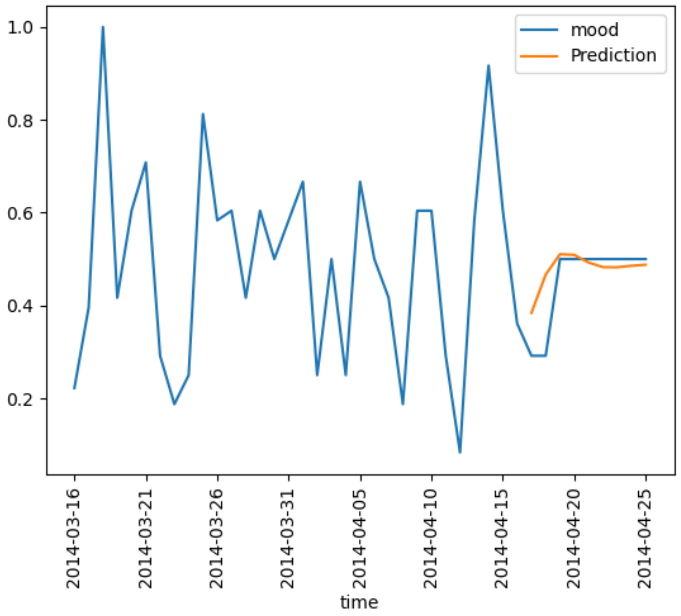

Fig. 3: Example of a prediction made by the ARIMA model for user AS14.02.

Figure 3 shows an example of a prediction made by the ARIMA model, here the model predicted the mood of user AS14.02. The figure shows the true mood, as found in the data over time in blue, and the prediction made by the ARIMA model in orange. The prediction seems to generally follow the trend of the mood

as reported in the data. As seen in Figure 4 we see that the corresponding MSE and MAE are also relatively low compared to predictions for other users.

6.2 Task 5B: Impact of evaluation metrics

In section 6.1 the different evaluation metrics are described. Figure 4 shows the result of the experiment described in section 5. On the left, the figure shows the Mean Squared Error (MSE) and on the right, we see Mean Absolute Error (MAE). It is noticeable that the values for MSE and MAE are about a factor of 10 smaller, which can be explained by the squaring of a value between -1 and 1. Furthermore, we see that the model behaves relatively the same under the different metrics. They, however, do not behave exactly the same. This is due to the fact that MSE gives more weight to outliers as squaring an outlier (bigger error) yields a bigger difference whereas MAE weighs all the differences the same. This is shown in an interesting occurrence in the last two users (AS14.32 and AS14.33) who have (approximately) the same MSE but not MAE. This clearly shows the influence of squaring the errors and therefore giving more weight to the outliers, in this case, AS14.32 has more extreme errors.

It is also noticeable that the model does not behave the same for every user. The results suggest that an ARIMA is not a universally usable model. In this study, we retrained the model for every user. The low performance might be due to the fact that there were few data points per user and therefore it might have been hard to sufficiently train the model. Future studies could look at using ARIMA for the whole dataset or use cross-validation to set the model parameters. Another solution might be to create more data points by, instead of grouping by day, group per 12 hours (morning and evening).

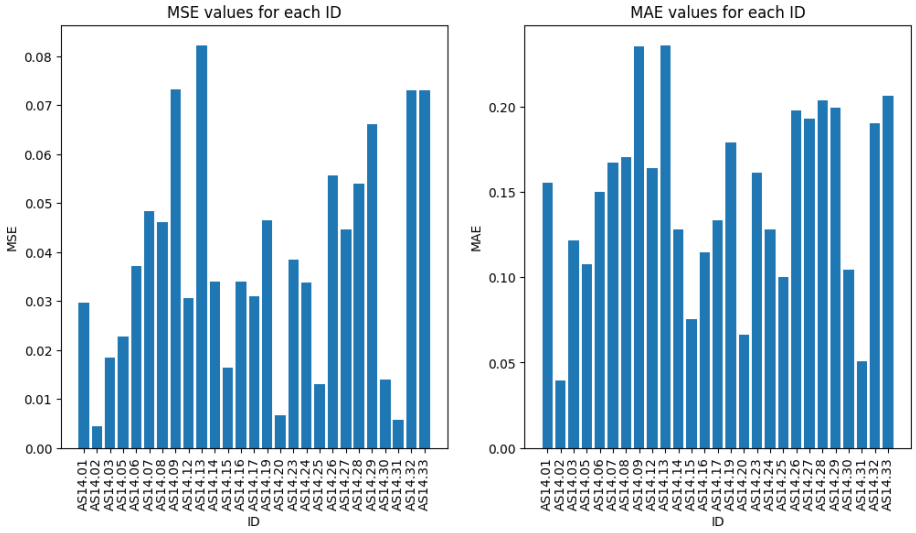

Fig. 4: Result of the experiment described in section 5. On the left, the MSE is shown, and on the right, the MAE is shown.

References

Achtert, E., Böhm, C., & Kröger, P. (2006). Deli-clu: boosting robustness, completeness, usability, and efficiency of hierarchical clustering by a closest pair ranking. In *Advances in knowledge discovery and data mining: 10th pacific-asia conference, pakdd 2006, singapore, april 9-12, 2006. proceedings 10* (pp. 119–128).

Ellis, C. (2021). *https://crunchingthedata.com/when-to-use-hierarchical-clustering/.* (Accessed: 29-01-2023)

Kaur, N. P. J. (2015). Cluster quality based performance evaluation of hierarchical clustering method. *2015 1st International Conference on Next Generation Computing Technologies (NGCT).*

Madhulatha, T. S. (2012). An overview on clustering methods. *arXiv preprint arXiv:1205.1117*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Winninng team. (n.d.). https://www.kaggle.com/competitions/talkingdata-adtracking-fraud-detection/discussion/56475. (Accessed: 23-04-2023)

Zhao, Y., Karypis, G., & Fayyad, U. (2005). Hierarchical clustering algorithms for document datasets. *Data mining and knowledge discovery*, *10*, 141–168.