FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Jyrki Kankaanranta

USING MACHINE LEARNING TO PREDICT SMARTPHONE USAGE

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
December 2022

Kankaanranta J. (2022) USING MACHINE LEARNING TO PREDICT SMARTPHONE USAGE. University of Oulu, Degree Programme in Computer Science and Engineering, 22 p.

ABSTRACT

This thesis shows the process of creating and analyzing a machine-learning model. It goes over prevalent classification algorithms and their advantages and disadvantages. Furthermore, techniques and metrics used to evaluate the performance of the model are introduced. In the latter part of the thesis, a Random Forest model is implemented. The objective was to predict the participants' smartphone usage, more specifically the category of an application they had opened. This starts with a pre-processing phase, where relevant information is extracted from the raw data. Multiple variations of the model are built, and the best-performing model was able to achieve 63.37% accuracy. Additionally, the features are scored to provide more insight into the model. The thesis ends with a brief discussion section, which contemplates the reasons behind the results, some of the model's deficiencies and how it could be improved.

Keywords: machine learning, Random Forest, smartphone

Kankaanranta J. (2022) ÄLYPUHELIMEN KÄYTTÖTOTTUMUSTEN ENNUSTAMINEN KONEOPPIMISEN AVULLA. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 22 s.

TIIVISTELMÄ

Tämä kandidaatintyö esittää koneoppimisen mallin liittyvän rakentamis- ja analysointiprosessin. Joitakin yleisiä luokittelualgoritmeja ja niiden hyötyjä ja haittoja käydään läpi. Myös mallin suorityskyvyn mittaamiseen käytettyjä tekniikoita esitellään. Työn myöhemmässä osassa rakennetaan satunnaismetsä-malli, jonka tavoitteena on ennustaa osallistujien älypuhelimen käyttötottumuksia. Käytännössä malli yritti määrittää juuri avatun sovelluksen kategorian. Tämä prosessi alkaa datan esikäsitelyllä, jossa olennainen informaatio eristetään raakadatasta. Tästä mallista tehtiin monta eri muunnelmaa, joista paras onnistui saamaan 63,37% tarkkuuden. Lisäksi koneoppimisominaisuudet pisteytetään, jotta mallista saadaan enemmän tietoa. Kandidaatintyö päättyy lyhyeen keskusteluosioon, jossa pohditaan tulosten syitä, sekä mallin puitteita ja parannuskohteita.

Avainsanat: koneoppiminen, satunnaismetsä, älypuhelin

TABLE OF CONTENTS

FOREWORD

I would like to thank Aku Visuri for providing the subject and supervising this thesis.

Oulu, December 4th, 2022

Jyrki Kankaanranta

LIST OF ABBREVIATIONS AND SYMBOLS

BN	Bayesian Network
CSV	comma-separated values
GBT	Gradient-Boosted Trees
HCI	human-computer interaction
KNN	K-Nearest Neighbors
LR	Logistic Regression
ML	machine learning
MDI	mean decrease in impurity
MLP	Multilayer Perceptron
RF	Random Forest
SVM	Support Vector Machine
UI	user interface

1. INTRODUCTION

For a few decades now, computers are being integrated more and more into everyday life. One noticeable manifestation of this is the use of online services. These days most human interactions seem to happen online, no small part due to the recent pandemic. There is more data available than ever before, and it will only increase.

Information is power: all this data provides opportunities for companies and academia. An obvious commercial utilization of the data is targeted advertising or recommendations. Likewise, computers or software could adapt on the user's habits, making the interactions easier and smoother. Better experience means more time spent using the product, resulting in more advertising revenue and subscription fees.

The research field of human-computer interaction (HCI) observes how humans interact with computers and seeks to improve this symbiosis. One of its goals is to make computers adapt to their users rather than another way around. Machine learning (ML), combined with the reaping of data, has been a keen interest in the field to accomplish this task.

The purpose of the thesis is to create a ML model that tries to predict users' smart phone usage. In practice, some information about current application combined with data from previously used ones are used to predict the category of the current app. The outline of the thesis is following. In the related work section, basics of machine learning process is introduced and some of its application in HCI. The next section focuses on the prevalent algorithms used in classification tasks. General properties of these algorithms is presented, as well as few studies that compare performances of some of these algorithms.

In the actual model building phase, first step is to extract some initial features from the raw data. Five different classification algorithms are tested with these features to get an estimate of how well they perform with this data set. After the algorithm is selected, new and bigger feature sets are extracted. There are multiple feature sets with slight variations. These feature sets are used with the chosen algorithm and analyzed how well they fare against each other.

2. RELATED WORK

2.1. Classification Process

Main idea of classification is to predict a discrete value of a some predetermined target variable. Features, other attributes related to this variable, are used to achieve this task. A feature can be measured with a numerical or a nominal value. For example, features consisting in the likes of age, height and pitch of their voice could be used to predict a person's sex.

There are a plethora of ML algorithms, with vastly different methods, to choose from for a classification task. One way to score these algorithms is to separate the data set into two distinct parts: the training set and the testing set. The purpose of these sets is quite self-explanatory. Model parameters are optimized to fit the training data, and the testing set is simply used to score the classifier.

Another popular way to score classifiers is to use a method called k-fold cross-validation. This model validation technique shows, how well results will generalize to an independent data set. Instead of having designated training and testing sets, k-fold cross-validation separates the data into k-amount of subsets. Each of these subsets will act as a test set in one round, while the other sets are used in training. The score will be the average of these rounds (Figure 1).

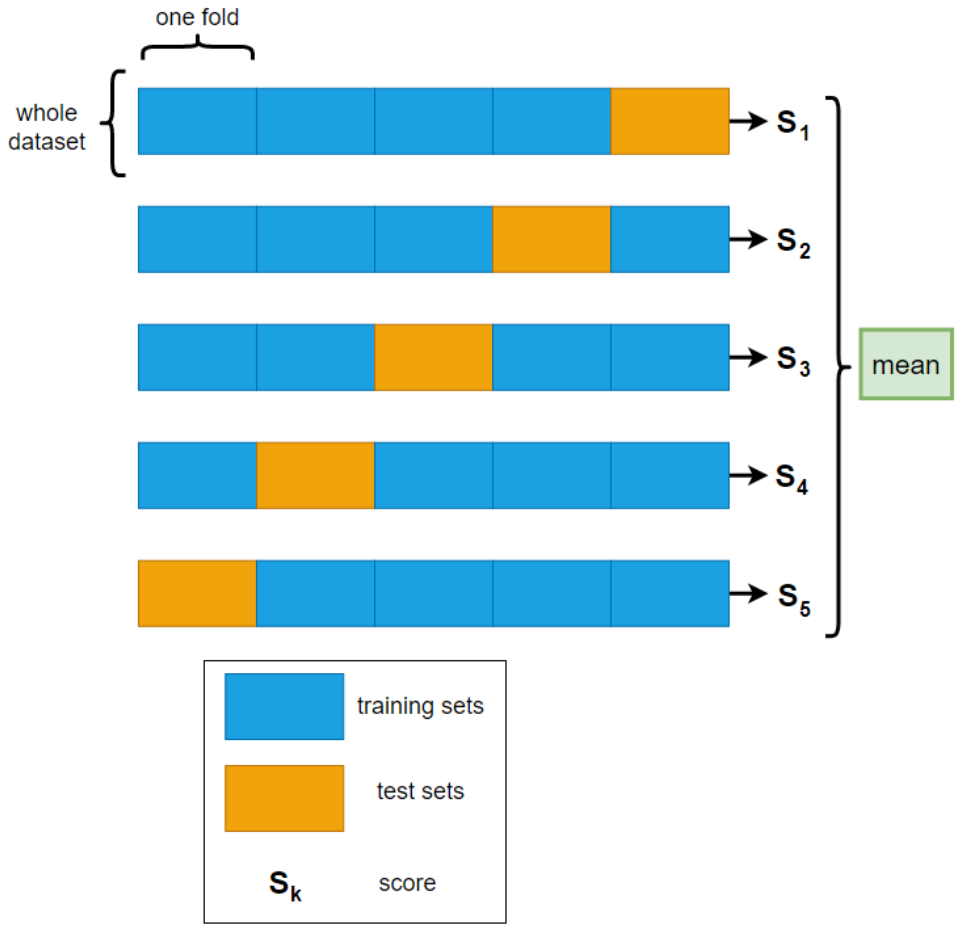

Figure 1. Representation of 5-fold cross validation

The score is typically represented in terms of accuracy, i.e. the portion of correctly classified instances among all instances. However, this approach might not reflect the validity of the predictions correctly, especially if the data has an uneven distribution of the target feature values. Other metrics are often included in the results to combat the deficiencies of accuracy. Two common ones are precision, which measures how many positive predictions are labeled correctly, and recall, which measures how many of the actual positive values were labeled correctly (Figure 2).

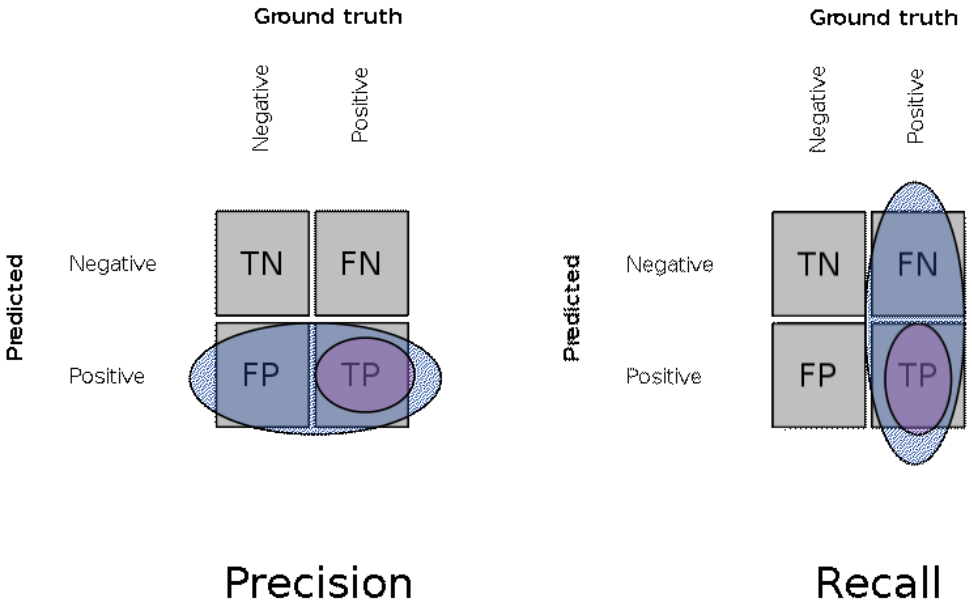

Figure 2. Illustration of precision and recall: TP means true positive and FP false positive, TN means true negative and FN false negative

2.2. Machine Learning in Human-Computer Interaction

Machine learning has been used in HCI for over 30 years now. In 1992 Dent et al. [1] created ML based calendar system. It predicted the meeting location and duration based on user's previous data. While the system was not completely automatic, it was still recognized as useful and kept in routine use.

Since then, especially smartphones have become a big source of information: virtually everyone uses them daily. They also contain a lot of possibilities for personal adjustments. Visuri et al. [2] build a model that predicts the user's preference for an incoming notification. This could be used to automatically highlight or repeat possibly important notifications. They managed to get 91.1% mean accuracy in predicting the correct preference.

Another example of the use of smartphones was done by Pielot et al. [3], who used them to detect boredom. First, they gathered data from mobile phones and built ML models to predict, if the user was bored or not. In second part of the study, the participants were offered links to online news articles. Engagement percentages with

these articles were improved by offering them to users, that were predicted to be bored. This kind of improved targeting could be beneficial to both users and advertisers.

Some medical use cases have also been experimented with. Rello et al. [4] trained ML models to predict dyslexia with a linguistic computer game. The model was able to reach 84.62% accuracy, but notably the precision, i.e. the fraction of correct predictions among all positive dyslexia predictions, was significantly smaller 63.75%. The data set used for training was also small, so the full potential of this approach could be bigger.

2.3. Classification Algorithms

2.3.1. Overview

Linear classifiers are relatively simple algorithms. They use a linear combination of the feature values to make their classification decision. The advantages of a linear classifier over a non-linear one are much shorter training and testing time. Notably, when used with the right data set, a linear classifier could get accuracy levels proportionate to a non-linear one [5]. Logistic Regression (LR) is an example of a linear classifier.

Not only are non-linear classifiers typically more accurate, some cases the data are not linearly separable, making linear classifiers ineffective [5]. Some of the most commonly used non-linear algorithms are Support Vector Machines (SVM), decision tree algorithms, Random Forest (RF) and neural nets like Multilayer Perceptron (MLP).

Each of the algorithms has its own disadvantages. SVMs need suitable data and it takes a comparatively long time to train the models. Neural nets need a lot of data to work optimally and the results are hard to interpret. On the other hand, decision tree classifiers are more interpretable and easier to use, but they can lose in accuracy for example to finely-tuned SVMs [6].

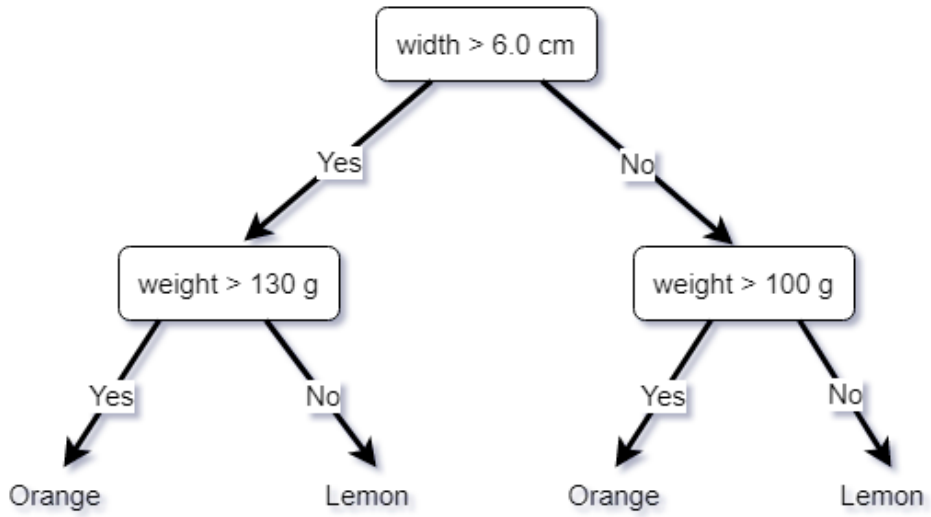

Figure 3. A binary decision tree that predicts, if a citrus fruit is a lemon or an orange

Random Forest also uses decision trees in a classification task. The algorithm takes a set amount of features randomly and builds a decision tree out of them. This process is repeated multiple times, and the prediction is obtained by the majority vote (Figure 4). RF is a relatively easy algorithm to implement: it needs little configuration and has a low risk of overfitting (fitting too closely to test data).

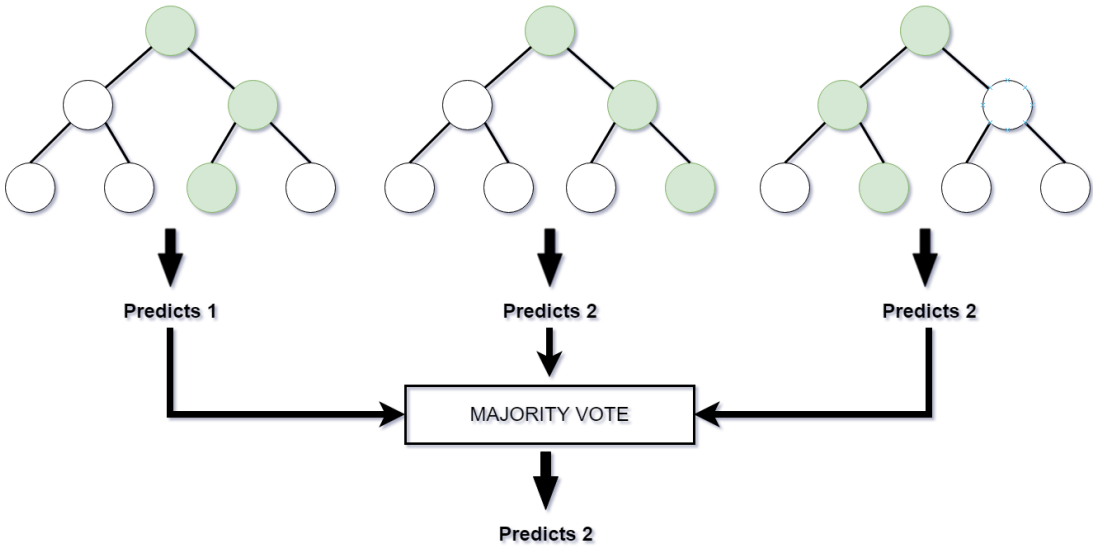

Figure 4. Representation of random forest algorithm

2.3.2. Studies

A study by Pielot et al. [3] tested the performances of LR, SVM and RF algorithms while trying to classify boredom from mobile phone data. In their case, the RF algorithm produced the best result. The same kinds of results were found by Luo et al. [7]. They gathered similar data from phones and tried to predict if the phone should be unlocked. Out of the J48 (a commonly used implementation of a decision tree algorithm), naive Bayes, RF and SVM classifiers, RF performed the best.

Narudin et al. [8] compared classification algorithms' ability to detect malware in mobile phones. They did 10-fold cross-validation test using RF, MLP, J48, Bayesian Network (BN) and K-Nearest Neighbors (KNN) algorithms. RF had the highest detection rate. In the same paper, another data set was used to rate the algorithms. This time the data was divided into training and testing sets. Surprisingly, KNN performed the best, having over ten percentage points higher recall than RF.

Krauss et al. [9] tested relatively recently adopted technology, Gradient-Boosted Trees (GBT). Accompanied by RF and deep neural network, it was used to predict stock market changes. While RF performed best with this data set, they concluded GBT to be able to yield more advantageous results than tuning-intensive deep neural networks.

3. DATA PRE-PROCESSING

This part of the paper is dedicated to the data pre-processing phase. The first section of this chapter goes over the starting data set. The second section contains the initial feature extraction, i.e. how the raw data was converted into features, which can be used by classification algorithms. Additionally, a comparison of the potential classifiers is in this section.

Lastly, ten different features sets are extracted for the final analysis. Two distinct approach are used, each having five of these features sets. The last chapter goes over these two approaches.

3.1. Data Overview

The raw data used in this thesis is from a 2022 study by Peltonen et al. [10]. The study followed smartphone app usage of the participants, documenting each time the foreground application was changed. The recorded data were stored in a comma-separated values (CSV) file. The relevant info in the file included:

- name of the application, e.g. "Chrome"
- identifier of the participant, "P1" for first participant, "P2" for the second one...
- timestamp in Unix time and in milliseconds
- usage time in milliseconds
- date and time in formats "YYYY-MM-DD" and "YYYY-MM-DD HH:MM:SS"
- category which the app was identified in Google Play Store

Starting data set contained over two million instances of data. There were 35 categories, with varying preciseness. For example, the "communication" category contained all text messaging apps as well as internet browsers, but gaming-related applications were divided into nine different categories. This made the distribution of instances quite uneven. You can see, how many instances the most common categories had in the table 1.

The original data contained a category called "other", which covered over half of the data. These instances were mostly from applications related to the operating system of the phone: for example, apps called "System UI" and "Ambient Display". Generally, the users would not open these apps themselves: the phone would do it automatically instead. Additionally, it seems like every time the home screen was entered, either by changing to another app or simply closing the current one, an instance of the operating system's app was added. As a result, if the previous category was anything else than "other", the following category would most of the time be "other". This would make the classification quite easy. For these reasons, instances with the category "other" were not used, changing the unique category count to 34.

Total amount of participants was 79. In the final data set, there was 819 808 instances. This makes the average amount of instances per a participant to a bit over ten

thousand. There was a big variance in the distribution: the most amount of instances for a participant was 65 575 and the least was just 190.

Table 1. Distribution of the instances

Category	Number of instances	Percentage of the data
Communication	317 272	38.70 %
Tools	176 765	21.56 %
Social	155 243	18.94 %
Productivity	65 951	8.04 %
Music and audio	20 738	2.52 %
Lifestyle	18 172	2.21 %
News and magazines	14 937	1.82 %
Media and video	14 317	1.75 %
Travel and local	7 157	0.87 %
Entertainment	6 963	0.85 %
Finance	6 040	0.74 %
...	...	...

3.2. Model Selection

3.2.1. Features

At first, only five features were extracted from the data. These features were extracted to a new CSV file and used to test different ML classifiers. The best-performing classifier will be used with more detailed data. You can see the features in table 2.

Table 2. List of the preliminary features

usage_time	time the app was used
hour_of_the_day	hour of the day the app was opened
last_category	category of the last used app
time_since_last_app	time since the previous app was used
is_weekend	was the app opened on Friday, Saturday or Sunday

Usage_time and hour_of_the_day were obvious features to add. Intuitively, apps belonging to one category should have a tendency to be used at different times of the day than apps from another category. Same goes for the apps' usage time. Hour_of_the_day were recorded as integer values from 0 to 23, which is not optimal, since the hour of the day is cyclical. The algorithms will not interpret values of 0 and 23 to be one hour apart. Fortunately, a only small part of the data was recorded just after midnight, so this should not have a lot of impact (Figure 5). Both of these features could just be copied from the original data set.

The next feature in the list is last_category. Just by looking at the datasheet, you could see a trend, that the same apps were recorded multiple times in a row. This was

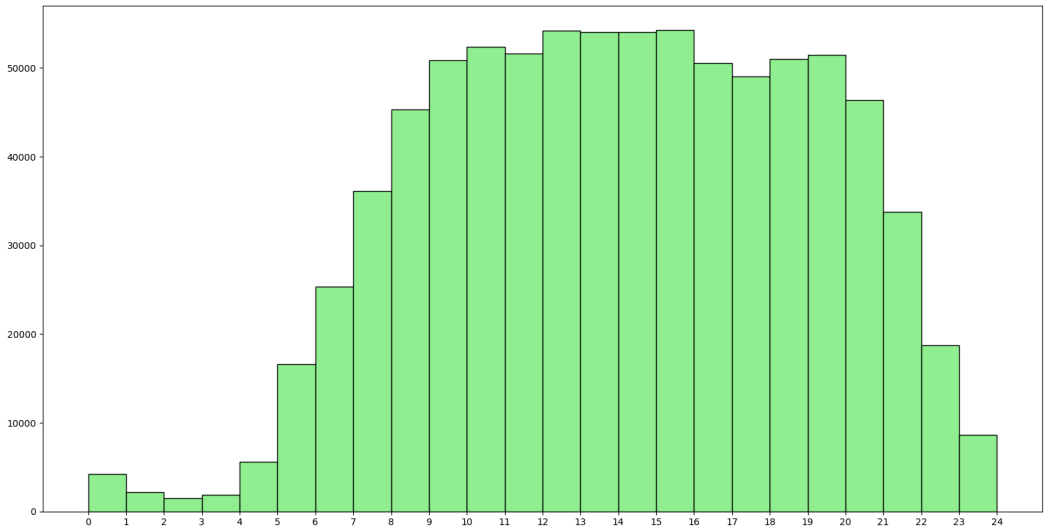

Figure 5. Histogram of hour_of_the_day feature values

partly the aftermath of the removal of the "other"-category: when a phone's screen is turned off during the use of an app, the app will be relisted in the database upon opening. Now, multiple instances of the same application are recorded in a row. Additionally, some of the apps will have a habit of being opened one after another. For example, shopping apps might open a mobile payment application to finish the transaction. Unlike the last two features, which simply took values from the same line of the CSV file, this feature needs to fetch the information from the previous line. As a result, the first instance of each participant do not have value for the last_category. In these scenarios, the value was set to "NOT_AVAILABLE".

Similarly to the last_category, obtaining time_since_last_app feature values required fetching information from the previous instance. The final value was obtained by simply subtracting the timestamp of the previous instance from the timestamp of the current instance. This value was set to zero for the first instance of each participant.

Maybe the most imaginative feature is the is_weekend feature. Instead of using numerical values from 0 to 6 to represent the days of the week, a binary feature that just shows the information about weekend was included. Application usage on weekdays should be quite similar, but a shift on weekends with different activities and more free time is reasonable to expect. Furthermore, by choosing a binary value instead of a numerical, or even a nominal value, less variance is incorporated while using the feature. Python's datetime[1] library's "weekday()" method was used to extract this feature from the date information.

3.2.2. Comparison of Algorithms

Having extracted the initial features, the testing of potential ML classifiers could begin. RF, KNN, MLP, LR and decision tree classifiers were scored by 10-fold cross-

[1] https://docs.python.org/3/library/datetime.html

validation. Scikit-learn[2] library's implementations of these classifiers were used. The best accuracy scores for each classifier can be seen in table 3. Worth noting, that while some parameter tuning was done manually, complete hyperparameter optimization was not done, so the accuracies might not be on their upper limit. That being said, the results should be good enough for accurate comparison.

Table 3. Accuracies of the classifiers

Random Forest	0.4979
decision tree classifier	0.4954
K-Nearest Neighbors	0.4217
Multilayer Perceptron	0.4078
Logistic Regression	0.3994

Following the trend from other studies, RF had the best score. It classified the category correctly almost half of the time. Notably, the accuracy of the decision tree classifier was almost as good as RF: it was only 0.25 percentage points smaller. The accuracies for KNN and MLP were clearly lower. Unsurprisingly, LR had the lowest accuracy of the algorithms with a 0.3994% correct classification rate. Still, it was not a lot behind the other classifiers. Since RF had the best score, it will be used in the classification.

3.3. Feature Sets

In the initial data set, only information from the current and the previously used application is used. To get absolutely everything out of the data, the algorithm should also take into consideration more prior instances: the feature values could have patterns that occur when inspecting more instances together. Two apparent ways to do this is to either take a set amount of prior instances or take every instance that were recorded in a time frame.

In the first approach, amount of prior instances used would be predetermined constant. For example, use data from the last five recorded applications. In the latter approach, the used instances would be restricted by a time frame, e.g. take the data, that were recorded in last two minutes. Five variations with different depths of both of these approaches will be used to see how well each of them work.

The number of features will significantly go up. Five features from every used prior instance are added:

- category
- usage_time
- hour_of_the_day
- is_weekend
- time_since_original_app

[2] scikit-learn.org

The features from the original instance will remain the same. The only exception is the time_since_last_app feature, which will be removed since it would have the same values as time_since_original_app from the previous instance.

3.3.1. Instance Restrained Approach

This approach uses information from predetermined amount of prior instances. A CSV data file was created for each feature set. Creating the CSV data files for this approach was quite straightforward: similarly to some of the initial features, getting the values was a simple task of fetching information from previous instances. Cases where information was not available, i.e. when there was not enough prior instances, the feature values were left empty.

Which amounts of previous instances should be used to get the best representation? Using the whole data set to test potential values would be inconvenient, as the time to create the CSV file for the features and train the algorithm with a feature set is quite long. More practical method was to test the milestones with part of the data. In this circumstance, data from one of participant was used. This should provide a rough idea, which milestones would most illustratively show the accuracy changes, when adding more prior instances. These five milestone values ended up being 1, 2, 5, 10 and 20 previous instances.

Table 4. A snippet of the final feature data

category	usage_time	hour_of_ the_day	is_weekend	category_1	usage_time_1	...
TOOLS	76934	11	0	TOOLS	47891	...
TOOLS	47891	11	0	TOOLS	15807	...
SOCIAL	1856	11	0	TOOLS	76934	...
TOOLS	2338	11	0	SOCIAL	1856	...
...	...	...	...	...	...	

3.3.2. Time Frame Restrained Approach

As already stated, the other approach is to take all the prior instances, which were recorded in a predetermined time frame. Creating the feature files for this method was slightly more complicated to achieve. Obviously, amount of instances in a time frame is not constant. Thus, in order to get the right amount of possible features in the CSV files, the maximum amount recorded apps in the time frame has to be calculated. This was achieved by looping through the instances and keeping track of the biggest value for this attribute. Results can be found in table 5.

After this step, the protocol was very similar to the previous approach. The only exception was that instead of retrieving data from the predetermined amount of previous instances, the information was fetched until the time window was exceeded. Also, the same testing method for the milestones was applied. The used time frames are 1, 2, 5, 10 and 20 minutes.

Table 5. Maximum amount of instances in the time frames

1 min	41
2 min	52
5 min	80
10 min	99
20 min	148

3.3.3. Individual Models

The data set obviously had recorded the participant identifier. They could be used to create models with data from just one user. People have different patterns of using their smart phones and this kind of separation could use this aspect to improve accuracy. To get a better idea how the classification would perform with just one person, these individual models will be built and scored. In practice, this is done by creating their own model for each of the 79 participants, scoring it with 10-fold cross-validation and then taking the average of the scores.

To summarize, ten versions of the general models are scored: five with instance-restrained models and five time-frame-restrained models. In addition, average of individual models will be scored with 10 different variations. Like with the general models, five of them will be with instance restrained models and five-time frame restrained models. The same instance and time frame milestones will be used to allow proper comparison.

4. RESULTS

This chapter goes over the results from all of the models: they are scored and the best ones are highlighted. In addition, features are scored with one of the models.

4.1. Results of the Different Feature Sets

The ten feature sets were scored with 10-fold cross-validation. Most of the RF classifier's parameters were not changed, i.e. they had the default values assigned by scikit-learn. Still, two parameters were tweaked:

- "n_estimators = 100", determines the number of decision trees the algorithm uses to make a prediction
- "max_depth = 20", determine the decision trees' maximum depth, i.e. how many nodes they can have

Again, these parameter values are just rough estimates to balance the accuracy and computing time. It is not possible to do unified parameter optimization for all ten feature sets, and doing them individually would be impractical considering the amount of time and computing power needed. The parameters can have a significant impact on the accuracy, and it is possible, that different feature sets would have performed better relative to others after complete parameter optimization.

The results can be seen in table 6 with the best results for both general and individual models highlighted. Using a set amount of previous instances yielded noticeably better results. The best accuracy was achieved by including the previous two instances. This feature set was able to add almost 10 percentage points to the accuracy of the initial model. After that, adding information from more instances starts to lessen the accuracy.

The time frame approach was less impressive. Accuracies of the general models are only slightly bigger than the accuracy of the initial RF model without the extra instances: even the best performing-feature set was not able to add one percentage point to the original accuracy. When a 20-minute time frame was used, the accuracy was actually smaller. Time frames somewhere between two to five minutes seem to give the best results, but the differences are not big.

Unsurprisingly, the averages of participants' individual models are more accurate than general models with both approaches. Same as the general models, the model using the two previous instances has the highest accuracy.

4.2. Feature Analysis

The point of feature analysis is to determine how much each feature contributes to the final prediction. There is multiple ways to score the features, and different algorithms can employ different methods. In this section, mean decrease in impurity (MDI), also called gini importance, is used. This method scores a feature by removing it from the pool of features and calculating how much the prediction accuracy decreases.

Table 6. Accuracies of the models

number of previous instances	1	2	5	10	20
general models	0.5452	**0.5945**	0.5874	0.5677	0.5440
average of individual models	0.5981	**0.6337**	0.6187	0.5960	0.5674
time frame (min)	1	2	5	10	20
general models	0.5037	0.5087	0.5087	0.5054	0.4932
average of individual models	0.5399	0.5501	**0.5523**	0.5478	0.5338

Since the accuracies were highest with the model that uses two previous instances, it will also be used when determining the feature importance scores. While the average of individual models did have a higher accuracy, the importance scores will differ for each model based on individual variance. For this reason, the general model is used. Since 10-fold cross-validation was used to score the models, it will be also used for this task. As a result, the features will be scored ten times, once for each fold. The average of these score is shown on table 7.

Table 7. Feature importance scores

category_2	category of the app two instances ago	0.1638
category_1	category of the previously used app	0.1544
usage_time_1	usage time of the previously used app	0.1319
usage_time	usage time of the current app	0.1222
time_since_original_app_1	time difference of current and previous app	0.1190
usage_time_2	usage time of the app two instances ago	0.0903
time_since_original_app_2	time difference of current app and the app two instances ago	0.0884
hour_of_the_day_2	the hour which app two instances ago was opened	0.0361
hour_of_the_day	the hour which current app was opened	0.0357
hour_of_the_day_1	the hour which previous app was opened	0.0352
is_weekend_2	is the app two instances ago opened on weekend	0.0077
is_weekend	is the current app opened on a weekend	0.0077
is_weekend_1	is the previously used app opened on weekend	0.0076

As could be expected, the category features were the most important. Surprisingly, category_2, which shows the category of the second previous app, is more important

than category 1, which shows the category of the previous app. The main reason might be the alternation of two applications, which still occurred even after the removal of the "other" category. Most notably, the keyboard applications were guilty of causing this.

The next important set of features was the usage times. Again quite unintuitively, the usage time of the previous application is more important than the usage time of the target application. A similar trend also continued with the two lowest scored-features: hour_of_the_day and is_weekend. In those cases', the importance of features two applications ago exceeded the others, although the difference is very slim. Furthermore, most of the cases hour of the day, and obviously the state of weekend, were the same in all three instances, so there should not be a lot of differences in importance.

5. DISCUSSION

The motive for this thesis was to build a machine learning model, that predicts smartphone usage by identifying categories of mobile apps. One usability for a model like this could be an application recommendation system. Phones would be able to suggest apps to use or have shortcuts on the home screen with a better conversion rates.

The best results were achieved by using two prior instances when extracting the features. This number would not be universally applicable: this number could be larger with an additional data. The more prior the instance, the less relevant the features are to the current application. Therefore, more data is needed to recognize accurate patterns. A model based on years of data could make more use of extra information from more prior instances. Also, a different algorithm could find obscure patterns better. Something like a deep neural network might get better results with more features, granted that there is a sufficient amount of data.

The time frame-restricted models performed worse than its counterpart. Since the number of available features for instances differed, most of the extra features did not have actual values. As a result, the algorithm gains next to no information when comparing the values of these features. Moreover, the extra features' values can come from vastly different time spots. Meaning, category_5 could have happened thirty seconds ago for one instance and five minutes ago for another one. The algorithm is not discrete enough to understand the time difference between these features.

A more elegant, and perhaps more effective, design would have been to separate the time frames into smaller sections and average the information in these smaller time frames into features. For example, separate a five-minute time frame into five, one-minute sections. Then take all the instances in a single one-minute section and take the average of the values for a combined feature. This way most of the features get an actual value, and the information is from the same time spot.

As shown in the result section, the highest accuracies were achieved using the individual models, i.e. only the data from one participant, which was expected. Not only do users have different habits, but they do not generally use apps from all of the categories. Therefore, the pool of possible categories is smaller making the prediction easier. It is important to differentiate the applications of these testing methods. The general method would demonstrate better how the approach would work on users with no previous data. In contrast, the results of the individual models indicate how the approach would work with users with prior data.

Another variation to test how the model would work on a user with little or no previous data would be into separate the participant to either a training set or a testing set. This method would have actually been superior to the 10-fold cross-validation with all of the data, since no data from the tested participants would have been used in the training stage.

6. REFERENCES

[1] Dent L., Boticariol J., McDermott J., Mitchell T. & Zabowski D. (1992) A personal learning apprentice. AAAI-92 Proceedings , pp. 96–103.

[2] Visuri A., van Berkel N., Okoshi T., Goncalves J. & Kostakos V. (2019) Understanding smartphone notifications' user interactions and content importance. International Journal of Human-Computer Studies 128, pp. 72–85.

[3] Pielot M., Dingler T., San Pedro J. & Oliver N. (2015) When attention is not scarce - detecting boredom from mobile phone usage. UbiComp '15: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing , pp. 825–836.

[4] Rello L., Romero E., Rauschenberger M., Ali A., Williams K., Bigham J.P. & White N.C. (2018) Screening dyslexia for english using hci measures and machine learning. DH '18: Proceedings of the 2018 International Conference on Digital Health , pp. 80–84.

[5] Yuan G., Ho C. & Lin C. (2012) Recent advances of large-scale linear classification. Proceedings of the IEEE 100(9) , pp. 2584–2603.

[6] Geurts P., Irrthum A. & Wehenkel L. (2009) Supervised learning with decision tree-based methods in computational and systems biology. Molecular BioSystems 5(12) , pp. 1593–1605.

[7] Luo C., Visuri A., Klakegg S., van Berkel N., Sarsenbayeva Z., Möttönen A., Goncalves J., Anagnostopoulos T., Ferreira D., Flores H., Velloso E. & Kostakos V. (2018) Energy-efficient prediction of smartphone unlocking. Personal and Ubiquitous Computing 23 , pp. 159–177.

[8] Narudin F.A., Feizollah A., Anuar N.B. & Gani A. (2016) Evaluation of machine learning classifiers for mobile malware detection. Soft Computing 20(1) , pp. 343–357.

[9] Krauss C., Do X.A. & Huck N. (2017) Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. European Journal of Operational Research 259(2) , pp. 689–702.

[10] Peltonen E., Sharmila P., Kuosmanen E., Koskimäki H. & Visuri A. (2022) The impact of smartphone usage on circadian cycles: A case study with wearable ring. 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) , pp. 148–154.