

UNIVERSITY OF ZAGREB  
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

MASTER THESIS No. 655  
**INFERRING PRESENCE STATUS ON  
MOBILE DEVICES**

Mia Primorac

Zagreb, June 2014.

UNIVERSITY OF ZAGREB  
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING  
MASTER THESIS COMMITTEE

Zagreb, March 13th, 2014

## MASTER THESIS ASSIGNMENT No. 655

Student: **Mia Primorac (0036450884)**

Study: Computing

Profile: Computer Science

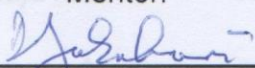
Title: **Inferring Presence Status on Mobile Devices**

Description:

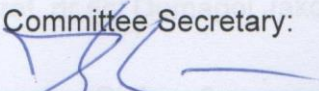
Describe the problem of inferring presence status on mobile devices without explicit user interaction. Build the environment for arbitrary machine learning method's examination on the problem of inferring presence status. Test the efficiency of genetic programming method on the given problem. Especially focus on methods based on building decision trees using available features set and properties of training set. Build classification system using a set of classifiers and ensemble learning. Compare the efficiency of built methods with other machine learning methods concerning given conditions and restrictions on mobile devices. In addition, attach source code of implemented programs, final results with necessary explanations and used references.

Thesis submitting date: June 30th, 2014

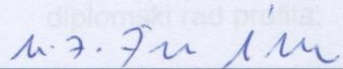
Mentor:

  
Prof. Domagoj Jakobović, PhD

Committee Secretary:

  
Prof. Tomislav Hrkać, PhD

Committee Chair:

  
Prof. Siniša Srbljić, PhD



Zagreb, 13. ožujka 2014.

## DIPLOMSKI ZADATAK br. 655

Pristupnik: **Mia Primorac**  
Studij: Računarstvo  
Profil: Računarska znanost

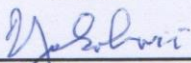
Zadatak: **Određivanje statusa korisnika mobilnih uređaja**

### Opis zadatka:

Opisati problem određivanja statusa korisnika mobilnih uređaja bez izravnog određivanja stanja od strane korisnika. Izraditi okruženje za ispitivanje postupaka određivanja statusa korisnika proizvoljnom metodom strojnog učenja. Ispitati učinkovitost postupka određivanja statusa temeljenog na genetskom programiranju. Posebnu pažnju posvetiti metodama izgradnje stabla odluke koristeći dostupan skup značajki i svojstva skupa za učenje. Ostvariti sustav klasifikacije korištenjem skupa klasifikatora i odgovarajućeg mehanizma glasovanja. Usporediti učinkovitost ostvarenih postupaka s drugim metodama strojnog učenja s obzirom na uvjete korištenja na mobilnim uređajima. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 14. ožujka 2014.  
Rok za predaju rada: 30. lipnja 2014.

Mentor:

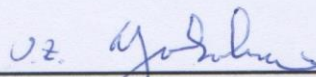


Izv.prof. dr.sc. Domagoj Jakobović

Djelovođa:

Doc. dr.sc. Tomislav Hrkać

Predsjednik odbora za  
diplomski rad profila:



Prof. dr.sc. Siniša Srbljić



## Contents

Introduction .....	1
1. Related Work.....	2
1.1. Pervasive and ubiquitous computing .....	2
1.2. Inferring User Activities .....	3
1.2.1. Activity Recognition Systems .....	3
1.3. Automatically Inferring Presence Status on Smartphones .....	5
1.4. Implementations in IM Applications .....	7
2. Applicable Machine Learning Methods .....	8
2.1. Categorization of Machine Learning Problems .....	8
2.2. Genetic Programming.....	9
2.2.1. Genetic Programming in Classification.....	12
2.2.2. GP for Extracting Decision Trees.....	13
2.2.3. GP for Learning Rule-Based Systems .....	15
2.2.4. GP for Learning Discriminant Functions .....	16
2.3. Clustering .....	17
2.3.1. K-means Algorithm .....	17
2.4. Decision Trees .....	19
2.5. K-nearest neighbors .....	20
2.6. Ensemble Learning .....	21
2.6.1. Bagging.....	21
2.6.2. Boosting.....	22
2.7. Naïve Bayes.....	23
2.8. Feature selection .....	24
2.9. Incremental learning .....	25
2.9.1. Learn++ .....	26

3.	Inferring Presence Status .....	28
3.1.	Adaptation of the Mobile Data Challenge Dataset .....	28
3.2.	Implementation of GP algorithms .....	30
3.2.1.	Implementation of GP for Learning Discriminant Functions.....	31
3.2.2.	Implementation of GP for Learning Decision Trees .....	33
3.2.3.	Implementation of Bagging Using GP .....	35
3.2.4.	The Comparison of Different GP Variants.....	35
3.3.	Implementation of AdaBoost .....	36
3.4.	Implementation of K-means Clustering .....	36
3.5.	Implementation of Decision Trees .....	37
3.6.	Implementation of K-Nearest Neighbors .....	39
3.7.	Implementation of Naïve Bayes .....	39
4.	Results and Comments .....	40
4.1.	Accuracy and Performance.....	40
4.1.1.	GP algorithms .....	40
4.1.2.	Non-GP Algorithms.....	42
4.1.3.	Results on Merged Datasets .....	45
4.2.	Feature Selection and Energy Efficiency .....	47
4.3.	Comments.....	49
	Conclusion .....	52
	References .....	53
	Summary.....	57
	Inferring Presence Status on Mobile Devices.....	57
	Sažetak.....	58
	Određivanje statusa korisnika mobilnih uređaja .....	58
	Abbreviations .....	59

# Introduction

During the past decade smartphones have evolved from simple communication devices to irreplaceable personal assistants with a variety of sophisticated sensors. The data collected from those devices is numerous and it can reveal a lot about user's behavior, activities and habits. One possible application of data collected from the smartphones is inferring presence status which will be discussed in this thesis.

In terms of computer and telecommunications networks, presence information is a status indicator that conveys the willingness and ability of a potential communication partner to communicate. The presence information is collected from the user (presentity) and it is stored and distributed via network service called the presence service to user's subscribers called watchers. Reliable information about user's availability is needed for many real-time communication services, mainly in instant messaging and VoIP services, and presence service is integrated into many of them. The most common presence states in IM services are "Available", "Unavailable", "Busy" etc.

The problem of inferring presence status combines several trade-offs. The most accurate presence service would be the one where presence status is 1) explicitly changed by the user and 2) it is up-to-date after every single action. The first condition causes frequent and annoying user intervention, while the second condition causes too frequent changes of presence status. It can cause an overload on presence server and watchers which leads to both quality of real-time service impairment and draining out battery on watcher's mobile devices. The ideal inferring method without user's interaction should be accurate, energy efficient and it should be able to accommodate to a specific user.

In first chapter, the related work, mainly from activity recognition, is described and some ideas applicable to the problem are highlighted. The second chapter gives the theoretical background of machine learning algorithms which were applied to the problem. The platform independent implementation of those algorithms is given in third chapter as well as the description of the used dataset and features. The last, fourth chapter, contains the accuracy and performance results for each algorithm and the results of feature selection. In the end, the possible application of each relevant algorithm is given together with plans for the future work and comments.

# 1. Related Work

In this chapter the problem of inferring user status is classified to adequate research area and core concepts and issues are discussed. Some solutions that partly solve problems of energy consuming will be presented as well as related work solving more general problem of inferring user activities and behavior. Many ideas applied on different anticipatory mobile computing are also highly applicable to inferring user status problem and that many of them used the same data set mentioned in 3.1. Further on, some existing implementations in commercial Instant Messaging (IM) applications will be presented and analyzed.

## 1.1. Pervasive and ubiquitous computing

The problem of inferring presence status is classified as a mobile computing problem. Mobile computing is a subtype of ubiquitous (omnipresent) computing which is closely related to pervasive computing. There is a very thin line between those research areas but what is important is the common core concept in all of them. Ubiquitous and pervasive computing is not any more strictly attached to personal computer; it can appear in any location on all kinds of devices starting from smartphones and tablets all the way to fridges and glasses. Most of those devices have issue with battery capacity, hence energy efficiency is a very important aspect in those popular research fields.

Smartphone is the most common mobile device that 1.4 billion people used at the end of year 2013 and its number is about to pass the number of PCs, based on [4]. They have evolved from simple communication devices to perceptive devices capable of inferring context around them. The research potential of personal data generated on smartphones is not yet achieved but it heading in a good way. Activity recognition, behavior prediction, location prediction are just some of the related problems.

The problem of security is often mentioned in the context of pervasive computing. The loss of privacy and distribution of personal data is concerning many people. It is the most common objection to development of the services that are highly involved in people's lives. The solution is a trade-off between the amount of information exposed and the needs of service. In the case of activity recognition, inferring user status is providing less



personal information than for instance location prediction. An accurate service that provides general presence information without many details is highly applicable in many other services and even in corporate environments among a bigger group of people, unlike services that reveal lots of private data.

## **1.2. Inferring User Activities**

Inferring presence status can be considered as a very simple way of inferring user activities which has tried to involve into all the aspects of human lives. Some of the activity recognition topics mentioned in [7] are inferring whether the user is jogging, commuting to work or maybe sleeping. Further on, inferring the user emotions, for instance anger and happiness, has also been analyzed, as well as discovering the most important places in everyday life using only the cellular identification. All that becomes possible thanks to a rich source of sensors available on every modern smartphone and adequate use of machine learning algorithms that will briefly be mentioned in this chapter.

### **1.2.1. Activity Recognition Systems**

Many modern research challenges related to anticipatory computing are presented in [7] are solved using interesting features and techniques and here will be given some of them.

In [5] and [8] a boosted ensemble of classifiers is used. UbiFit Garden System [5] is based on gadgets that measure user activity during the day. The goal is to infer their activities and to encourage regular physical activity by showing personal messages on mobile display. It consists of a fitness device, an interactive application and a glanceable display. An extra effort was taken to collect the data from users. UbiFit Garden uses the Mobile Sensing Platform [6]. MSP is pager-sized battery powered computer with several sensors: 3D accelerometer, barometer, humidity, visible and infrared light, temperature, microphone and compass. It infers physical activities in real time using a set of boosted decision stump classifiers that have been trained to infer walking, running, cycling, using an elliptical trainer and using a stair machine. [5] Only 3D accelerometer and barometer are used to infer those actions.

In [8] boosting algorithm AdaBoost is successfully applied to population scale activity recognition from a large amount of data.

In [5], [6] and [8] more complex activities were inferred on more detailed datasets, but some ideas can be adopted for inferring presence status. Boosted ensemble of weak

learners is often proved to be an accurate and extremely fast classifier, not only in activity recognition, but also in pattern recognition. What should be avoided in an energy efficient solution on smartphones, and it was used in [5] and [8], is using the accelerometer which consumes lots of energy.

Identifying frequently visited locations is also among common topics. The data collected in [9] is classified using online nearest neighbor classification. In feature extraction they were using light sensor and microphone which is not applicable in this thesis. However, k-NN remains an interesting option, and especially 1-NN that was used in [9] to assure very fast online classification.

Predicting human interruptibility is highly related with predicting presence status because by predicting user presence we predict user interruptibility, which is exactly the opposite term. While inferring usage context on a smartphone or any other mobile device remains difficult due to the sheer amount and quality of highly user-specific data, predicting is even more so. [7]

Predicting user interruptibility with sensors described in [10] was approached from human-computer interaction point of view using a priori social conventions of whether or not an interruption is appropriate in a given moment. Features were extracted from the video records of people working in their offices. Subjects were prompted for interruptibility self-reports at random, but controlled, intervals, averaging two prompts per hour. Subject activities were recognized from video and counted. The position of the subject in the room, number of guests in the room, drinking, eating and writing are only some of the recorded actions that are later used as features. Estimator subjects were enrolled and they were supposed to evaluate the recordings as if they were walking into that situation and needed to decide how interruptible the video subject was prior to deciding whether to interrupt the video subject. After the experiments the results from video subjects were compared with an opinion of estimator subjects and data from recordings is labeled with a certain precision which is high if a subject on the video answered the same as estimator subjects. Machine learning models used in this work are decision trees and naïve Bayes. The results with decision trees were fulfilling the expectations and higher accuracy was achieved than with naïve Bayes. An interesting approach in this work, that can be applied later, is collecting prompts from two groups of users and using them for labeling the collected data with certain precision. Decision trees and naïve Bayes remain as potential techniques for inferring presence status.

In [12] Nokia Mobile Data Challenge dataset was analyzed for predicting human movement and detecting social groups like friends and acquaintances. They have shown that by means of multivariate nonlinear predictors it is possible to exploit mobility data of friends in order to improve user movement forecasting. [12] The dataset they were using is composed of information related to 39 users including GPS traces, telephone numbers, call, SMS and WLAN history and Bluetooth. GPS traces are used to analyze the movement of the users, while colocation and number of phone calls between individuals are used to determine social interactions.

Other work based on Nokia MDC dataset and published on Nokia Mobile Data Challenge Workshop [13] is focused on semantic place prediction, location prediction, human behavior and interactions prediction etc.

Cloud computing gives an additional opportunity for saving the energy on mobile devices. An interesting use of cloud computing is introduced in [14]. The amount of energy spent on processing GPS data is reduced by uploading raw signals to the cloud for processing, relieving a battery limited device from the burden of computation. The drawback is the increased communication which consumes both device and network resources, but the amount of energy spent on mobile devices is still decreased by a significant percentage. This principle can be applied whenever computational resources spend far more energy than uploading the necessary data to cloud, for instance to offline training of GP classifiers. Most of the related work mentioned until now was either using a set of features that is not applicable to inferring presence status either the processing and learning methods were far beyond our needs. However, some ideas and machine learning models are applicable to inferring presence status. Online inferring of user activities, light machine learning models, collecting direct information from users and the inclusion of cloud computing are some of them.

### **1.3. Automatically Inferring Presence Status on Smartphones**

This thesis is supposed to continue the work from the paper [1] presented in this chapter. The same Nokia MDC dataset presented in chapter 3.1 was used. The main problem with inferring presence status was the lack of ground truth in the data since the given dataset does not contain the information about the presence statuses. The ground truth had to somehow be estimated and it was done with an automaton depicted in Fig. 1-1.

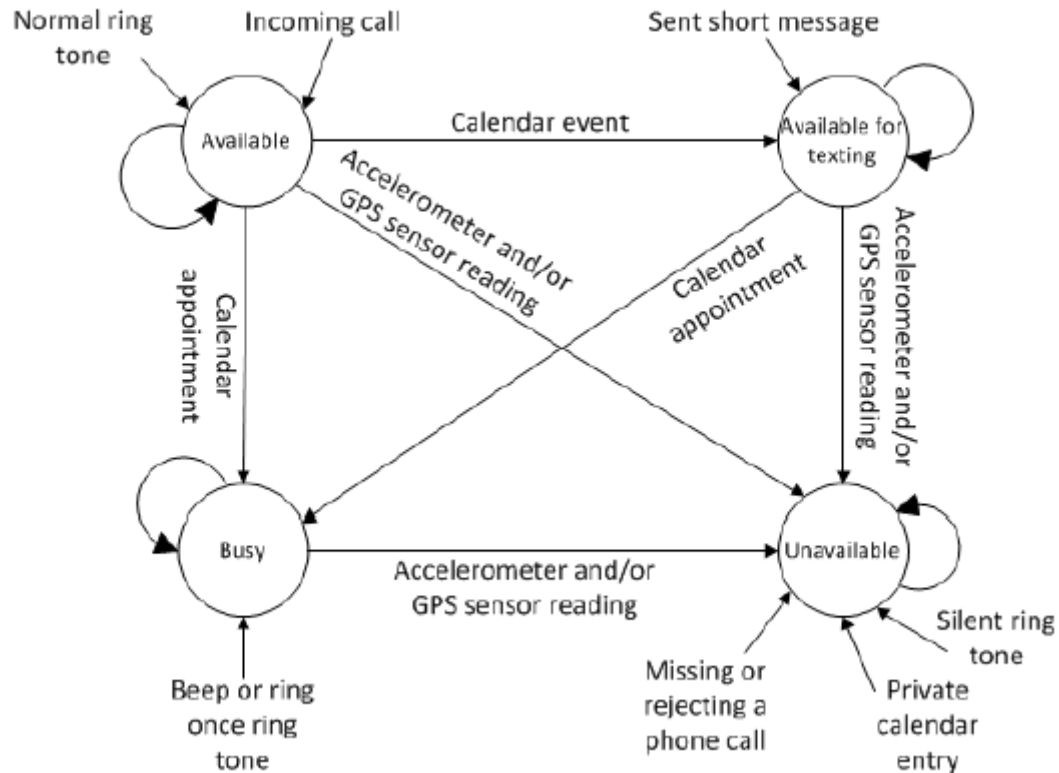


Figure 1-1 An automaton for user presence status assignment [1]

The four states of an automaton are Available, Unavailable, Busy and Available for Texting and they correspond to four possible presence statuses. The automaton is pretty straightforward. The presence status is set to Available after answering a phone call or setting the ring tone to normal. State Unavailable is reached whenever user misses or rejects a phone call, has a private entry in a calendar or sets the ring tone to silent. Busy is a special state of being unavailable and it can be reached when a user sets the ring tone to beep or ring once. This arc is problematic because on today's smartphones this special ring tone types as beep, ring once and ascending do not exist. Further on, Available for Texting is a special case of being Available and presence state is switched to it when a user sends a short message while the previous state is either Busy or Unavailable. This makes the previous user state as one of the most important features.

The data for each of 38 users was accumulated inside five minutes time windows in order to save the battery on subscribed mobile devices and to reduce the messaging load on presence server as well as the network traffic.

The initial set of features from MDC dataset is reduced to: call type (voice call or SMS), call direction (incoming, outgoing or missed), calendar event class (public or private), event type (appointment or event), event status (tentative or confirmed), ring type (normal,

ascending, ring once, beep or silent), accelerometer (slow, moving) and GSM cell identifier.

For inferring the presence status (which has already been defined using the mentioned automaton) the logistic regression was chosen. It is not very clear why that model was chosen or would some other model outperform it. The best achieved average accuracy on five-fold cross validation is 85%. The important fact is that with only two features, ring tone and GSM cell identifier, the accuracy does not deviate too much except for a few users. It is important information because these features are not very energy consuming unlike accelerator or GPS.

In this thesis the goal is to outperform the results presented in [1] and to compare different machine learning classification algorithms tested on the problem. Apart from accuracy, the time execution and energy consumption are also very important criteria if the final goal is to infer presence status directly on smartphones.

## **1.4. Implementations in IM Applications**

Today, the most popular IM applications in the world still do not offer an automatic presence service without user intervention. The question whether that is necessary and among which group of users it is applicable, remains. For now, the only information available to user contacts are when he/she was last seen online (WhatsApp, Facebook Chat, Viber,...) or the last presence status explicitly changed by user or by his/her absence (Skype, Google+ Hangouts,...). In the second case, user can choose between several statuses like Online/Available, Do Not Disturb/Busy or Away/Idle.

On ex-Google Talk status is automatically changed from Available to Idle if user is away from the device for more than 10 minutes [15].



## 2. Applicable Machine Learning Methods

In this chapter used machine learning methods, for both supervised and unsupervised learning will be presented. Genetic programming and its application to classification will be given in details. Theoretical implementation independent explanations will be given as well as basic components for every algorithm that are used to make a systematic categorization of the existing algorithms.

### 2.1. Categorization of Machine Learning Problems

The main approaches distinguished in machine learning are supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, we have a training set with training examples in the form of (*input*, *output*) and the goal is to infer a relation between the set of inputs and the set of outputs. If output is a discrete value then the problem is a classification problem and if output is a continuous value it is a regression problem [18].

In unsupervised learning, the data is unlabeled. The goal of unsupervised learning is to find the intrinsic structure, relations or affinities present in data. The most common unsupervised learning tasks are clustering and association discovery.

Reinforcement learning is based on reasoning from a complete model of environment. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them [29].

Semi-supervised learning is a technique that falls between supervised and unsupervised learning with a small completely labeled training data and a bigger unlabeled training data. The common assumption in semi-supervised learning is that following: the examples which are close to each other are more likely to share a label. It can easily be applied to clustering all labeled and unlabeled examples together and assigning the most frequent label in each cluster to unlabeled examples from the same cluster.

## 2.2. Genetic Programming

Genetic programming (GP) is an evolutionary algorithm that was in the first place designed for creating a working computer program from a high-level statement of a problem, as its name implies. It was proposed by J.R. Koza in [20]. Like any other evolutionary algorithm it is a probabilistic search algorithm inspired by certain points in Darwinian theory of evolution for solving different NP-hard domain independent problems, not only creating the computer programs. [17]

What makes GP to be an evolutionary algorithm is following: 1) a population of individuals, 2) a fitness-biased selection method which gives higher probability to better solution to be chosen for breeding and 3) general inheritance method where genetic operators of crossover and mutation are applied to chosen individuals, usually respectively [21].

The flowchart of genetic programming is designated in Fig. 2-1. In Fig. 2-1  $gen$  is the current number of generations,  $N$  is the population size,  $p_c$  is the crossover probability and  $p_m$  is the mutation probability.

The flowchart is very similar to the flowchart of genetic algorithm (GA) which is also an evolutionary search algorithm used for solving NP-hard problems. The difference between GP and the original GA is in the representation of individuals. In GA solutions are always the same size and they are usually presented by a bit vector or a number array, although more complex structures are also possible if it is required by the nature of the problem. In GP the individuals can grow and grow if it is not controlled by any mechanism, while that could never happen in GA.

Types of GP differ from each other in having different problem representation. In other words, different data structures are used to define an individual. The most commonly used type of GP is tree based GP and apart from representing the computer programs it can be used for representing different kinds of arithmetical and logical expressions. Every tree-like structure representing an individual consists of two types of genes: functions and terminals. Terminals are the leaves of the tree (nodes without children) that can be actions, variables and constants, while functions are the nodes with children. The function's children provide the arguments for the function. The set of terminals and functions is called the set of primitives [22] [23].

Other problem representations are also possible. Constrained Syntax GP is based on trees with more than two children and a set of constraints is used to maintain the validity of tree

after applying the genetic operators. Graph based GP uses graphs for representing individuals and it is the most complex representation structure among the given GP types. It is used for an evolution of parallel programs. Linear GP uses a list of machine language instructions as problem representation. The last common GP type is grammar based and its individuals are created using the production rules [23].

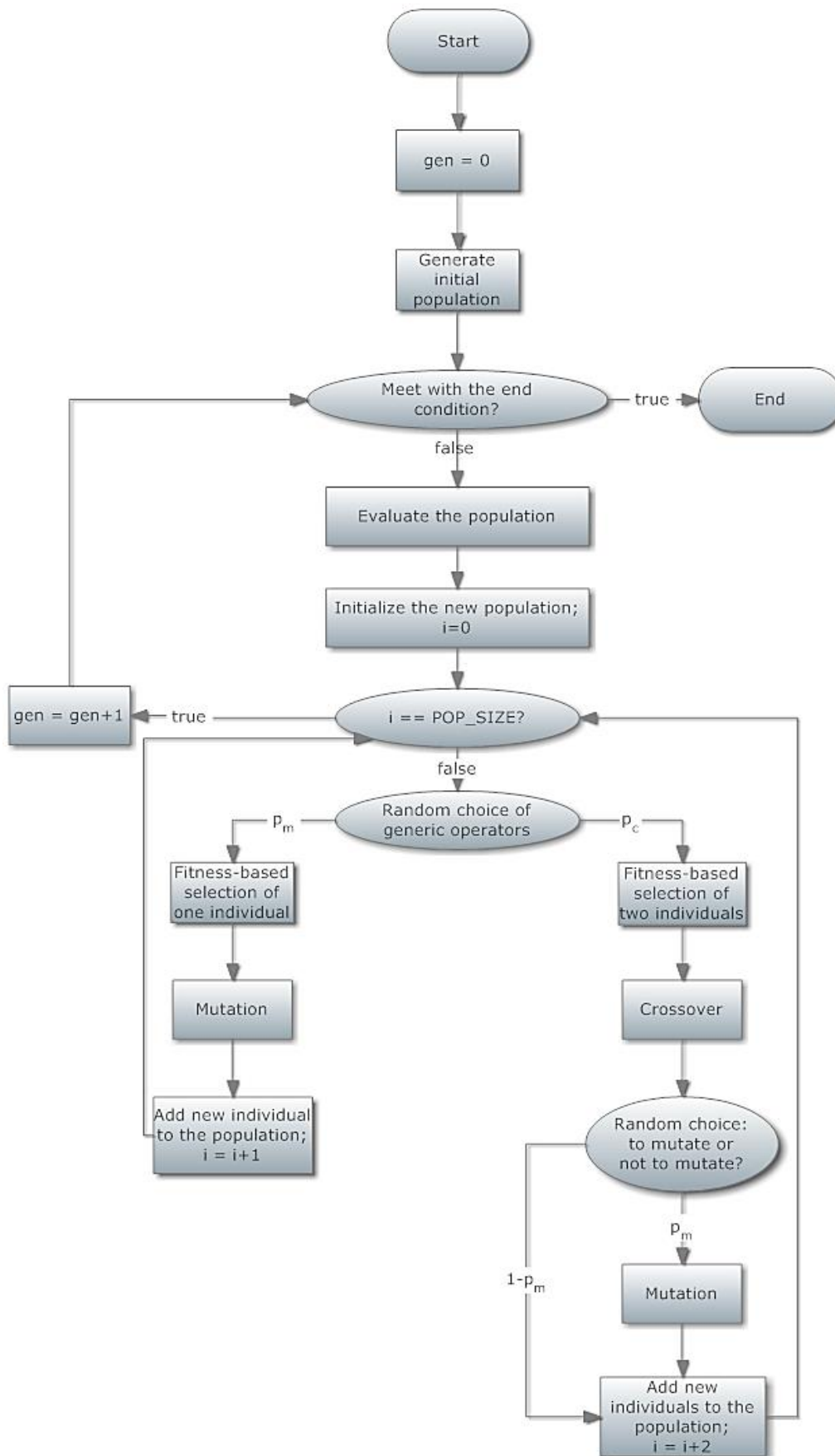


Figure 2-1 Flowchart of genetic programming

### 2.2.1. Genetic Programming in Classification

Genetic programming and other evolutionary techniques have been successfully applied to both supervised and unsupervised learning problems, but this thesis is mainly concentrated on the classification problems. GP is a flexible and powerful evolutionary technique that can be very suitable for training classifiers. It is easy to see the three components that make GP suitable for classification algorithm. As first, it easily adapts to different representation formalisms applicable to classification like decision trees, discriminant functions, classification rules etc. As second, GP itself is a search and optimization algorithm and as third, any preference criterion can be expressed as a fitness function that guides the search process.

Different interesting applications of GP in classification tasks are depicted in Fig. 2-2. The feature selection is performed implicitly as the result of the evolutionary process, since only a subset of features, represented by variables that are more likely to be chosen, appear in the final list.

Feature construction using GP can be performed in many different ways. Basically the principle remains the same in all of them – the new constructed features are codified by tree-like individuals with arithmetic operators and functions in the internal nodes while the original features and optionally constants are placed in the leaf nodes. The process of construction can be wrapped into classification algorithm or it can be executed separately before it.

Model extraction is the most common use of GP in classification. The population consists of individuals that represent one classifier. Fitness function usually measures the quality of classification and sometimes the size of an individual. The most common methods are extracting decision trees, classification rules and discriminant functions.

GP was also successfully used for ensemble learning. The idea behind ensemble classifiers is to use a group of base classifiers instead of only one classifier and combine them into a single classifier in a way that increases their accuracy. GP can play two roles in building the ensemble classifier. The first one is building the base classifiers and the second one is combining them. In both cases training data is separated into subsets and each base classifier is trained on only one subset. It can decrease the execution time for big training sets and there is no unused data. In the perfect scenario different base classifiers in an ensemble capture different patterns or aspects of pattern embedded in the data and through ensemble their decisions are incorporated into a final prediction. When GP is used for



building the base classifiers then they are usually combined using a simple voting mechanism. Some common base classifiers are naïve Bayes, C4.5, linear classifiers, decision stumps etc. They are translated into a mathematical expression and used as building blocks for GP.

Many interesting parallel implementations of GP ensemble are referenced in [21]. There are two common methods for parallelization of evolving ensembles of GP classifiers: island approach and team approach. They are often used for parallel evaluation of any population-based evolutionary algorithm (for example GA). Island approaches usually produce a team of strong individuals that cooperate poorly, while team approaches produce a team of weak individuals that cooperate strongly [21].

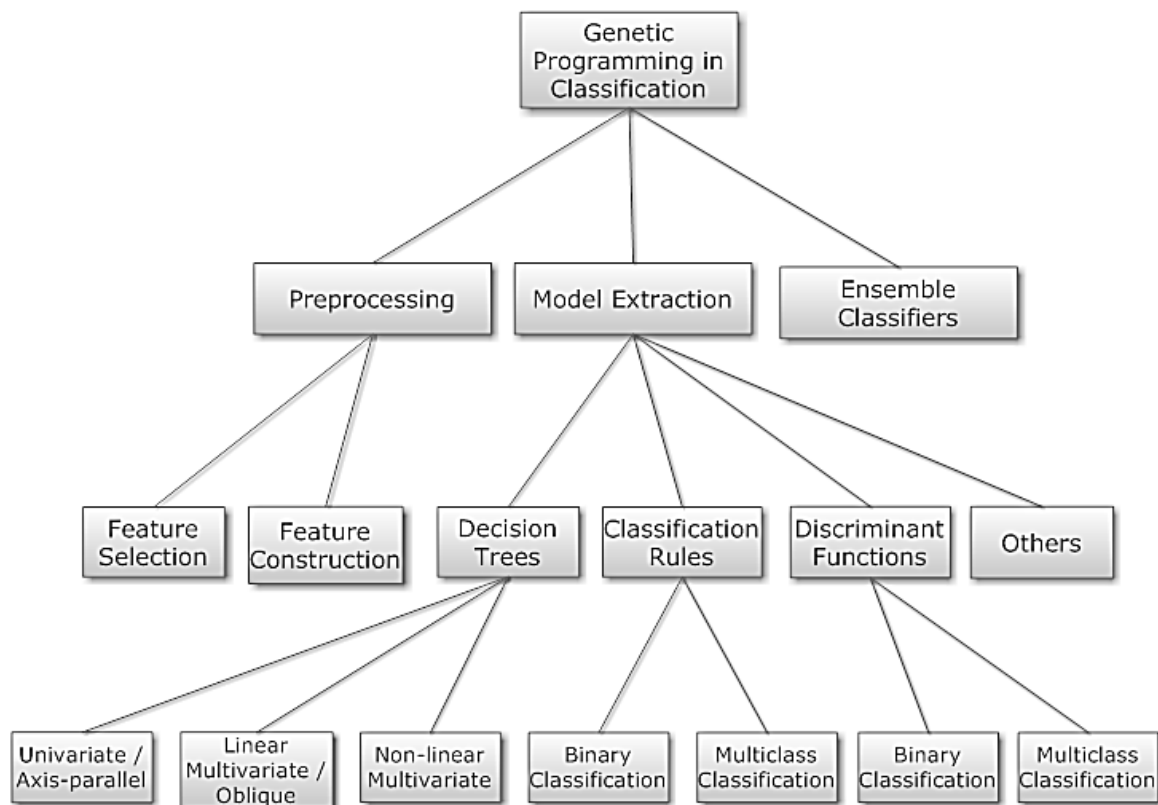


Figure 2-2 Applications of GP in classification tasks [21]

### 2.2.2. GP for Extracting Decision Trees

Decision tree is an obvious choice for representing the GP individual since its population often consists of tree-like structures. It was the first GP approach for classification proposed by J.R. Koza [19]. Every decision tree contains zero or more internal nodes and one or more leaf nodes. The internal nodes have two or more children nodes and they

contain splits, which test the value of an expression of the attributes [21]. Depending on the result of outcome value one of the arcs from internal node to its children is chosen. Each leaf node is labeled with a certain class label.

There are three main types of decision trees designated in Fig. 2-2 that differ from each other in how the feature space is partitioned.

Univariate or axis-parallel decision trees can test only a single variable at each internal node. They split the feature space by hyper planes parallel with axis. An example of axis-parallel decision tree and its division of two-dimensional feature space is shown in Fig. 2-3.

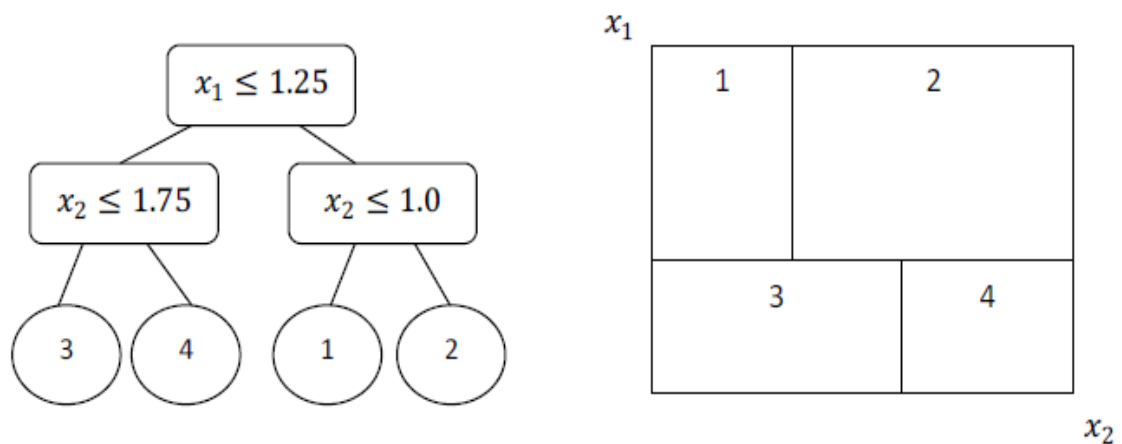


Figure 2-3 Axis-parallel decision tree and its division of 2D feature space [28]

Linear multivariate or oblique decision trees can test a linear combination of features at internal nodes. The tests are geometrically equivalent to hyper planes at an oblique orientation to the axis of the feature space. An example of oblique decision tree is designated in Fig. 2-4.

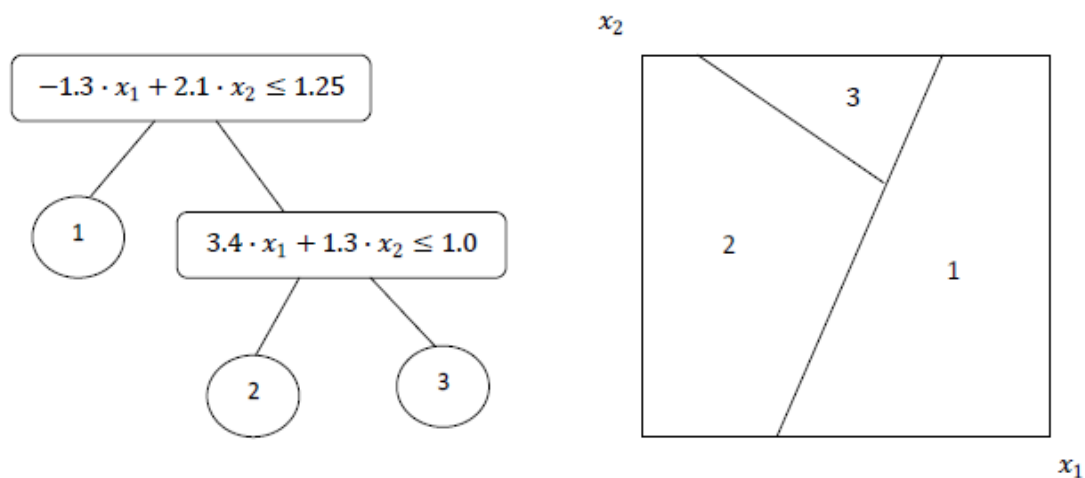


Figure 2-4 Oblique decision tree and its division of 2D feature space [28]

The third type of decision trees mentioned in [21] are nonlinear multivariate decision trees. This type of decision tree can contain nonlinear combinations of features inside internal nodes and therefore a nonlinear partitioning of the feature space can be done. It can be used for more difficult problems with complex decision boundaries.

### 2.2.3. GP for Learning Rule-Based Systems

Genetic programming can also be used for creating classification rules. Every rule consists of antecedent and consequent. The antecedent contains a combination of attributes and logical operators, while the consequent contains the value predicted for the class. If a data instance satisfies the condition in antecedent it is assigned to the class from consequent. An individual can represent a single rule or a set of rules. In the first case some consolidation method has to be used to create the final classifier after the evolutionary process.

An example of a GP individual representing a classification rule is designated in Fig. 2-5. The rule which is represented by the individual is following:

```
IF      ((X1 < 3)
OR      ((X1 >= 3) AND (X2 >= 25) AND (X3 < 31)))
THEN Class 1.
```

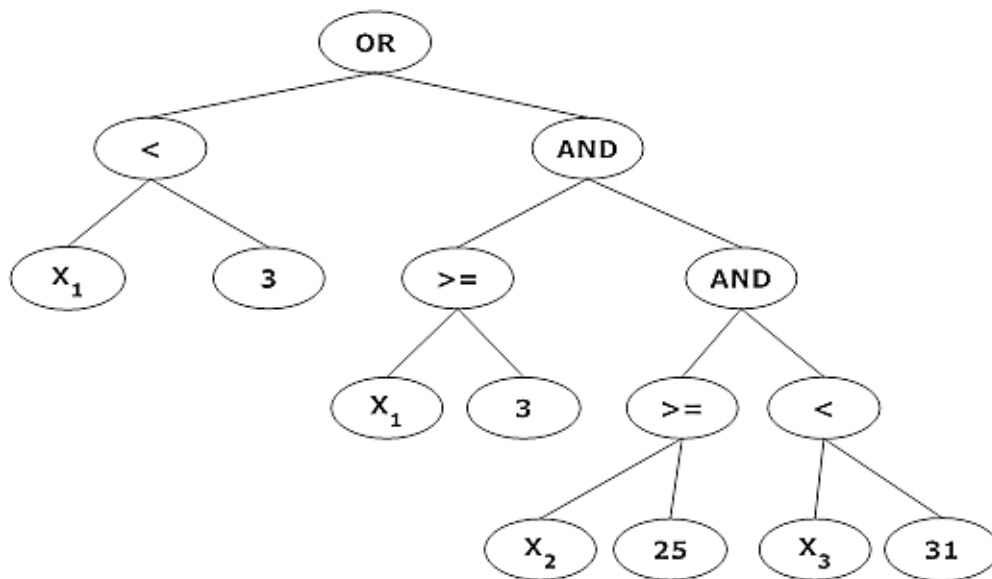


Figure 2-5 GP individual representing a classification rule

Some simple methods for creating rule-based classifiers are restricted to binary problems. It is always possible to create multiclass classifier for N classes with N-1 binary classifiers using one-against-all approach but the result of such classifier is often not well

interpretable. Methods for directly handling multiclass problems are generally more suitable because they can be applied more easily and it is easier to understand the rationale behind the classification that system outputs.

#### 2.2.4. GP for Learning Discriminant Functions

Discriminant functions are mathematical expressions also represented as tree-like structures with functions as internal nodes and variables and constants as leaf nodes. The variables represent the attributes of a data instance that has to be classified. The output value is computed from the operations performed on the values of the given attributes and the output determinates the output class.

Fig. 2-6 shows a discriminant function represented as a GP individual. The function in encoded form is:

$$X_1 * 0.26 + X_2 * 0.23 + X_3 * 1.07 + X_4 * 12.5.$$

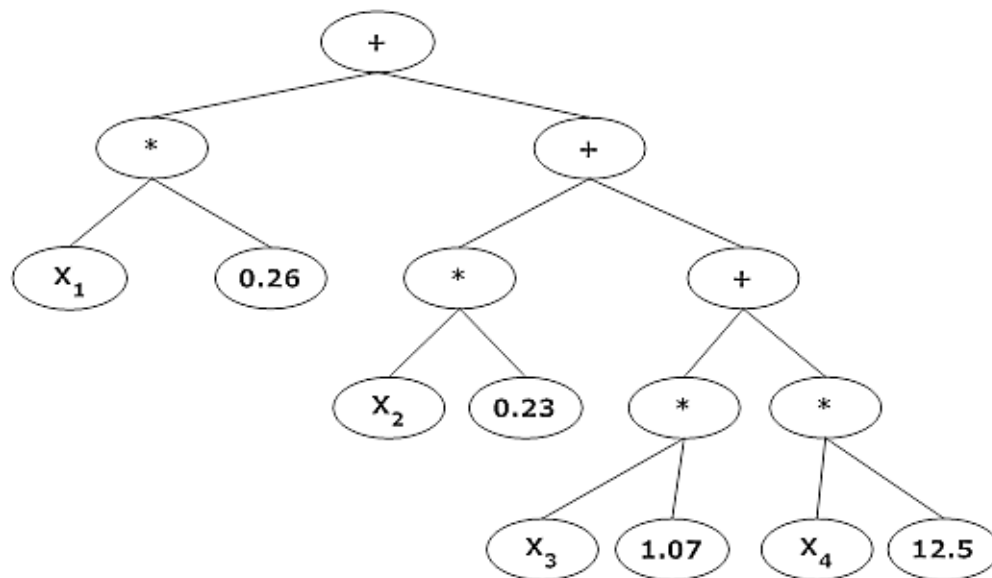


Figure 2-6 GP individual representing a discriminant function

Any mathematical function can be used in internal nodes, for instance: sine, cosine, logarithm etc.

If the problem is binary classification then a single discriminant function is enough for the distinction of classes – if the output is greater than the given threshold then the instance is assigned to positive class, otherwise it is assigned to negative class. The threshold is usually set to zero.

If the problem is multiclass classification then two approaches are possible. As first, N-1 discriminant functions that behave like binary classifiers can be used and then combined,

like the classification rules, using the common one-against-all approach for combining binary classifiers. As second, it is possible to interpret one discriminant function as multiclass classifier, but then N-1 threshold values have to be defined.

The obvious and the most common approach for integrating this representation into GP is to have a population of discriminant functions that use a subset of predefined functions in their internal nodes.

The fitness function usually reflects the accuracy of the discriminant function, but like in every other tree-like structure information like the total number of nodes, the maximal depth or the generalization ability can be taken into calculations as well. In some related work, for instance in [26] and [27], the fitness function is composed of classification accuracy and generalization ability that was calculated on a small part of training set that was not used for training.

## 2.3. Clustering

Clustering is a classic unsupervised learning problem. The goal of clustering is to partition the unlabeled data set into previously determined number of clusters,  $K$ . Intuitively cluster is a comprising group of data points whose inter-point distances are small compared with the distances to points outside of the cluster [25].

### 2.3.1. K-means Algorithm

K-means algorithm is the simplest and the most common clustering algorithm. Suppose we have a data set  $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  and a fix number of clusters  $K$ . Every cluster is represented by its centroid  $\boldsymbol{\mu}_k$ ,  $k \in \{1, \dots, K\}$ . The quality of partitioning the data examples into clusters represented by  $\boldsymbol{\mu}_k$  is determined by the cost function:

$$J = \sum_{i=1}^N \sum_{k=1}^K b_k^{(i)} \left\| \mathbf{x}^{(i)} - \boldsymbol{\mu}_k \right\|^2.$$

The value  $b_k^{(i)}$  is a binary indicator variable describing which of the  $K$  clusters data point  $\mathbf{x}^{(i)}$  is assigned to. If data point  $\mathbf{x}^{(i)}$  is assigned to cluster  $k$  then  $b_k^{(i)} = 1$ , and  $b_j^{(i)} = 0$  for  $j \neq k$ . The cost function represents the sum of the squares of the distances of each data point to its assigned vector  $\boldsymbol{\mu}_k$ . The final goal of the algorithm is to find values for  $\{b_k^{(i)}\}$  and  $\{\boldsymbol{\mu}_k\}$  so as to minimize  $J$  [25]. The function would be zero if all examples would perfectly overlap with their centroids and that is a trivial and not probable case. In order to minimize the cost function  $J$  every data point has to be assigned to the closest centroid. However, the



question how to find such centroids so that the function reaches its minimum remains. The answer is in an iterative algorithm because the solution cannot be found in a closed-form. The iterative algorithm starts with the random initialization of the centroids. After that in every iteration and for every training example  $\mathbf{x}^{(i)}$  the value  $b_k^{(i)}$  is calculated for every  $k \in \{1, \dots, K\}$  by assigning each training example to the closest centroid  $\mu_k$ . The cost function can be now be minimized with fix values of  $b_k^{(i)}$  and by calculating gradient  $\nabla_{\mu_k} J = 0$  and solving it by  $\mu_k$ . The final expression for  $\mu_k$  is:

$$\mu_k = \frac{\sum_i b_k^{(i)} \mathbf{x}^{(i)}}{\sum_i b_k^{(i)}}.$$

The new value of centroid  $\mu_k$  corresponds to the mean of all the data points  $\mathbf{x}^{(i)}$  assigned to cluster  $k$ . The two phases of re-assigning training examples to clusters and re-computing the cluster means are repeated in turn until there is no further change in assignments (or until a certain number of iterations is exceeded).

Finding the global minimum of function  $J$  depends on the choice of initial centroids. There are several ways of initializing the centroids among which choosing random points in the feature space is the worst one. Choosing the positions of  $K$  random examples for the positions of centroids is a very common approach but it is still not resistant to local optimums. An interesting approach is to calculate the principal component using a method PCA and divide the feature space to  $K$  equal intervals which also divide examples into  $K$  groups. Centroids are then initialized as the means of examples in each group.

The best approach for choosing centroids is given by the algorithm k-means++. The idea behind the algorithm is very simple, while the significant reduction of the cost function is proven. The first centroid is randomly chosen among one of the training examples. Every other centroid after it should be chosen as far as possible from the ones that have already been chosen. The probability of choosing the position of training example  $\mathbf{x}^{(i)}$  for the next centroid is proportional to the square distance from  $\mathbf{x}^{(i)}$  to the closest already existing centroid  $\mu_k$  as shown on following formula [18]:

$$P(\mu_i = \mathbf{x}^{(i)} | D) = \frac{\|\mathbf{x}^{(i)} - \mu_k\|^2}{\sum_j \|\mathbf{x}^{(j)} - \mu_k\|^2}.$$

## 2.4. Decision Trees

Decision trees are very popular because they can easily be interpreted by humans and the extracted knowledge can be clearly presented and visualized. They can also be interpreted as a set of decision rules.

In the simplest variant of induction of decision trees internal nodes are variables assigned to one attribute and every attribute is interpreted as nominal. For each possible value of the attribute one arc is leading from internal node to either another internal node either to a leaf node. The classification process starts from the root node and for every attribute its value is examined and the path on the tree is determined. All the leaf nodes are assigned to class labels. A simple example of such decision tree from the paper where they were introduced is designated in Fig. 2-7.

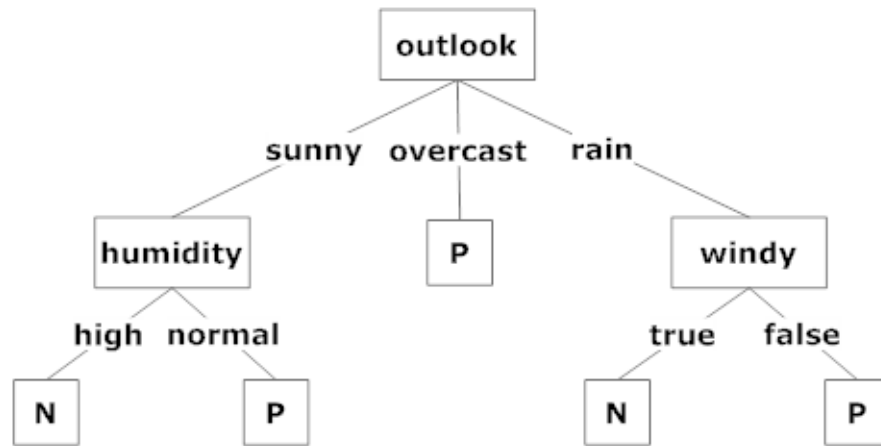


Figure 2-7 A simple decision tree [24]

The most common algorithms for decision trees learning are ID3 and C4.5. They are both greedy algorithms proposed by Quinlan [24]. The basic idea behind ID3 algorithm is to associate each internal node with an attribute which is the most informative among the attributes not yet considered in the path from the root. Information gain is used for measuring how “informative” the attribute is. Let  $D$  denote a set of training examples and let  $A$  denote an attribute in  $D$  with  $n$  possible values  $v_i$ ,  $i \in \{1, \dots, n\}$ . The information gain of attribute  $A$  on dataset  $D$  is the difference between entropy on dataset  $D$ ,  $H(D)$ , and a sum of entropy values on a subset of dataset  $D$ ,  $Dv_i$ , that remains after dividing  $D$  using possible values  $v_i$  of attribute  $A$ , as follows in [18]:

$$I(D, A) = H(D) - \sum_{i=1}^n \frac{|Dv_i|}{|D|} H(Dv_i).$$

The biggest possible information gain is equal to entropy and it is reached when for each attribute value  $v$  a unique classification can be made for the class attribute.

C4.5 is an upgrade of ID3 with several extensions. C4.5 can deal with training sets that have records with unknown attribute values by evaluating the gain for an attribute by considering only the records where that attribute is defined [35]. With C4.5 handling both discrete and continuous values is possible. Further on, the algorithm goes back through the tree once it has been created and attempts to remove branches that do not help by replacing them with leaf nodes. This procedure is called tree pruning.

## 2.5. K-nearest neighbors

K-nearest neighbors is a simple and lazy instance-based learning algorithm that can be used for both classification and regression. The idea is to classify a data instance by looking at its closest neighbors. The distance between  $n$ -dimensional data instances  $x^{(i)} = (x_1, \dots, x_n)$  is usually calculated using Euclidean metrics:

$$d(x^a, x^b) = \sqrt{\sum_{i=1}^n (x_i^a - x_i^b)^2}$$

In  $k$ -NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its  $k$  nearest neighbors. Typically,  $k$  is a small integer value. If  $k = 1$ , then the object is simply assigned to the class of that one single nearest neighbor.

The training algorithm consists only of adding all the training instances to a list. The classification of unseen instance is more complex. For a given data instance  $x_q$  with unknown classification its  $k$  closest neighbors have to be found:  $x_1, x_2, \dots, x_k$ . This results in much bigger complexity of test than training procedure. The output of the algorithm is the most frequent class label among the nearest neighbors which is formally given as the value of the function  $h(x_q)$ . The help function  $\delta(a, b)$  is equal to one if  $a=b$ , 0 otherwise. The equation for output function is:

$$h(x_q) = \underset{v \in \{0,1,\dots,K\}}{\operatorname{argmax}} \sum_{i=1}^k \delta(v, y^{(i)})$$

The advantages of the algorithm are its resistance to noise in the data and its simplicity. Determining the right parameter  $k$  is very important and problem specific. If  $k$  is too small,

the algorithm is very sensitive to noise in the data. If  $k$  is too big a borderline decision becomes too smooth and oversimplified. The ideal  $k$  is highly related with the number of examples. With a big number of examples even a smaller  $k$  can result in good generalization ability.

## 2.6. Ensemble Learning

Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn only one hypothesis from the training data, ensemble methods try to construct a set of hypotheses and then combine them [30]. Ideally, different base classifiers in an ensemble capture different patterns or aspects of a pattern embedded in the whole range of data and then, through ensemble learning, these different patterns or aspects are incorporated into a final prediction [21].

Two main issues have to be addressed for applying ensemble approach. As first, the basic classifier has to be chosen. Basic classifiers can belong to the same type, or to different types of classifiers, but diversity among them is very important for ensembles. Generally, to get a good ensemble, the base learners should be as accurate as possible, and as diverse as possible [30]. As second, combining base classifiers has to be solved. Two common ensemble methods are bagging and boosting.

### 2.6.1. Bagging

Bagging is a method for combining basic classifiers using a simple voting mechanism. The idea is following: we have  $N$  basic classifiers that produce  $N$  hypothesis  $h_i$ . Each hypothesis is independent and makes error with probability  $p$ . Probability that the majority voting of  $N$  hypotheses will make an error, if  $k$  basic classifiers make an error, reduces significantly.

Each basic learner is trained on its bootstrap sample. A bootstrap sample is obtained by subsampling the training dataset with replacement, where the size of a sample is as the same as that of the training data set. Thus, for a bootstrap sample, some training examples may appear, but some may not [30].

The pseudo code of bagging algorithm is following:

**Input:**

*Data set  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ;  
 Base learning algorithm  $L$ ;  
 Number of learning rounds  $T$ ;*

**Process:**

*for  $t = 1, \dots, T$ :  
     Generate a bootstrap sample  $D_t = \text{Bootstrap}(D)$   
     Train a base learner  $h_t$  from the sample  $D_t$ ,  $h_t = L(D_t)$   
 end.*

**Output:**

*$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^T \mathbf{1}(y = h_t(x))$  where  $\mathbf{1}(a)$  is 1 if  $a$  is true and 0 otherwise.*

## 2.6.2. Boosting

Boosting is a whole family of algorithms among which the most popular one is AdaBoost (Adaptive Boosting). Boosting is in general referred to as the process of turning a weak learner into a strong learner.

In practice the hypotheses from basic learners are rarely independent and some hypotheses have fewer errors than others, so all votes are not equal. The idea is to take the weighted majority.

Most boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final classifier. When they are added, they are typically weighted in specific way usually related to the accuracy of the weak learners. After a weak learner is added, the data is reweighted so that misclassified examples gain weight and correctly classified examples lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified [31].

### 2.6.2.1 AdaBoost

The AdaBoost algorithm was the first practical boosting algorithm and still remains one of the most widely used and studied, with applications in numerous fields. The pseudo code of AdaBoost from [32] is given as follows:

**Input:**

- *Data set  $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$  with labels  $y_i \in Y = \{1, \dots, C\}$  drawn from a distribution  $D$*
- *Base learning algorithm  $\text{WeakLearn}$*
- *Number of learning rounds  $T$*

**Initialization:**

- *$D_1(i) = 1/m$  for  $i=1, \dots, m$*



**Process:**

for  $t = 1, \dots, T$ :

- Call *WeakLearn*, providing it with the distribution  $D_t$
- Get back a hypothesis  $h_t : X \rightarrow Y$
- Calculate the error of  $h_t$ :  $\varepsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$   
If  $\varepsilon_t < \frac{1}{2}$ , then set  $T := t-1$  and abort loop.
- Set  $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
- Update distribution  $D_t$ :

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t, & \text{if } h_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

where  $Z_t = \sum_i D_t(i)$  is a normalization constant chosen so that  $D_{t+1}$  becomes a distribution function

end.

**Output:**

The final hypothesis:

$$h_{final}(x) = \arg \max_{y \in Y} \sum_{t: h_t(x)=y} \log \frac{1}{\beta_t}$$

AdaBoost is a serious candidate for applying to the problem of real-time inferring presence status on mobile devices because of its speed and accuracy.

## 2.7. Naïve Bayes

Naïve Bayes is a simple probabilistic classifier based on Bayes' theorem and a naïve assumption that features are independent random variables. The basic idea is to find out the probability of the previously unseen instance belonging to each class and then to pick the most probable one.

Bayes' theorem, well known in probability theory and statistics, enables mathematical manipulation of conditional probabilities:

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}.$$

We try to maximize  $p(c_j | d)$  which is the probability of instance  $d$  being in class  $c_j$ . Other parts of formula are the probability of generating instance  $d$  given class  $c_j$ ,  $p(d | c_j)$ , the probability of occurrence of class  $c_j$ ,  $p(c_j)$  and the probability of instance  $d$  occurring,  $p(d)$ . The last component can be ignored since it is the same for all classes.

Naïve Bayesian classifier assumes that features have independent distributions and it simplifies calculating  $p(d | c_j)$  which becomes equal to the multiplication of the probabilities of class  $c_j$  generating the observed value for every feature separately.

Naïve Bayes faces some difficulties when the initial naïve assumption is very incorrect. It is often applied to text classification problems but is has also been successfully applied to inferring interruptions on mobile phones in [33] and therefore it is an interesting method for the problem in this thesis.

## **2.8. Feature selection**

In machine learning, feature selection is the process of selecting a subset of relevant features for use in model construction.

It is an obligational part of the data analysis process especially when the number of features is very big which is common for problems like text classification or gene selection. In the mentioned problems hundreds to tens of thousands features have to be handled in an efficient and accurate way.

Feature selection techniques are used even if the number of features is relatively small in order to improve the model interpretability, to reduce the training time and to enhance the generalization ability by reducing over fitting. They differ from techniques used for the large number of features. The most common criterion for eliminating features is accuracy, although it can be combined with for instance energy efficiency, especially in mobile computing.

The simplest feature selection algorithm is to test all possible subsets of features finding the one for which the best value of criterion is achieved. The best subset contains the least number of features that most contribute to accuracy. This kind of exhaustive search is in generally impractical, except for a small number of features.

The most common feature selection methods are divided into complete, heuristic and random methods.

The main feature selection heuristic methods are forward and backward selection. Forward selection starts with no features and then adds them one by one choosing always the one that decreases the error the most. The process stops when any further addition stops decreasing the error significantly. Backward selection starts with all the features and then removing them one by one if removing the chosen feature decreases the error. The process stops when any further removal increases the error significantly. Forward and backward selections exist in many different variants.

The most common random techniques for feature selection are genetic algorithm and simulated annealing. Minimum Redundancy Maximum Relevance (mRMR) which was

proposed in [36] uses mutual information, correlation or distance measures to select the features which remain relevant in the presence of other selected features.

The summary of feature selection techniques from [34] is designated in Figure 2-8.

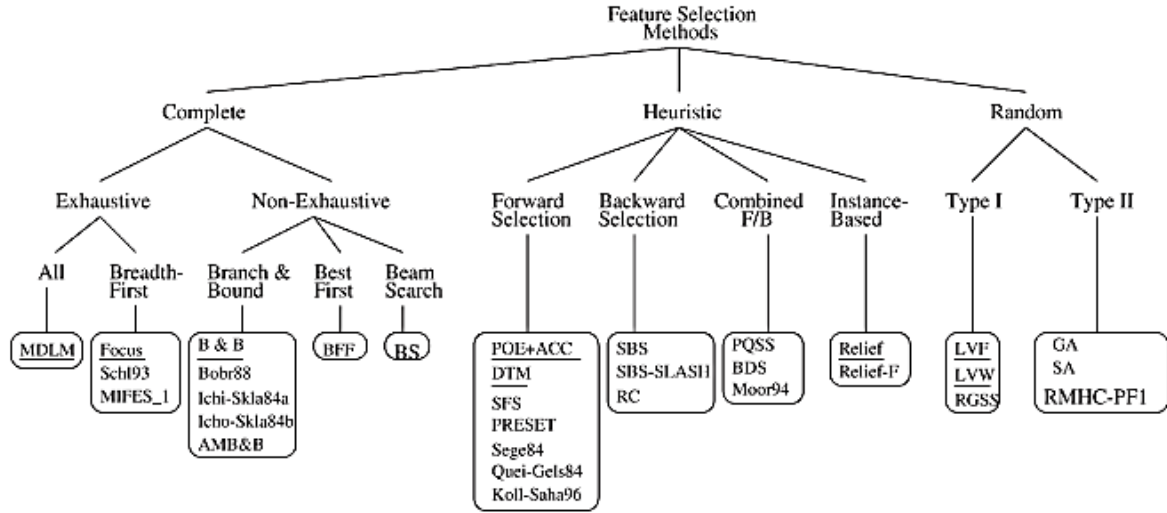


Figure 2-8 Summary of feature selection methods

## 2.9. Incremental learning

Incremental learning, also sometimes called adaptive, online or transfer learning, is a machine learning paradigm used when all training examples are not available at the beginning of the learning process, but they appear in small batches over time. A typical approach for learning new information involves discarding the existing classifier, and retraining the classifier using all of the data that has been accumulated thus far [38]. The incremental learning assumes adjusting what has already been learned according to the new examples without completely relearning the model on the whole training set, but without forgetting the previous knowledge. It is very useful in many different problems including inferring presence status because a logical approach for implementing such application on mobile phones would require adapting to user's habits over time without keeping all the previously collected data in memory of mobile device and without repeating the whole learning process.

Incremental learning can be both supervised and unsupervised. Formally defined, an incremental learning algorithm has to meet the following criteria [39]:

- it is able to learn and update with every new labeled or unlabeled data,
- it preserves previously acquired knowledge,

- it should not require access to the original data,
- it generates new class or cluster when required, even divides or merges clusters when needed and
- it is dynamic in nature with the changing environment.

Some of the incremental learning algorithms are Learn++ [38], incremental induction of decision trees [41] etc.

### 2.9.1. Learn++

The most popular supervised incremental learning algorithm is Learn++ which adapts AdaBoost algorithm. Learn++ has been improved several times to meet all incremental learning criteria. Like AdaBoost, Learn++ is based on the ensemble of weak classifiers, for instance decision stumps, and weighted voting mechanism. Each new classifier added to the ensemble is trained using a set of examples drawn according to a distribution, which ensures that examples that are misclassified by the current ensemble have a high probability of being sampled. In an incremental learning setting, the examples that have a high probability of error are precisely those that are unknown, or that have not yet been used to train the classifier [39].

Learn++.NC was later developed for learning New Classes (NC) with new data from existing classes assumed to remain stationary. A dynamically weighted consult-and-vote mechanism was applied. Looking at the decisions of each classifier, each classifier decides whether its decision is in line with the prediction of other classifiers and the classes on which it was trained and if the result is negative it reduces its voting weight. [39]

None of the mentioned algorithms can be employed to incremental learning in non-stationary environment. In other words, they assume that existing class boundaries remain constant over time, which does not have to be true. For enabling to correctly learn classes even if the underlying generation function  $f$  changes the new algorithm Learn++.NSE (Nonstationary Environment) was developed.

The pseudo code of Learn++ algorithm is following:

**Input:** For each database drawn from  $D_k = 1, \dots, K$

- Sequence of  $m$  training examples  $S_k = \{(x_{1k}, y_{1k}), (x_{2k}, y_{2k}), \dots, (x_{mk}, y_{mk})\}$
- Base learning algorithm *WeakLearn*
- Number of learning rounds  $T_k$

for  $k=1,2,\dots, K$ :

**Initialization:**

- $w_1^i = D(i) = \frac{1}{m_k}, \forall i$ , unless there is prior knowledge to select otherwise

**Process:**

for  $t = 1, \dots, T_k$ :

- Set  $\mathbf{D}_t = \mathbf{w}_t / \sum_{i=1}^m w_t(i)$  so that  $\mathbf{D}_t$  is a distribution
- Randomly choose training data subset  $TR_t$  according to  $\mathbf{D}_t$
- Call *WeakLearn*, providing it with  $TR_t$
- Get back a hypothesis  $h_t : X \rightarrow Y$
- Calculate the error of  $h_t$ :  $\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$  on  $S_k$   
If  $\varepsilon_t < \frac{1}{2}$ , then set  $t:=t-1$ , discard  $h_t$  and go to randomly choosing another training set
- Set  $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
- Call *weighted majority*, obtain the overall hypothesis  
 $H_t(x) = \arg \max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$  and compute the overall error  
 $E_t = \sum_{i:H_t(x_i) \neq y_i} D_t(i) = \sum_{i=1}^m D_t(i) [|H_t(x_i) \neq y_i|]$   
If  $E_t > 1/2$ , set  $t = t - 1$ , discard  $H_t$  and go to randomly choosing another training set
- Set  $B_t = E_t(1-E_t)$  and update the weights of the instances:  
$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & \text{if } H_t(i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

end.

end.

**Output:**

Call *weighted majority* on combined hypothesis  $H_t$  and output:

$$H_{final}(x) = \arg \max \sum_{k=1}^K \sum_{\substack{t:h_t(x)=y \\ y \in Y}} \log \frac{1}{\beta_t}$$

Learn++ could be applied to inferring presence status on mobile devices so that the algorithm learns the individual behavior of a certain user and periodically adjusts the model with small batches of collected data.

### 3. Inferring Presence Status

In this chapter the used dataset will be introduced and programming-environment independent implementation description of used algorithms and their parameters will be given. Every algorithm was validated using stratified 5 fold cross validation.

#### 3.1. Adaptation of the Mobile Data Challenge Dataset

Mobile Data Challenge (MDC) data set was collected during Lausanne Data Collection Campaign from October 2009 until March 2011 from around 200 individuals over more than a year [11]. The logs contain information related to GPS, WiFi, Bluetooth and accelerometer traces, but also call and SMS logs, multimedia and application usage.

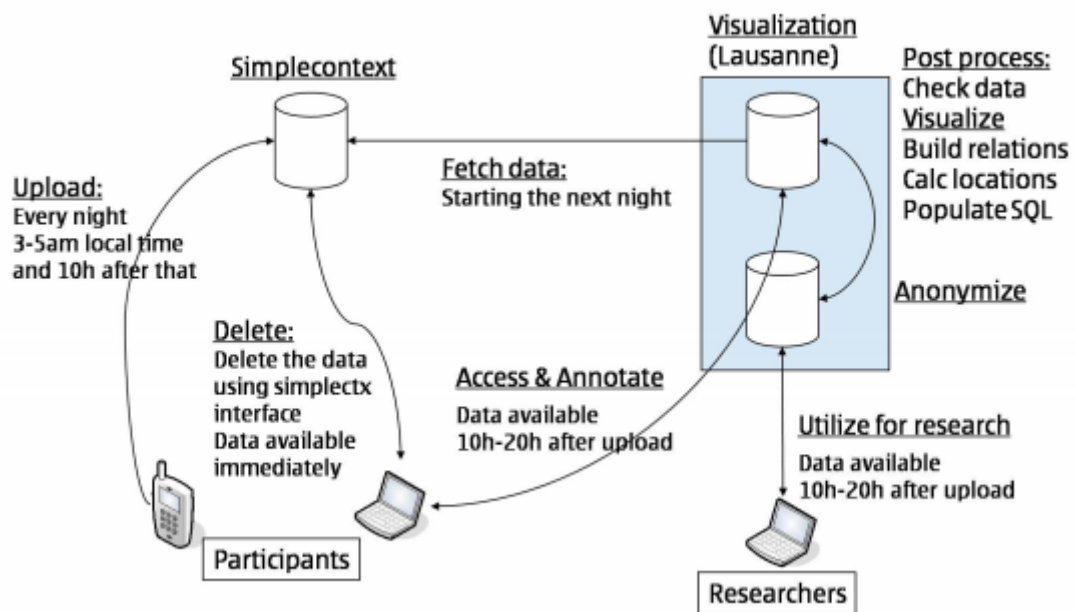


Figure 3-1 LDCC data flow, progressing from mobile data from volunteers to anonymized data for research [11]

The subset of the data used for the purpose of this work contains 17,861,168 actions from 38 users collected over 8154 days. The actions are following: call log, calendar, system log, accelerometer log, GPS sensor log and GSM cell identifier log. The figure 3-1 shows how the data was collected during LDCC.

Not all the actions were separately used for status inferring because one of the goals is to reduce the number of status changes, but retain the acceptable accuracy. As suggested in [1], the actions were aggregated every 5 minutes into one record used for learning and evaluation of the chosen models.

Data was aggregated on two different ways depending on preferences and needs of the used classifiers. For instance, it is known that discriminant function based GP performs better on numerical than nominal features, unlike the decision trees for which it is exactly the opposite.

In the numerical dataset call log and calendar were used as numerical features. The features were number of missed calls, number of incoming calls, total call duration in minutes, number of confirmed events etc.

In the nominal dataset all of the features were turned to nominal using the value with maximal frequency in the given time window. For instance, instead of using one feature for every call type, it becomes only one feature. If in the observed 5 minutes window there are more outgoing calls then missed or incoming calls then in the generated example the single feature “call type” is set to value “missed”.

Some other adaptations have been made too. Since the phones have also changed since the year 2009 and most of the smartphones do not have “beep”, “ring-once” nor “ascending”, those ring types are reduced to two remaining ring types - “beep” to “silent” and others to “normal”.

All the features in nominal and arithmetic dataset are enumerated in Table 3-1.

<i>Nominal dataset</i>		<i>Arithmetical dataset</i>		<i>Energy consumption</i>
<i>Name</i>	<i>Values</i>	<i>Name</i>	<i>Values</i>	
Calendar status	tentative, confirmed, none	Calendar status confirmed	numeric	Low
		Calendar status tentative	numeric	
Calendar class	public, private, none	Calendar class public	numeric	Low
		Calendar class private	numeric	
Call voice	voice, none	Number of calls	numeric	Low
		Call duration	numeric	
Call SMS	SMS, none	Number of SMSes	numeric	Low
Call direction	incoming, outgoing, missed, empty	Number of incoming calls	numeric	Low
		Number of outgoing calls	numeric	
		Number of missed calls	numeric	
Ringtone	normal, silent, empty	Ringtone	normal, ascending, once, beep, silent, empty (as numeric)	Low
GSM cell	nominal	GSM cell	numerical	Low
Previous status	Available, Unavailable, Busy, Texting			Low
Time period	early morning, mid-morning, late morning, around noon, afternoon, late afternoon, evening, night			Low
Accelerometer	slow, moving, empty			High

Table 3-1 Features

## 3.2. Implementation of GP algorithms

In this chapter for the chosen GP algorithms the used parameters will be given together with designated performance of the best individuals on training and test set where it is applicable. For Bagging GP and GP 1-against-all it is not possible to present the training



and test error through generations since discriminant functions were trained sequentially, one after another.

### 3.2.1. Implementation of GP for Learning Discriminant Functions

For the purpose of this classifier the arithmetical dataset from chapter 3.1 was used. Nominal features like ringtone and time period were turned to numerical, while numerical features stayed numerical.

Discriminant functions were trained in two ways:

1. For each run only one discriminant function was trained - multiple threshold values were determining the final output of the classifiers.
2. For each run N discriminant functions were trained as binary classifiers, one for each class. Functions were combined using 1-against-all method where every binary classifier is trained to tell apart data instances of the belonging class from all other data instances.

For inner nodes two subsets of functions were used:

1. +, -, /, \*, If, >, <, Pow, &, |, Max, Min, Exp, Log
2. +, -, /, \*, Sq, Sqrt, If, If3, >, <, !, Pow, &, |, Xor, Max, Min, Exp, Log, Sin, Cos

It is visible that the sets of functions contain both mathematical and logical functions. Thus, this specific implementation is a hybrid between learning rule-based systems and learning discriminant functions. Leaf nodes contain variables (attributes) and real constants. For population initialization ramped half-half method was used.

Two different fitness functions were applied:

- simple classification accuracy on a training set – the number of correctly classified instances divided by a total number of instances,
- fitness function combined of training accuracy, validation accuracy and a parameter validation proportion (0.5) as follows:

$$\text{valProp} * \text{validationFitness} + ((1 - \text{valProp}) * \text{trainingFitness}).$$

The validation fitness is not calculated on real validation set from the cross validation. One subset of training data is left aside for this purpose.

Genetic operators that were used are: crossover (switching random subtrees from two individuals), mutation of the node (the value of a single node is changed in one individual), mutation of the subtree (replacing the subtree by a random one) and creating a brand new

random program (the new program replaces the whole individual). 4-tournament selection was used. The ramped half-half method was used for initializing the population. The elitism is used in every GP variant so the best individual cannot be lost.

The algorithm finishes when it reaches 200 generations or when an individual with fitness bigger than a certain number is found. At first the fitness threshold was set to 0.9 and then to 0.99. The population size is 100 individuals. The maximal depth of the tree is 7 and the population size is 100. The relation between training and test error through first 100 generations is designated in Fig. 3-2. Test error is constantly slightly bigger than training error, but the difference between them is very small. An example of the final solution is designated in Fig. 3-3.

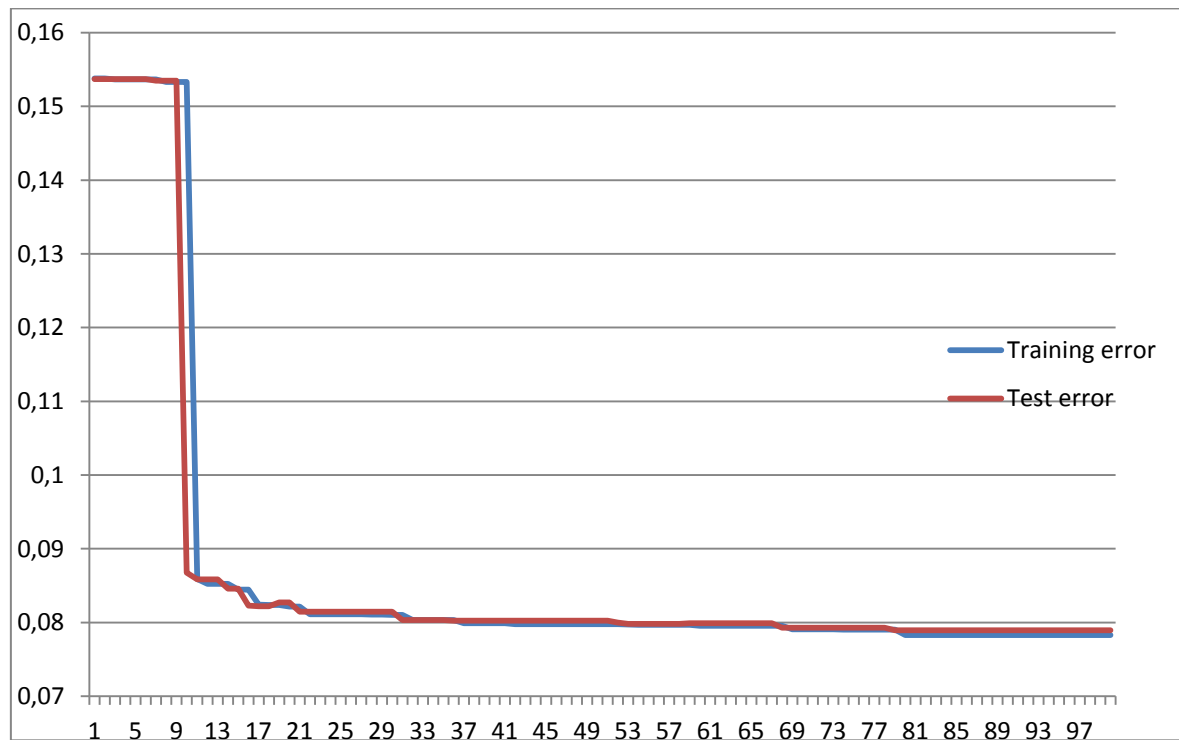


Figure 3-2 Training and test error on GP Discriminant Function Multiclass

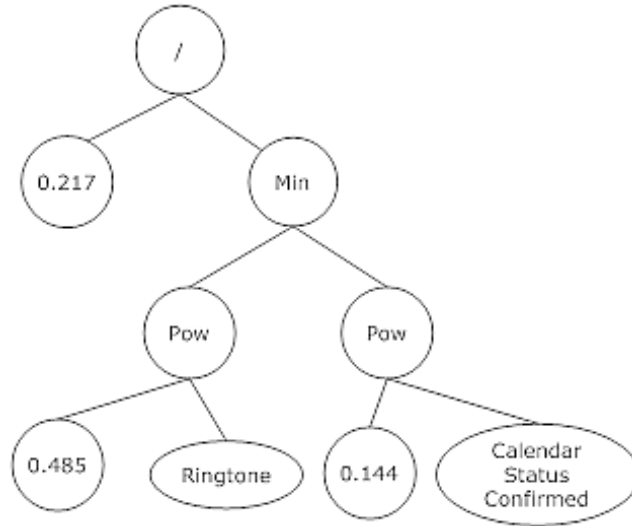


Figure 3-3 An example of discriminant function

### 3.2.2. Implementation of GP for Learning Decision Trees

For applying GP to learning decision trees the nominal data set mentioned in 3.1 was used. All the features are turned to nominal and used as nominal. GSM cell identifier is also used as nominal feature after determining all the possible values during the preprocessing.

For leaf nodes, only the nodes with class labels were used, while the nodes assigned to other attributes are placed as internal. Classification error is used as fitness function. Tournament selection is applied as selection operator. The algorithm stops after 50 generations because after 50 generations in most of the cases there is no significant progress in training or test fitness. An example of training and generalization error on a random user during the first 100 generations is designated in Figure 3-4. It is visible that even after 40 generations the fitness remains the same because the local optimum was reached. It is obvious that the algorithm is stuck in the local optimum because the generalization error also remains the same for more than 50 generations. This is not an isolated case of such behavior.

One decision tree created by GP is designated in Figure 3-5.

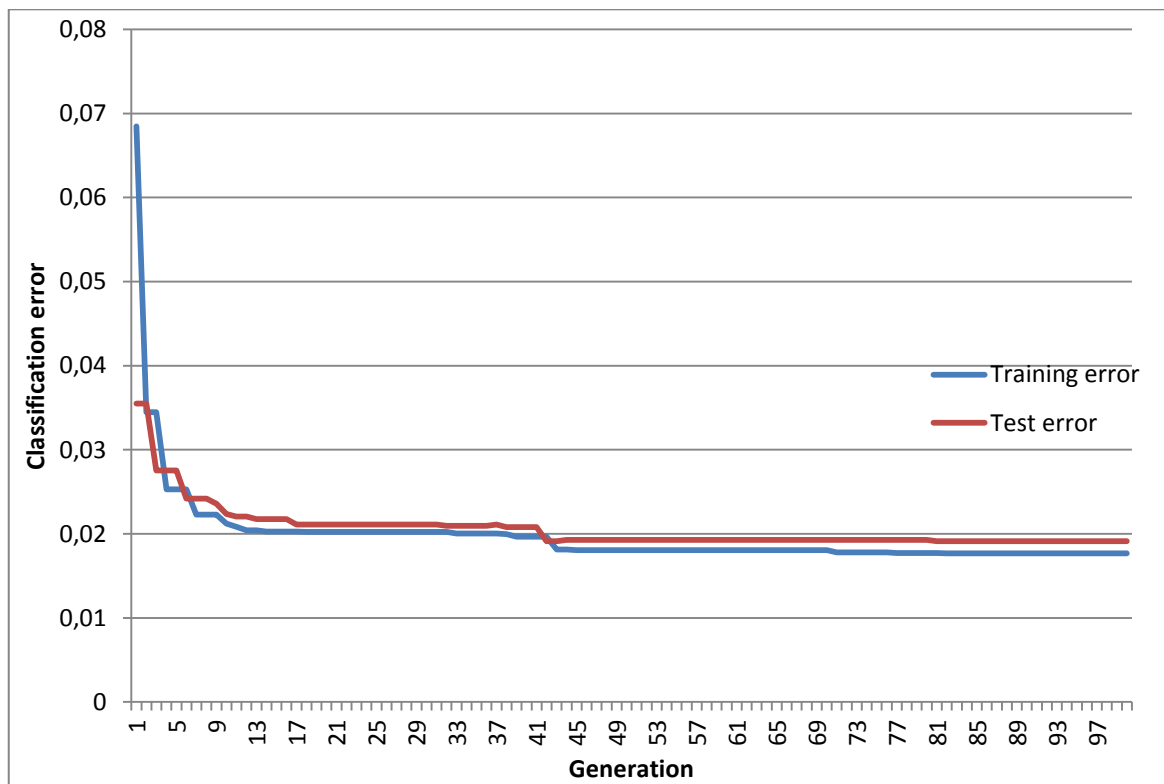


Figure 3-4 GP decision trees - training and test error on one random user

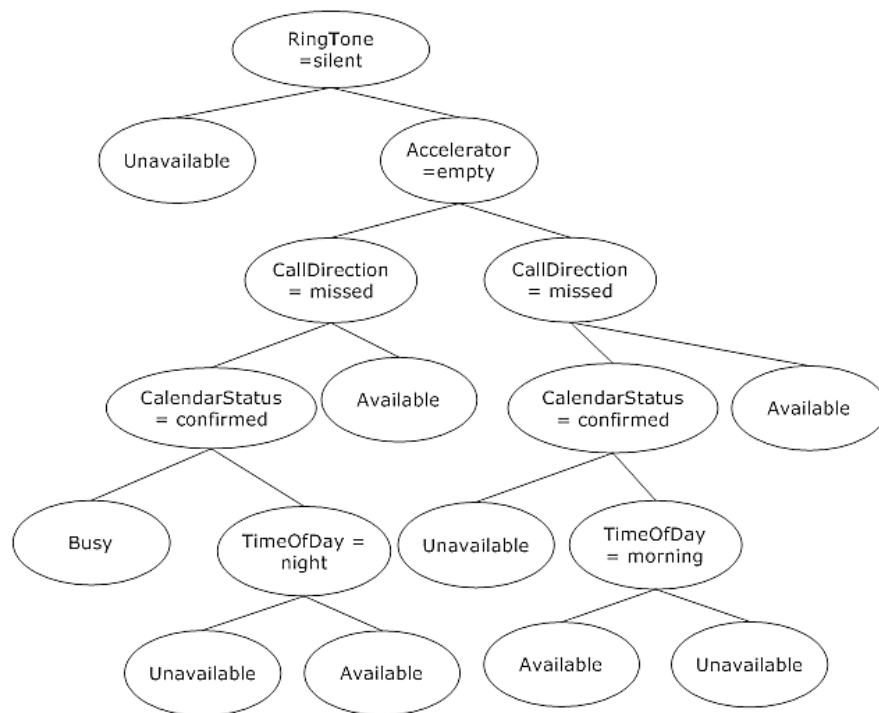


Figure 3-5 The best GP-made decision tree for a random dataset

### 3.2.3. Implementation of Bagging Using GP

Bagging was implemented using the discriminative functions trained by GP as the weak classifiers. The number of generations used for training them was reduced to 50 generations (4 times less than the number of generations used for non-ensemble GP discriminative function classifiers). A simple voting mechanism is implemented with 5 weak learners and a bootstrap sample big as 60% of available instances. The numerical dataset was used.

### 3.2.4. The Comparison of Different GP Variants

By comparing decision-tree-based GP with discriminant-function-based GP the superiority of the first one is obvious on both training and test set. It was expected since the nature of the problem is more adequate to be solved with decision tree as the representation model. Discriminant function, although combined with logical operators, is more appropriate for regression problems. In Fig. 3-6 and 3-7 it can be noticed how discriminant functions suddenly finds better solution while decision tree reaches the best solution gradually. Later will be shown that the execution of one discriminant function based GP for every class was much faster than only one execution of decision tree-based GP.

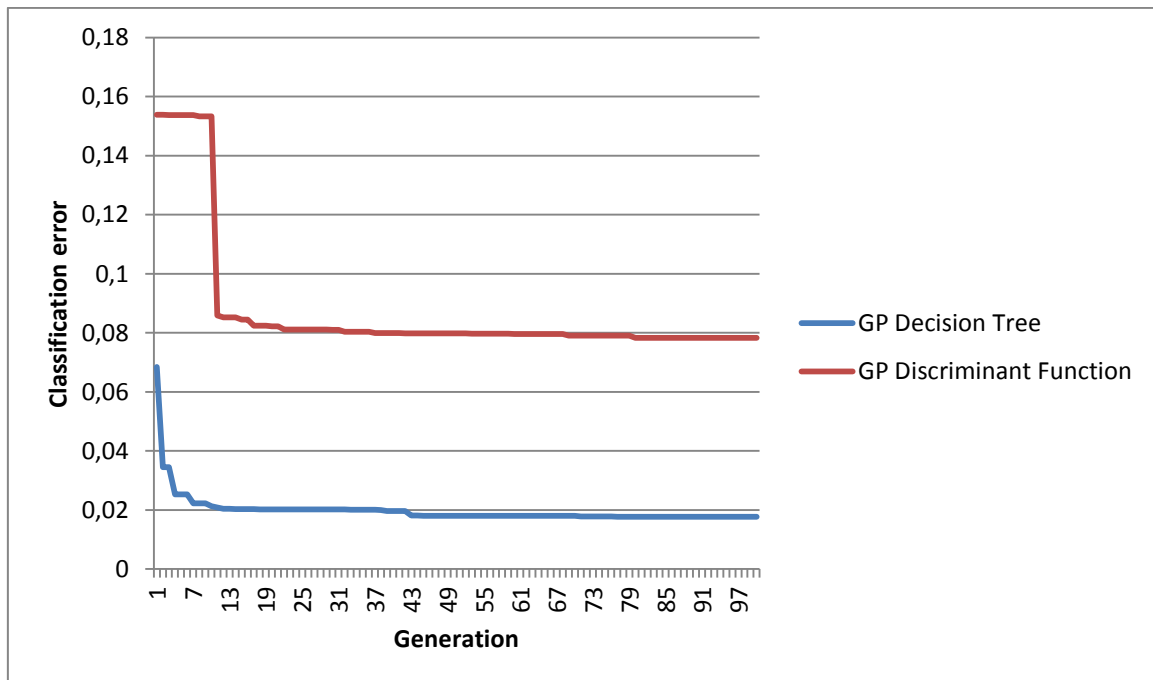


Figure 3-6 Classification error on the training set

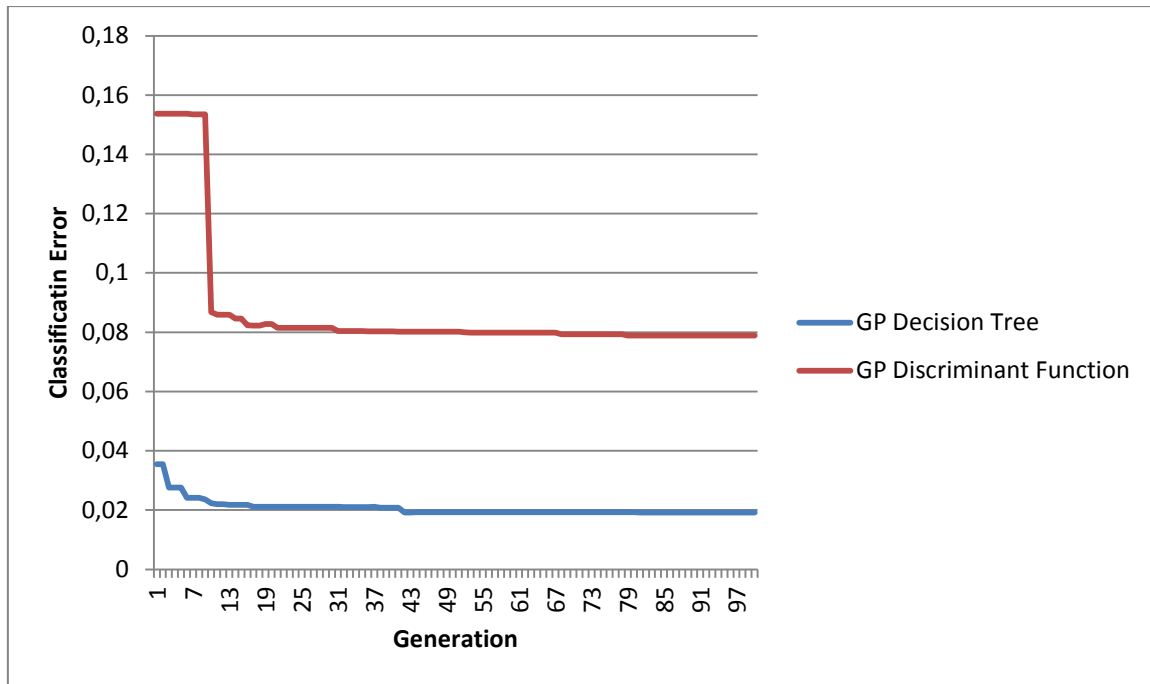


Figure 3-7 Classification error on the test set

### 3.3. Implementation of AdaBoost

The implementation of AdaBoost is based on its original idea from [37]. The maximal number of iterations tried to find classifier with non-zero error is 10. 100% of weight mass was used in training because there was no need for additional speeding-up. For the base classifier a simple one-level decision tree was used, so called decision stump.

### 3.4. Implementation of K-means Clustering

The most important parameter in clustering is the number of clusters. Since the number of classes is known in advance it was automatically assumed that four cluster will fit every user. However, in some user data classes Available for Texting or Busy, or both of them, do not appear at all. After the data was divided into clusters, the match between the labels on each instance and a corresponding cluster had to be made. The chosen match between clusters and labels achieves the best possible accuracy. For some datasets, there were more or less labels on data than clusters so the matching procedure failed. Dividing the dataset to five folds probably increased the chance that rare classes will be missing. The numerical data set was used. The ideal number of cluster for some datasets is given in Table 3-2.

<b>Dataset</b>	2	5	7	9	10	17	26	34	42	50	51
<b>k</b>	4	3	3	4	4	4	3	3	3	4	4
<b>Dataset</b>	60	63	68	75	82	83	109	120	139	141	160
<b>k</b>	4	2	4	3	3	3	4	4	4	2	3

Table 3-2 Values for parameter k in k-means

### 3.5. Implementation of Decision Trees

For the purpose of inferring user status the algorithm very similar to C4.5 was applied on both nominal and arithmetic dataset since it works well with both types. For internal nodes that represent nominal attributes there is one outgoing edge per every possible attribute value. For numerical attributes the outgoing edges are labeled with disjoint edges. The difference between the decision tree built only of nominal attributes and the decision tree that also contains numerical attributes is designated in Figure 3-8. The trees are built using the nominal and the arithmetical dataset from the same random user.

The tree induction algorithm used in this case creates nodes from the root to leaves so that whenever a new node is created it assigns an attribute to that node to maximize the discriminative power of that node measured by a selected criterion. For this specific implementation the sum of information gain and gain ratio was used. The algorithm stops on one of the following conditions: when no attribute reaches certain threshold ( $minimum\_gain = 0.1$ ), when the maximal depth is reached or there are less than a certain number of examples in the current subtree ( $minimal\_size\_for\_split = 4$ ). For the parameter *maximal\_depth* the optimal value for every dataset is separately determined from the values {5, 9, 13, 16, 20} using another nested 5-fold cross validation. In the end the tree is pruned and leaves that do not bring any discriminative power are removed.

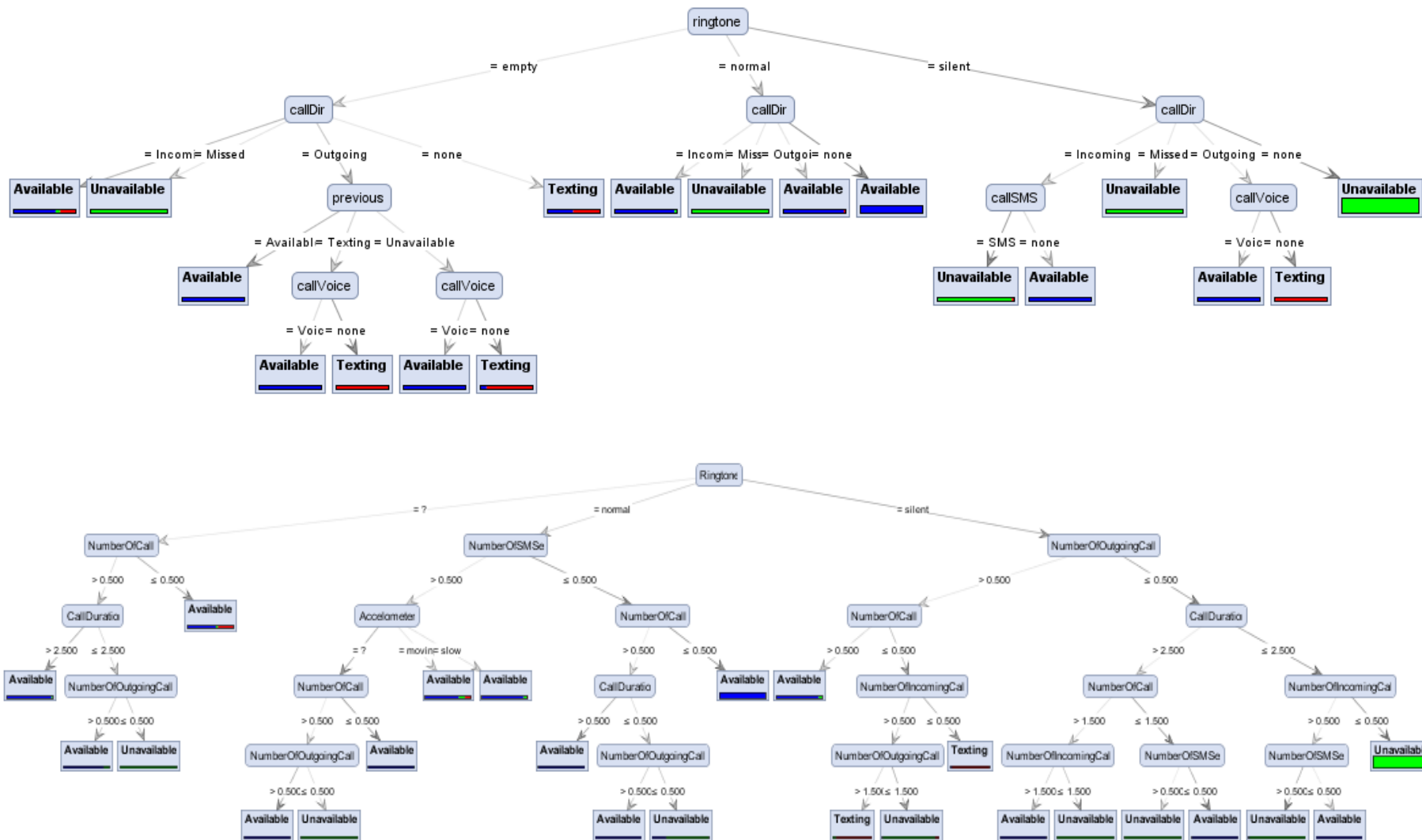


Figure 3-8 Decision trees for nominal and arithmetical dataset



### **3.6. Implementation of K-Nearest Neighbors**

Parameter  $k$  was optimized through six iterations from 1 to 100. The optimization of parameter  $k$  is in fact the only task of the training phase in k-NN. Once the best  $k$  is determined the training time becomes negligible, but the classification of an unseen example is more demanding and it depends on the number of training examples. The algorithm requires a lot of memory for a large number of training examples because all the training examples are necessary for the classification. The numeric dataset and Euclidean distance were used. When the parameter  $k$  is set to 1, the 1-NN algorithm can be applied for classifying the unseen instance in short time until the number of training examples surpasses certain value. Therefore, this case was separately observed in order to compare its accuracy with the accuracy of k-NN with optimal  $k$ .

### **3.7. Implementation of Naïve Bayes**

Naïve Bayes learner is optimized by only on one parameter, whether to use Laplace correction to prevent high influence of zero probabilities or not. The model with better accuracy is chosen.

## 4. Results and Comments

In this chapter the final results and conclusions will be given. Here are some questions to be answered:

- Which algorithms reach the highest accuracy on the given dataset?
- Which algorithms are applicable given their training and test time?
- How accuracy behaves on merged datasets?
- What are the consequences of removing the energy inefficient features?
- Which subset of features is the best for certain algorithms?
- Where and how is each algorithm applicable?

Algorithms are grouped into GP and non-GP algorithms and compared among each group. The duration of training phase is measured as the duration of the whole training process with 5 validation iterations while the accuracy is calculated as the average value of accuracies achieved in different iterations.

### 4.1. Accuracy and Performance

#### 4.1.1. GP algorithms

The accuracy results of tested GP algorithms are shown in Table 4-1. Among GP algorithms the highest accuracy was reached for GP based on decision trees. It is followed by GP based on discriminative functions (1-against all) with stopping criteria of fitness bigger than 0.99 or the number of generations bigger than 200. GP ensemble consisted of undertrained discriminative multiclass functions outperformed the less trained 1-against-all discriminative functions and multiclass discriminative functions. Ensemble of weak learners justified the expectations and the simple voting of the worst GP-based classifiers improved its average accuracy by 15%.

The difference between GP variants is even more visible on box plots designated in Fig. 4-1 and Fig. 4-2. In Fig. 4-1 the population of each box consists of 10 accuracy results, one for each run of GP on that specific user. In Fig. 4-2 the population of each box consists of 38 average accuracy results, one for each user and each one representing the average of 10 executions on one user.

Genetic programming has many advantages, but real-time training is definitely not one of them. Table 4-2 shows the duration of training process for each of the first four GP algorithms. GP decision tree reached the best accuracy and the worst duration of the training process. Once trained, the best individuals can easily be applied to the classification of the unseen individuals even in real-time.

Algorithm	GP Decision Trees	GP 1-against-all Discrim. Func. (max fitness 0.99)	GP Ensemble Bagging	GP 1-against-all Discrim. Func. (max fitness 0.9)	GP Multiclass Discrim. Func.
<b>AVG</b>	0.988695	0.983017891	0.976407318	0.975467426	0.848965
<b>MEDIAN</b>	0.992321	0.99147385	0.988776255	0.988376924	0.945528
<b>MAX</b>	0.99981	0.999784	0.999784	0.999775	0.999607
<b>MIN</b>	0.944881	0.869200116	0.869481336	0.869129935	0.494651

Table 4-1 Accuracy results for different GP types

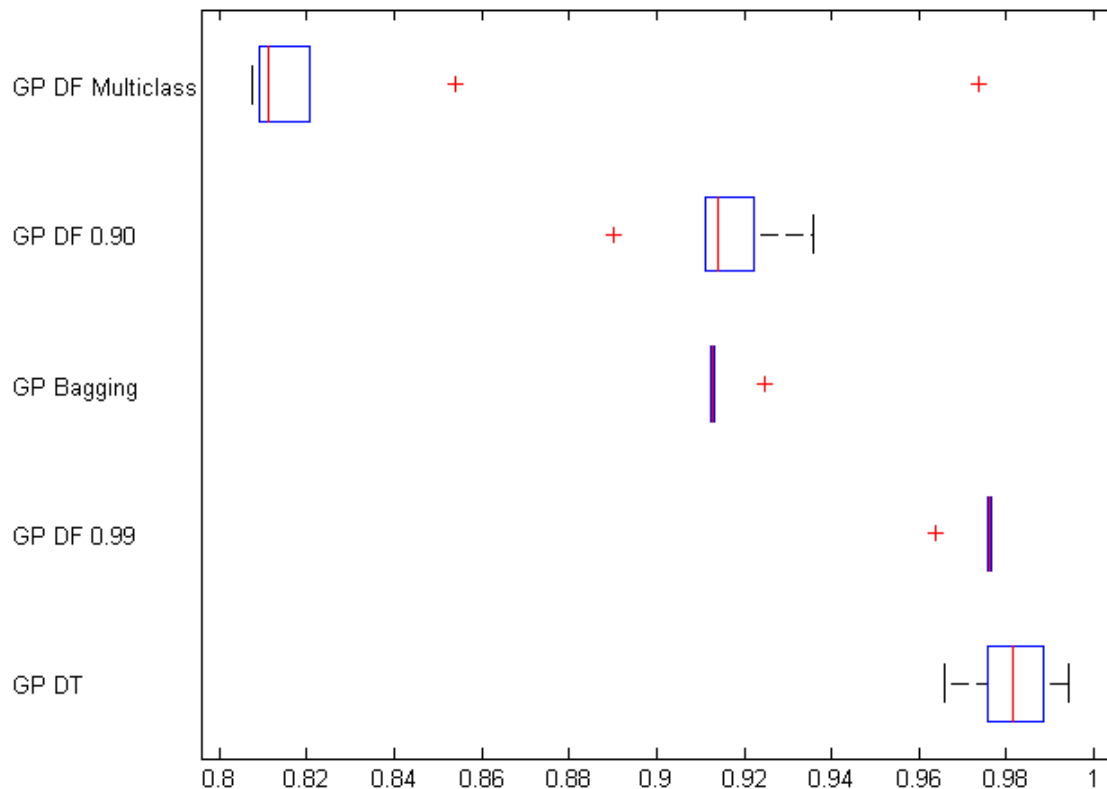


Figure 4-1 Box plot of GP accuracy on one random user

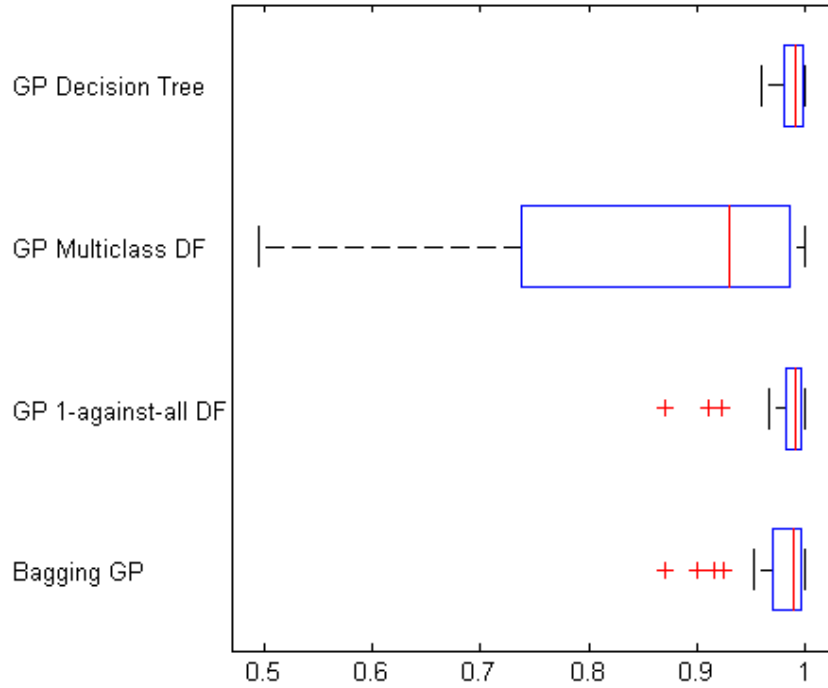


Figure 4-2 Box plot of GP accuracy on the average of every user

Algorithm	GP 1-against-all Discrim. Func. (max fitness 0.9)	GP 1-against-all Discrim. Func. (max fitness 0.99)	GP Ensemble Bagging	GP Decision Trees
<b>AVG</b>	0 01:30:06	0 03:15:58	0 02:07:17	0 19:34:56
<b>MEDIAN</b>	0 00:37:22	0 02:25:26	0 00:47:59	0 19:15:30
<b>MAX</b>	0 14:21:32	0 14:07:23	0 23:39:10	2 08:25:00
<b>MIN</b>	0 00:01:00	0 00:03:01	0 00:03:02	0 02:31:00
<b>Category</b>	>1 hour	>1 hour	>1 hour	>>1 hour

Table 4-2 Training execution time for different GP types (d hh:mm:ss)

#### 4.1.2. Non-GP Algorithms

The best non-GP algorithms considering the accuracy are C4.5, AdaBoost based on decision stumps and k-nearest neighbors. C4.5 is slightly better when applied to nominal dataset than arithmetical dataset (Table 4-3). Regarding the great success of AdaBoost, the incremental learning algorithm Learn++ from the chapter 2.9.1, that is based on the very same principle as AdaBoost, is expected to be equally accurate, but it was not implemented.

Although in some relative work naïve Bayes was successfully applied, in this case its accuracy is very bad, around 50%. Some features from the arithmetical dataset that were

used for naïve Bayes classifier are dependent, for instance the number of calls is proportional with call duration, and this is partly the reason of such lousy performance of naïve Bayes because its naïve assumption of independent variables is not correct. K-means was tried just to get the impression about the data itself when the labels are unknown at first. The number of distinct labels  $M$  differs from user to users and each dataset was clustered to  $M$  clusters. The accuracy of 65% shows that the decision boundaries between datasets overlap. It could be improved by another feature subset or by changing the way of data labeling. The box plot designated in Fig. 4-3 shows how worse naïve Bayes is than the rest of the classification algorithms. In order to better compare the top three classifiers their accuracy is designated in Fig. 4-4. Each box represents the population of 38 accuracy results, one for each user. C4.5, AdaBoost, 1-NN and k-NN with optimal k are more-less equally successful if accuracy is the only criteria.

When compared by training time, AdaBoost outperforms all the other algorithms (which require training). When AdaBoost and C4.5 are compared as the two best candidates for real-time presence status inferring, AdaBoost is about 14 times faster than C4.5, although every algorithm whose median is less than 10 minutes is the candidate for applying to mobile devices (Table 4-4).

Note that the training time of k-nearest neighbors is in fact the time spent on optimizing the parameter k. Apart from optimizing the parameter k k-NN does not actually require any training. The problematic part with k-NN is the duration of classifying the unseen instance which depends on the number of training instances. Classifying the new instance is linear in the size of the training set as we need to compute the distance of each training instance from the test instance. However the test time decreases for small k and therefore the 1-NN was compared with k-NN with optimal k. The accuracy of 1-NN does not lag behind optimal k-NN very much which is very good news because we have another candidate for applying to inferring presence status directly on mobile devices.

Algorithm	C4.5 Nom.	C4.5 Arith.	AdaBoost Arith.	k-NN Arith.	1-NN Arith	k-means Arith.	Naïve Bayes Arith.
<b>AVG</b>	0.9892	0.9862	0.9857	0.9826	0.9794	0.6650	0.5510
<b>MEDIAN</b>	0.9977	0.9947	0.9912	0.9944	0.9957	0.6501	0.4877
<b>MAX</b>	0.9998	0.9997	0.9999	0.9997	0.9999	0.9705	0.9954
<b>MIN</b>	0.8912	0.8665	0.9076	0.8195	0.7304	0.3905	0.0914

Table 4-3 Accuracy for non-GP algorithms on arithmetic dataset

Algorithm	AdaBoost Arith.	Naïve Bayes Arith.	C4.5 Nom.	k-means Arith.	C4.5 Arith.	k-NN Arith.
<b>AVG</b>	00:00:21	0:02:32	0:07:22	0:06:03	0:24:45	1:45:28
<b>MEDIAN</b>	00:00:17	0:00:25	0:04:00	0:04:00	0:16:30	0:59:30
<b>MAX</b>	00:01:08	0:28:00	0:44:00	0:10:00	1:32:00	6:10:00
<b>MIN</b>	00:00:05	0:01:00	0:01:00	0:02:00	0:02:00	0:14:00
<b>Category</b>	<10 min	<10 min	<10 min	<10 min	>10 min <1 hour	>1 hour

Table 4-4 Training execution time for non-GP algorithms (hh:mm:ss)

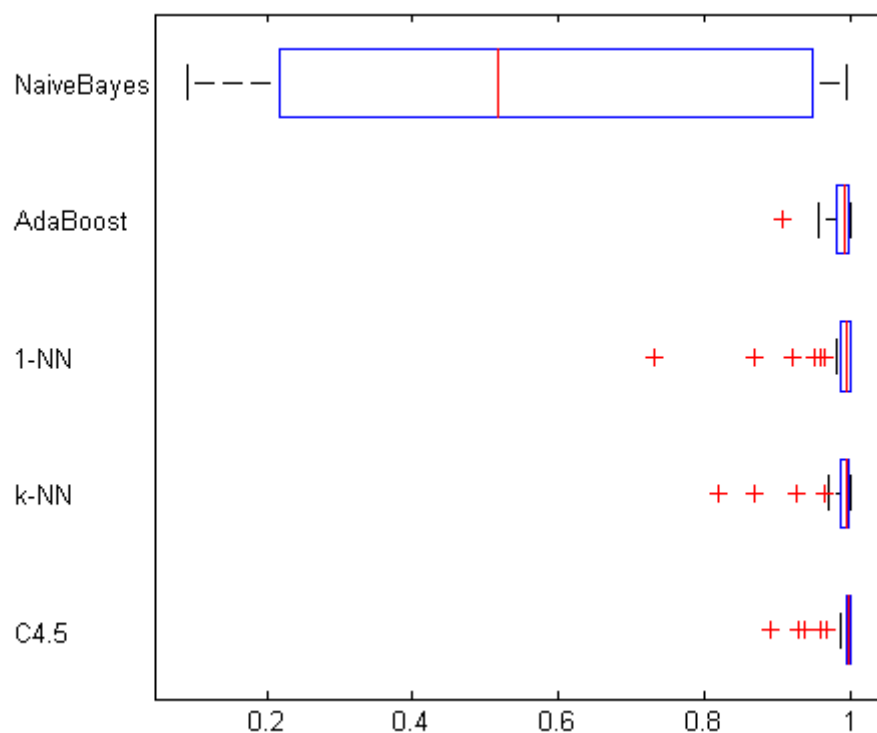


Figure 4-3 Box plot of classification accuracy of non-GP algorithms

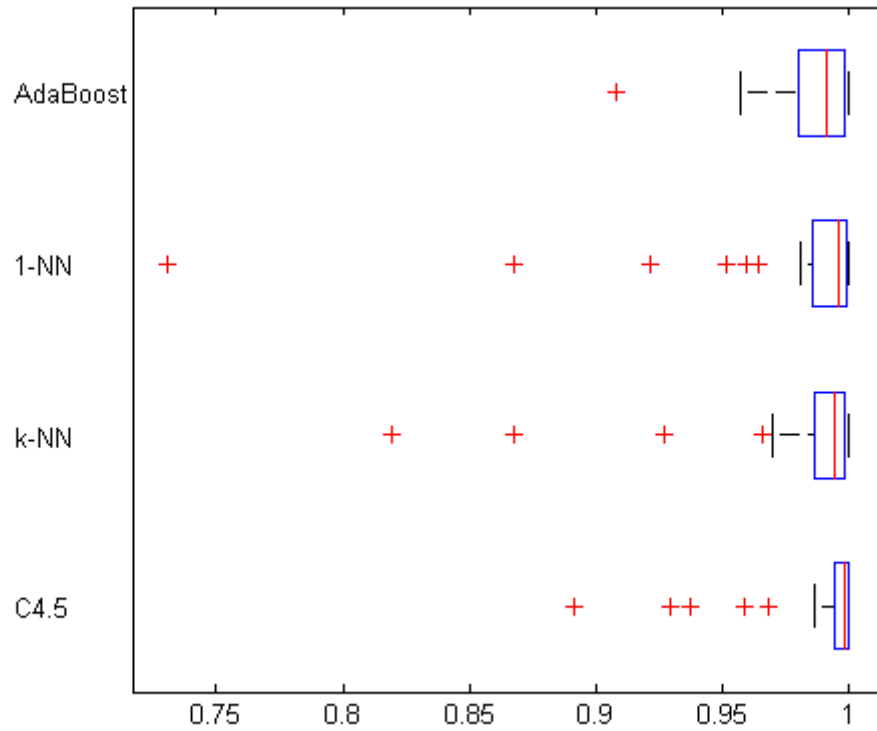


Figure 4-4 Box plot of classification accuracy of the best non-GP algorithms

### 4.1.3. Results on Merged Datasets

An interesting remaining question is how accuracy behaves on datasets consisted of many users' datasets. It was tested with the best GP based on discriminative functions and with AdaBoost. The median was decreased by 5% with GP and by 6% with AdaBoost when tested on merged datasets. The execution time of GP is terrifying, but AdaBoost's average training time is still under 10 minutes (Table 4.5). Therefore, to further examine AdaBoost's behavior it was tested on different merged datasets. The comparison of results starting from the best user up to 20 best users it designated in Fig. 4-5. Accuracy on merged dataset is lower than the average accuracy on separated datasets, but still very high (> 99%). To try a more realistic ensemble of users, 20 random users were chosen and the same procedure was repeated but this time the order of users was not anyhow sorted. The accuracy on merged dataset significantly reduced compared with average accuracies. However, the accuracy above 94% is still very good and it opens up an opportunity to use the accumulated data from multiple users for building the classifier.

To clarify Figures 4-5 and 4-6 an additional explanation is provided. In Fig. 4-5 the top 20 results sorted by accuracy were taken. The first column is the same because the dataset is

equal and it consists of the data from only one user. The second column in the graph is in the first case the result of the algorithm on one data set consisted of the datasets from the two first users, while in the second case the value is calculated as the average value of two separate executions of AdaBoost, each one on separate dataset. The last column contains the accuracy on the accumulated dataset that contains the data of 20 different users and the average value of 20 different executions of AdaBoost on the separate datasets.

The difference between Fig. 4-6 and Fig. 4-5 is only in the order and choice of datasets, the procedure remains the same.

Algorithm	GP Discrim. Func.	AdaBoost
Average Accuracy	0.943327462	0.92785842
Median Accuracy	0.941023552	0.92781306
Average Execution Time (dd hh:mm:ss)	15 09:32:00	00 00:09:53

Table 4-5 Results on merged dataset of 15 random users

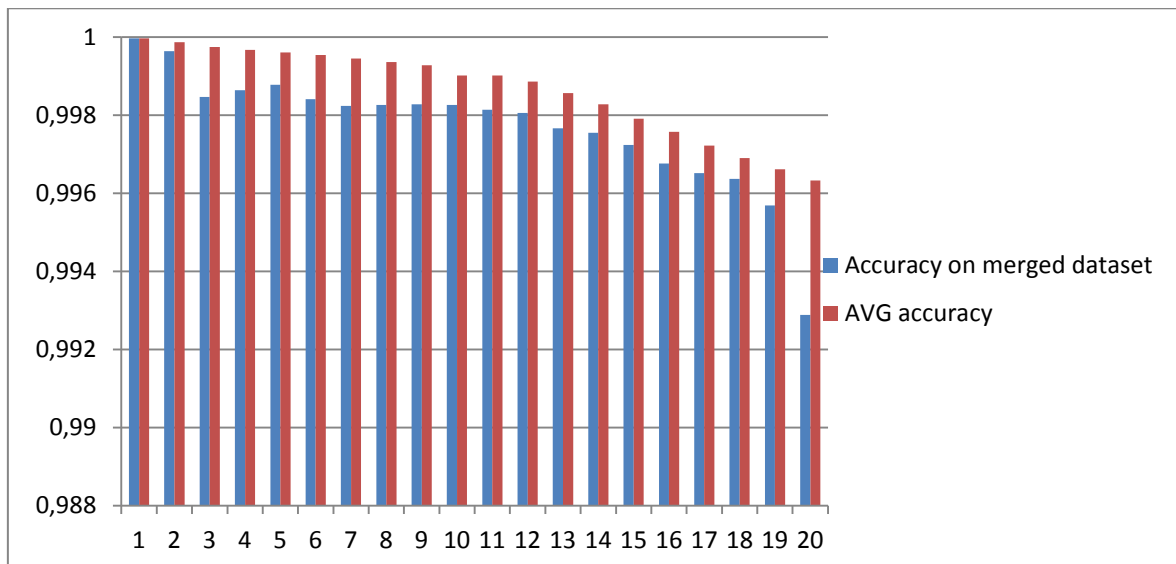


Figure 4-5 The comparison of AdaBoost's average accuracy on separate datasets with accuracy on the merged dataset (top 20 users)



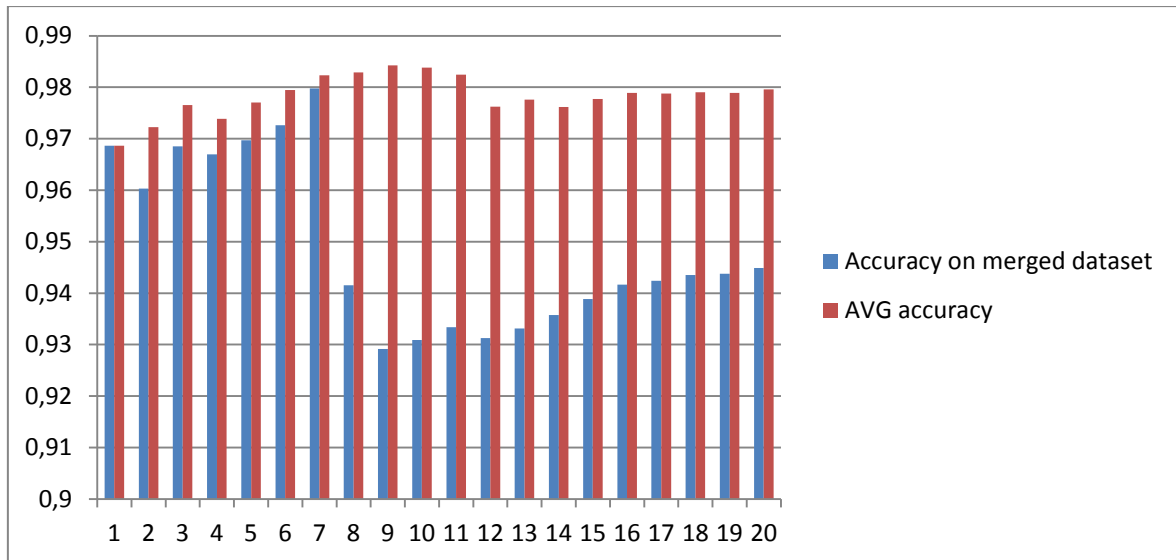


Figure 4-6 The comparison of AdaBoost's average accuracy on separate datasets with accuracy on the merged dataset (random 20 users)

## 4.2. Feature Selection and Energy Efficiency

The most energy consuming features available in MDC Dataset are GPS, GSM (but not GSM cell identifier), accelerator and WiFi. In this thesis only accelerator was used among them. The best accuracy was noticed on decision-tree-based GP among GP algorithms and on C4.5 and AdaBoost among non-GP algorithms.

By comparing the values from Table 4-7 with all the features included with Table 4-6 which is calculated without using the accelerator as a feature it shows that after removing the accelerator there is no significant change. In some aspects removing the accelerator even increases the accuracy.

To investigate such behavior and possibly to increase the accuracy and reduce the training time further feature selection was made. Since there are only 10 features in nominal dataset, even the exhaustive search feature selection is applicable. It selects the best features for an example set by trying all possible combinations of features. The aggregated results are shown in Table 4-8. For every feature the percentage and a number of users which use it after feature selection is given (the total number of users is 38). The results explain why the accuracy did not change after removing the accelerator – it is the least frequently used feature used by only 23.68% of users. Call direction, ringtone and previous status are all used by around 90% of the users. For two users only the call direction is relevant and the best accuracy is achieved by that single feature and this is the reason why

call direction is the most frequently used (97.37%). In average the perfect feature set contains approximately 6 features, but the ideal feature set differs a lot from users to user. The complete result of exhaustive search for every user can be found in addition to this thesis.

A result achieved on the optimal feature set optimized separately on each dataset is given in Table 4-9. It is the best accuracy that has been achieved so far, and the training time is reduced together with the number of features.

<b>Algorithm</b>	<b>GP Decision Trees</b>	<b>C4.5</b>	<b>AdaBoost</b>
<b>AVG</b>	0.989335	0.987869	0.982108968
<b>MEDIAN</b>	0.992707	0.997423	0.985077443
<b>MAX</b>	0.999916	0.999814	0.999460656
<b>MIN</b>	0.948193	0.891289	0.907560541

Table 4-6 Accuracy of on the subset of 19 users **without** accelerator

<b>Algorithm</b>	<b>GP Decision Trees</b>	<b>C4.5</b>	<b>AdaBoost</b>
<b>AVG</b>	0.988695	0.987193	0.974815
<b>MEDIAN</b>	0.992321	0.997423	0.986346
<b>MAX</b>	0.99981	0.999814	0.999654
<b>MIN</b>	0.944881	0.891188	0.866526

Table 4-7 Accuracy of on the subset of 19 users with **all** features

<b>Call Direction</b>	<b>Ringtone</b>	<b>Previous Status</b>	<b>Voice</b>	<b>SMS</b>
97.37% (37)	92.11% (35)	89.47% (34)	76.31% (29)	60.53% (23)
<b>Calend. Status</b>	<b>Calend. Class</b>	<b>Time of Day</b>	<b>GSM cell ID</b>	<b>Accelerator</b>
55.26% (21)	47.37% (18)	44.74% (17)	42.11% (16)	23.68% (9)

Table 4-8 Results of exhaustive search feature selection – percentage of users that contain the feature in their ideal feature set

<b>AVG</b>	<b>MEDIAN</b>	<b>MAX</b>	<b>MIN</b>
0.994032016	0.998541588	0.999969779	0.891715266

Table 4-9 The accuracy of C4.5 on optimized feature sets

### 4.3. Comments

Table 4-10 shows where can training phase be performed for each algorithm considering the training and test time for each algorithm. Algorithms who can be trained on devices are the ones executed in less than 10 minutes which do not require too much memory resources. Based on the results from the previous chapter, AdaBoost based on decision stumps fits the best in that category, but C4.5 is also a good candidate. Since AdaBoost showed extraordinary performance and accuracy, Learn++ has a great potential to be even more successful but the possibility of incremental learning. GP has some extensions for incremental evolution but its implementation on mobile device is not a way to go because it requires too many resources. Some k-NN incremental methods have been applied to activity recognition problems but it still faces some issues with accuracy, it depends on the order of the training instances and k-NN in general requires too much memory for saving the training set if it is large. Other incremental learning algorithms exceed the subject of this thesis.

Regarding the test phase, as shown in Table 4-11, all the algorithms except k-NN have a good predisposition to easily classify unseen instances on the device. It opens up an opportunity to train any model in the cloud and then to transfer it to mobile device where it can be applied to new instances. Note that such approach requires extra network traffic which can also drain out the battery.

TRAINING PHASE	Device	Incremental Learning on a Device	Cloud
<b>Genetic Programming (all types)</b>	Not applicable	Not applicable	Applicable
<b>Decision Tree C4.5</b>	Applicable	Possible with major changes [41]	Applicable
<b>K-NN</b>	Not applicable for big datasets	Not (yet) applicable	Applicable
<b>AdaBoost</b>	Applicable	Possible (Learn++)	Applicable

Table 4-10 Characteristics of training phase

TEST PHASE	Device	Cloud
<b>Genetic Programming (all types)</b>	Applicable	Applicable
<b>Decision Tree C4.5</b>	Applicable	Applicable
<b>K-NN</b>	Not applicable for big datasets	Applicable
<b>AdaBoost</b>	Applicable	Applicable

Table 4-11 Characteristics of test phase

In the end what was achieved and concluded in this thesis is following:

- The most appropriate machine learning algorithms for inferring presence status were examined and tested. AdaBoost, C4.5 and 1-NN showed to be the top three algorithms with the best combination of accuracy and execution time of training and testing, respectively. Learn++ is the incremental learning variant of AdaBoost and if only one algorithm has to be chosen, Learning++ is suggested for use in real applications on mobile devices.
- Different genetic programming approaches were tested including the ensemble of weak discriminative functions trained on relatively small number of generations. The most accurate and the slowest one was decision-tree-based GP.
- The problem of the small representation of classes Texting and Busy exists in some datasets.
- While labeling the data, Busy was determined as the state which occurs if the user sets its ringtone to beep or ring once. Those are specific ringtone types on old Nokia's smartphones from 2011 when the dataset was collected. It is not applicable to the majority of today's smartphones. The automaton for labeling should therefore be updated.
- The best feature subset was identified. The most discriminative features in 90% of datasets are 1) call direction, 2) ringtone and 3) previous status. That is not surprising when the way of labeling the data is observed (chapter 1.3). What is

surprising is that further feature removing in order to achieve energy efficiency did not have to be made because exhaustive feature selection algorithm has already marked the accelerator as unnecessary feature in almost 80% of datasets, so it would be removed anyways. Accelerator and GSM are used on three transitions on the automaton for labeling the data and yet their contribution was not significant. The reason might be aggregating the data into 5 minute windows.

- To get the real-life impression of the most common user behavior further testing on real users with their interaction should be made. If the decisions of classifier previously trained on the data labeled by automaton agree with real user interventions then the final conclusions about the success of this approach can be brought. When a small amount of real data is collected, it can be used for labeling the examples from the Mobile Data Challenge dataset by clustering the union of labeled and unlabeled examples and afterwards labeling the unlabeled examples by the majority in their cluster. It is a common semi-supervised learning approach.
- The accuracy on merged datasets remains satisfying for chosen algorithms. It possibly offers an opportunity to train the classifier on a prepared dataset before deploying it to devices. It solves the problem of the lack of training data at the beginning of the learning process.
- What is completely new in the dataset used in this thesis, compared to the one from [1], is adding the time of day as one of the features. Its inclusion was beneficial in 44.74% of datasets which is better than both accelerator and GSM (23.68% and 42.11%).

## Conclusion

Inferring presence status has yet unused potential to a variety of different communication services. Its inferring could be used within a closed social network or an organization and it could make the communication between presentities more efficient. With this application as the final goal, different machine learning algorithms were tried on the dataset from the Mobile Data Challenge dataset. The data was aggregated in 5 minute windows to reduce the status change and it was labeled using the approach in [1]. Genetic programming algorithms intended only for offline learning showed the best accuracy with the decision tree as representation of the individual, but the training time is too long even for offline learning. The other used GP types are a single discriminative function used as multiclass classifier, a combination of discriminative functions in ensemble with simple voting mechanism, and combination led by 1-against-all approach. Non-GP algorithms tried are decision tree induction (C4.5), k-means which revealed the problem of missing classes in some datasets, k-nearest neighbors, naïve Bayes and AdaBoost. Although naïve Bayes was successfully applied to some related activity recognition problems, it failed on inferring presence status by showing almost random behavior. AdaBoost, C4.5 and k-NN (or 1-NN) showed high accuracy and execution time acceptable even for mobile devices. An incremental learning algorithm proposed for future application is Learning++.

Accelerometer was detected as almost unnecessary feature which is great news because it is at the same time the most energy consuming feature. The exhaustive feature selection based on decision trees was performed and call direction, ringtone, and previous status are detected as the most discriminative features. Although datasets differ a lot from each other, some general rules can be applied to a union of datasets from different users. The question whether that happens because the datasets were labeled using the very same automaton which implies certain rules remains. Since the rules for labeling the data were known from the very beginning, to justify the results further examination of the approach should be made on real users. The suggested approach is to implement decision stump Learn++ to mobile devices, with the possibility of prompting the users for presence status self-reports at random, and to previously train it on the preprocessed nominal dataset consisted of several datasets from Mobile Data Collection.

## References

- [1] A. ANTONIĆ, D. JAKOBOVIĆ, I. PODNAR-ŽARKO, Inferring Presence Status on Smartphones: The Big Data Perspective, IEEE Symposium on Computers and Communications, 2013.
- [2] Presence information, [http://en.wikipedia.org/wiki/Presence\\_information](http://en.wikipedia.org/wiki/Presence_information), 22.05.2014.
- [3] Presentity, <http://en.wikipedia.org/wiki/Presentity>, 22.05.2014.
- [4] There Will Soon Be One Smartphone for Every Five People in the World, <http://www.businessinsider.com/15-billion-smartphones-in-the-world-22013-2>, 22.05.2014.
- [5] S. CONSOLVO, D. W. McDONALD, T. TOSCO, M. Y. CHEN, J. FROELICH, B. HARRISON, P. KLASNJA, A. LAMARCA, L. LEGR, R. LIBBY, I. SMITH, J. A. LANDAY, Activity Sensing in the Wild: a Field Trial of UbiFit Garden. In CHI'08. Florence, Italy, 2008.
- [6] T. CHOUDHURY ET AL., The Mobile Sensing Platform: An Embedded System for Capturing and Recognizing Human Activities, IEEE Pervasive Computing Special Issue on Activity-Based Computing, 2008.
- [7] V. PEJOVIĆ, M. MUSOLESI, Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges, CoRR abs/1306.2356 2013.
- [8] S. ABDULLAH, N. D. LANE, T. CHOUDHURY, Towards Population Scale Activity Recognition: A Scalable Framework for Handling Data Diversity, AIII'12. 2012.
- [9] U. MAURER, A. ROWE, A. SMAILAGIĆ, D. P. SIEWIOREK, eWatch: A Wearable Sensor and Notification Platform, BSN '06, 2006.
- [10] J. FOGARTY, S. E. HUDSON, C. G. ATKESON, D. AVRAHAMI, J. FORLIZZI, S. KIESLER, J. C. LEE, J. YANG, Predicting Human Interruptibility with Sensors, ACM Transactions on Computer-Human Interaction, 2005.

- [11] J. K. LAURILLA, D. GATICA-PEREZ, I. AAD, J. BLOM, O. BORNET, T.-M. T. DO, O. DOUSSE, J. EBERLE, M. MIETTINEN, The mobile data challenge: Big data for mobile computing research, Proc. Mobile Data Challenge by Nokia Workshop, 2012.
- [12] M. DE DOMENICO, A. LIMA, M. MUSOLESI, Independence and Predictability of Human Mobility and Social Interactions, Proc. Of the Nokia Mobile Dana Challenge Workshop, 2012.
- [13] Mobile Dana Challenge 2012 Workshop, <https://research.nokia.com/page/12340>.
- [14] J. LIU, B. PRIYANTHA, T. HART, H. S. RAMOS, A. AF LOUREIRO, Q. WANG, Energy Efficient GPS Sensing with Cloud Offloading, SenSys'12, 2012.
- [15] Changing my online status, <https://support.google.com/talk/answer/23917?hl=en>, 26.05. 2014.
- [16] Google Talk, [http://en.wikipedia.org/wiki/Google\\_Talk](http://en.wikipedia.org/wiki/Google_Talk), 26.05. 2014.
- [17] Genetic Programming, <http://www.genetic-programming.org/>, 26.05. 2014.
- [18] D. BAŠIĆ, J. ŠNAJDER, Strojno učenje, FER, Zagreb, 2012.
- [19] J. R. KOZA, Concept formation and decision tree induction using the genetic programming paradigm, Parallel Problem Solving from Nature - Proceedings of 1st Workshop, Volume 496, Springer – Verlag, str. 124 – 128, 1991.
- [20] J. R. KOZA, Genetic Programming – on the programming of computers by means of natural selection, MIT Press, Cambridge, 1992.
- [21] P. G. ESPEJO, S. VENTURA, F. HERRERA, A Survey on the Application of Genetic Programming to Classification, IEEE Transactions on Systems, Man and Cybernetics, 2010.
- [22] M. WALKER, Introduction to Genetic Programming, 2001.
- [23] H. JABEEN, A. R. BAIG, Review of Classification Using Genetic Programming, International Journal of Engineering Science and Technology Vol.2(2), 2010, 94-103.
- [24] J. R. QUINLAN, Induction of decision trees, Mach. Learn. 1, 1, Machine Learning, 81 – 106, 1986.



- [25] C. M. BISHOP, Pattern Recognition and Machine Learning. Springer, New York, NY, USA, 2006.
- [26] R. E. MARMELESTEIN, G. B. LAMONT, Pattern Classification Using a Hybrid Genetic Program – Decision Tree Approach, Proc. 3rd Annu. Conf. Genet. Program., San Mateo, CA: Morgan Kaufmann, 1998., pp. 223-231
- [27] K. HENNESSY, M. G. MADDEN, J. CONROY, A. G. RYDER, An Improved Genetic Programming Technique for the Classification of Raman Spectra, Know. Based Syst., vol. 18, no. 4-5, pp. 217-224, 2005.
- [28] I. STOKIĆ, Primjena genetskog programiranja u strojnom učenju, Master thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2011.
- [29] R. S. SUTTON, A. G. BARTO, Reinforcement Learning: An Introduction Cambridge, MA: MIT Press, 1998.
- [30] Z.-H. ZHOU, Ensemble learning, S. Z. Li ed. Encyclopedia of Biometrics, Berlin: Springer, 2009, 270-273.
- [31] Boosting: [http://en.wikipedia.org/wiki/Boosting\\_\(machine\\_learning\)](http://en.wikipedia.org/wiki/Boosting_(machine_learning)), 26.05. 2014.
- [32] R. E. SCHAPIRE, Explaining AdaBoost, in Bernhard Schölkopf, Zhiyuan Luo, Vladimir Vovk, editors, Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik, Springer, 2013.
- [33] TER HOFTE, Xensible interruptions from your mobile phone, Mobile HCI'07, Singapore, 2007.
- [34] DASH, M., H. LIU, Feature selection for classification. Intelligent Data Analysis, 1(1-4), 131-156, 1997.
- [35] Building Classification Models: ID3 and C4.5, <http://www.cis.temple.edu/~giorgio/cis587/readings/id3-c45.html>, 26.05. 2014.
- [36] H. C. PENG, F. LONG, C. DING, Feature selection based on mutual information: criteria of max-dependency, max-relevance and min-redundancy, IEEE Transactions on Pattern Analysis and Machine Learning Intelligence, 27 (8): 1223-1238, 2005.
- [37] Y. FREUND, R. E. SCHAPIRE, Experiments with a new boosting algorithm, Proc. International Conference on Machine Learning, 148-156, Morgan Kaufmann, San Francisco, 1996.

- [38] R. POKLAR, L. UDPA, S. S. UDPA, V. HONAVAR, Learn++: An Incremental Learning Algorithm for Supervised Neural Networks, IEEE Transactions on Systems, Man and Cybernetics, Vol. 31, No. 4, 2001.
- [39] R. R. ADE, P. R. DESHMUKH, Methods for Incremental Learning: A Survey, International Journal of Data Mining & Knowledge Management Process, Vol. 3, No.4, 2013.
- [40] K. FOERSTER, S. MONTELEONE, A. CALATRONI, D. ROGGEN, G. TROESTER, Incremental kNN classifier exploiting *correct - error* teacher for activity recognition, International Conference on Machine Learning and Applications
- [41] P. E. UTGOFF, Incremental Induction of Decision Trees, Machine Learning, 4, 161-186, 1989.

# Summary

## Inferring Presence Status on Mobile Devices

Keywords:

presence status, mobile computing, machine learning, ensemble learning, genetic programming

Automatically inferring presence status on smartphones without user intervention is applicable in many IM and VoIP communication services, but it has not yet been implemented in any of them.

It is a problem that requires accurate and energy efficient machine learning methods and a well-chosen subset of features. The objective of this thesis was to test several classification algorithms and to compare them by different criteria: accuracy, training time, test time and the possibility of incremental learning on the given dataset. All the given criteria meet in boosting algorithm AdaBoost based on decision stumps, in fact in its incremental edition called Learn++. C4.5 and 1-NN are also possible candidates, with some necessary adjustments. Several GP variants were tried for a potential use after offline training phase and decision tree based GP showed the highest accuracy, but the slowest training time.

The accuracy on accumulated datasets from different users does not decrease significantly. The exhaustive feature selection method was performed and within more than 80% of users only the energy efficient features are necessary to achieve the best possible accuracy.

# Sažetak

## Određivanje statusa korisnika mobilnih uređaja

Ključne riječi:

status prisutnosti, računarstvo u pokretu, strojno učenje, učenje ansamblom, genetsko programiranje

Automatsko određivanje statusa korisnika mobilnih uređaja bez interakcije sa samim korisnikom primjenjivo je u brojnim komunikacijskim VoIP uslugama i uslugama trenutnog poručivanja, ali u praksi nije još zaživjelo.

Radi se o problemu koji zahtijeva točne i energetske učinkovite metode strojnog učenja te ispravno odabran podskup značajki. Cilj rada bio je isprobane klasifikacijske algoritme usporediti po sljedećim kriterijima: točnost, trajanje učenja, trajanje testiranja te mogućnost nadoučavanja na zadanom skupu podataka. Za najpogodniji algoritam s obzirom na zadane kriterije odabran je algoritam AdaBoost temeljen na stablima odluke sa samo jednom razinom, odnosno njegova inačica koja nudi mogućnost postupnog učenja, Learn++. C4.5 i 1-NN također su mogući kandidati, ali uz određene prilagodbe. Isprobano je nekoliko varijanti genetskog programiranja te su stabla ostvarila najveću točnost, ali i najduže učenje, stoga je GP pogodan isključivo za primjenu na mobilnim uređajima nakon završenog proces učenja.

Točnost na agregiranom skupu podataka od različitih korisnika ne smanjuje značajno. Iscrpnom pretragom podskupova značajki kod više od 80% testiranih korisnika jedino su energetske učinkovite značajke bile potrebne kako bi se ostvarila najbolja moguća točnost.

## Abbreviations

IM	Instant Messaging	<i>trenutno poručivanje</i>
AdaBoost	Adaptive Boosting	
k-NN	k-Nearest Neighbors	<i>k-najbližih susjeda</i>
MDC	Mobile Data Challenge	
SMS	Short Message Service	<i>kratka poruka</i>
WLAN	Wireless Local Area Network	<i>lokalna mreža koja se zasniva na bežičnim tehnologijama</i>
GPS	Global Positioning System	<i>globalni navigacijski sustav</i>
GSM	Global System for Mobile Communications	
GP	Genetic Programming	<i>genetsko programiranje</i>
GA	Genetic Algorithm	<i>genetski algoritam</i>
mRMR	Minimum Redundancy	<i>minimalna zalihost</i>
	Maximum Relevance	<i>maksimalna redundantnost</i>
LDCC	Lausanne Data Collection Campaign	