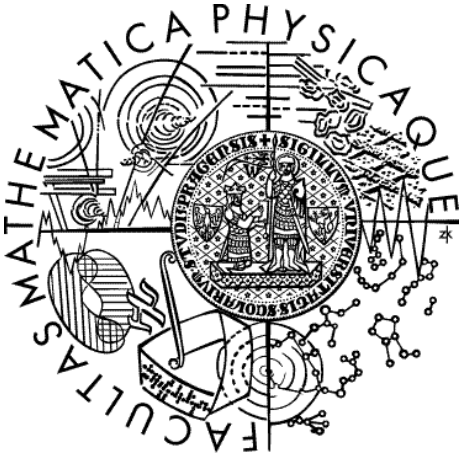
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Juraj Figura

Machine Learning for Google Android

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: RNDr. Ondřej Bojar Ph.D.

Study programme: Computer Science

Specialization: General Computer Science

Prague 2012

I would like to thank RNDr. Ondřej Bojar Ph.D. for his supervision, valuable advices, ideas and help. Also, I would like to thank my family for the support during my studies.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, May 22, 2012 Juraj Figura

Názov práce: Strojové učenie pre Google Android

Autor: Juraj Figura

Katedra: Ústav formální a aplikované lingvistiky

Vedúci bakalárskej práce: RNDr. Ondřej Bojar Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Práca sa venuje téme strojového učenia. Popisuje teoretický základ úlohy klasifikácie a zameriava sa na dva algoritmy–rozhodovacie stromy a naivný Bayesov klasifikátor. Použitím týchto algoritmov sme naimplementovali knižnicu pre platformu Android. Knižnica poskytuje základnú funkcionalitu pre klasifikačnú úlohu a je navrhnutá s dôrazom na jednoduchosť a efektivitu, vzhľadom k tomu, že je určená pre mobilné zariadenia. Funkčnosť knižnice bola otestovaná na rozsiahlom súbore dát a jej presnosť bola porovnateľná s inými implementáciami. Dôležitou časťou práce je aplikácia využívajúca našu knižnicu. Aplikácia zbiera dáta o kultúrnych podujatiach a pomáha užívateľovi v ich filtrovaní podľa osobných preferencií. Keďže dáta sú získavané online zo skutočných serverov, nejedná sa len o jednoduchú ukážku, ale o použiteľnú a potenciálne užitočnú mobilnú aplikáciu.

Kľúčové slová: strojové učenie, úloha klasifikácie, Android

Title: Machine Learning for Google Android

Author: Juraj Figura

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Ondřej Bojar Ph.D., Institute of Formal and Applied Linguistics

Abstract: The thesis discusses the topic of machine learning. It describes the theoretical base of the classification task and focuses on two algorithms–decision trees and Naive Bayes classifier. Using these algorithms we have implemented a library for the Android platform. The library provides the basic functionality for the classification task and it is designed with an emphasis on simplicity and efficiency, given that it is aimed for mobile devices. The functionality of the library has been tested on a large data set and its precision has been comparable to other implementations. An important part of the thesis is an application using our library. The application collects data about culture events and helps the user to filter some of them according to his or her personal preferences. As the data are obtained online from real servers, it is not only a sample demonstration, but a usable and potentially useful mobile application.

Keywords: machine learning, classification task, Android

Contents

1. Introduction

1.1 Motivation

Machine learning is an interesting field in Computer Science. As computers make progress in terms of storage space, computational speed, mobility or networking, there is a need to take advantage of that and produce some useful results. Machine learning can be used when we do not know an exact algorithm to solve a task, but we can provide enough data to learn from (e.g. hand-written text recognition). A lots of algorithms have been developed for machine learning, but still there is a challenge in creating real applications based on them.

We have decided to create a library, which would be easy to use by a programmer with no experience with machine learning algorithms, and which would help his application to behave more intelligently.

We have chosen the Android platform, because we consider mobile devices very perspective and there is a great space for new ideas in this field. Android is the fastest growing mobile platform, millions of devices (mobile phones, tablets, ...) run Android and its success is based on cloud Google applications as GMail or GMaps [7]. Another great feature is that the most of the applications are available at a single place[1], which is natively supported by the platform, anyone can make his application publicly visible there and the users know where they can find it. Moreover, the devices are available for much lower prices than those of the currently biggest Android rival - iOS[2].

Secondly, it is a platform for which the development is supported very well. It uses one of the most common programming languages–Java, and for its SDK there are lots of tutorials and support.

There are many Android applications, lots of them are useful and provide a lot of information, but there is a lack of some intelligent filters or features that enable greater customization. That is exactly a task suitable for machine learning.

The result of our thesis is a library for machine learning and a non-trivial Android application based on the library. Both of them are included on the attached CD (see Appendix A).

1.2 Related work

There are several projects with a similar effort. However, they are not finished yet or they are too complicated to use and their sizes are often too big. The *android-reasoning-util*[3] aims to the sensory data of a device, *ml4android*[4] attempts to use a wide range of algorithms. Android version of Weka[5] is already available, however its size can be a problem.

Instead, we bring out a ready-to-use, small library, together with an interesting application.

[1] https://play.google.com/
[2] http://www.apple.com/ios/
[3] http://code.google.com/p/android-reasoning-util/wiki/Home
[4] http://code.google.com/p/ml4android/
[5] https://github.com/rjmarsan/Weka-for-Android

1.3 The structure of the thesis

In the first part of the thesis, we introduce the theoretical issues of machine learning. We focus on the *classification task*, which predicts a target class from a set of attributes. We define the task of *concept learning*, some basic terms and algorithms. That leads to an important issue of the inductive bias, which simply says, that we cannot classify new instances without having some assumptions while learning.

Then we examine 2 algorithms for the classification task–the Decision tree algorithm and the Naive Bayes classifier. These belong to some of the most frequently used ones and we have implemented them in our library.

The second part discusses the development of the classification library. We describe the representation, the data structures and take a closer look to the implementation of the two algorithms. We introduce a few problems, e.g. the necessity of implementing a custom serialization for decision trees.

The main part of the library is a pure Java code, so it is able to operate on a desktop machine as well. However, we extend that by an Android dependent class which helps to create a graphic user interface (GUI) describing the learning task. Moreover, this extension provides some methods that enable the communication between the GUI and the computing part of the library. These methods make it easy to develop basic Android applications using our machine learning toolkit.

An important part of the thesis is the testing of the library. We used a relatively large data set to evaluate the precision. The results have been very close to the results of one of the most popular machine learning applications–Weka[6] [1]. Here, we also mention the importance of the representation of a classification task for time and space efficiency.

To document the usefulness of our library, we devote the last part of the thesis to the application we have developed using our library. The application is called EventAdviser and it attempts to be a user friendly Android application that helps the user in filtering some of culture events, which may be of his or her interest. The application can download real data from two information servers and learns to sort them according to the user's preferences. It is designed to be easily extensible by plugins that enable more sources of the data to be included. Here, we also demonstrate some principles of using the library.

[6] www.cs.waikato.ac.nz/ml/weka/

2. Theoretical background

In this chapter, we define the basic terms connected with machine learning. We start with the *concept learning*, which is, in fact, a special case of the classification task, but it is simpler. This will lead into one of the most important things about the machine learning - the inductive bias. Then we will discuss two classification task algorithms - the Decision tree algorithm and the Naive Bayes classifier.

The most of this chapter is based on the first 3 chapters of Mitchell (1997) [2].

2.1 Definitions and the concept learning

2.1.1 Machine learning

The goal of machine learning is to improve the performance of a computer program with experience. There are a lot of tasks that can be solved using machine learning, including speech recognition, playing games, automatic driving of a vehicle, diagnosing medical cases and data mining (getting some valuable knowledge from large databases). It is also known by common Internet users, for whom machine learning filters spam mail, shows user-aware advertisements or helps with customized search.

Various algorithms have been invented for machine learning. They use results from the fields of artificial intelligence, probability and statistics, information theory, neurobiology and others. Some of the algorithm categories are Decision trees, Artificial Neural Networks, Bayesian learning and Genetic Algorithms.

In the thesis, we shall talk about *supervised learning*. That means the algorithm has input-output pairs to learn from. If the output is not present, it is called *unsupervised learning*. The last type is *reinforcement learning* where the algorithm is awarded or punished after its (multiple) actions.

2.1.2 Concept learning

A concept learning task is a simple learning task that has much in common with classification task and illustrates the main ideas.

A concept describes a subset of objects in a larger set, so we can think of it as a boolean-valued function over this larger set. The task of concept learning is to approximate this function. For example, we want to find a general definition for a bird over a set of animals.

The set of all items over which the concept is defined is called the set of instances, denoted X. The function is called the target concept, denoted $c : X \mapsto \{0, 1\}$.

Then we have a set of training examples $D = \{x_i\}$, which can be positive ($c(x_i) = 1$) and negative($c(x_i) = 0$). The goal is to estimate c, i.e. create a hypothesis.

The set of all possible hypotheses H is determined by the representation. The easiest way is to have only a conjunction of constraints, which means that a

hypothesis would be a vector of constraints specifying values of attributes. Each attribute can either

- indicate that any value is acceptable
- specify a single required value
- indicate that no value is acceptable

We should find $h \in H$ such that h(x)=c(x) for all $x \in X$. However, we do not have information about all elements of X. The only information we have is about our training examples. Our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data.

2.1.3 Simple algorithms

There are a few simple algorithms for the concept learning task that demonstrate the main problems to deal with and which are solved later by the Decision tree algorithms. Let us define some notions.

A hypothesis h is consistent with a set of training examples if and only if $h(x) = c(x)$ for each example $< x, c(x) >$ in D.

An example x satisfies a hypothesis h when $h(x) = 1$.

A hypothesis covers a positive example when it correctly classifies it as positive.

General to specific ordering of hypotheses: Let h_j and h_k be boolean-valued functions defined over X. Then h_j is more general than or equal to h_k if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

We also define *strictly more general than* relationship similarly. This relation is important because it provides a useful structure over the hypothesis space H for any concept learning problem.

Find-S

This algorithm finds a maximally specific hypothesis. It starts with the most specific hypothesis in H, which is, in the simplest representation mentioned above, conjunction of empty sets. Then it uses more-general-than ordering, takes all positive training examples and for each of their attributes, replaces it by the next more general constraint that is satisfied by x. This algorithm ignores negative examples, it just tries to cover all the positive ones.

The algorithm is based on two main assumptions:

1. The correct target concept is in H.
2. There are no errors in the training data.

If these are met, the output hypothesis will be consistent with all training examples, including the negative ones, which follows from the definition of more-general-than ordering.

The algorithm cannot even detect inconsistency in training examples, because it ignores the negative ones.

We do not know whether we should prefer the most specific hypothesis to more general ones. Moreover, there can be more maximally specific hypotheses.

Candidate-Elimination

The version space, with respect to hypothesis space H and training examples D is, by definition, the subset of H consistent with the training examples in D.

Candidate-Elimination algorithm starts with all H and then removes any hypothesis inconsistent with any training example. Since it is unreal to enumerate all H, it uses a representation of Version space that consists of its most general and most specific members. Then, as needed, it is possible to enumerate all members between these to sets in the general-to-specific ordering. Thanks to this ordering, during the algorithm, just the general and specific boundaries are changing.

With the same assumptions as in Find-S, this algorithm will converge to the correct hypothesis. Ideally, S and G boundaries are one identical hypothesis at the end. Otherwise, we can use the partially learned concept to classify new instances with some degree of confidence. For example, a new instance can be classified positive by all hypotheses in the current Version space. This happens if and only if the instance satisfies every member of S, because every other hypothesis in current Version space is at least as general. Similarly for all negative classification using all members of G. In other cases, the new instance is somewhere between the boundaries and we can compute the degree of confidence.

Having the training data with errors, the algorithm removes the correct target concept from Version space, but with sufficient additional data, we detect an inconsistency, because the final Version space would be empty.

2.1.4 Inductive bias

The hypotheses space is determined by the representation of hypotheses. Using conjunction of constraints, we cannot describe even a simple disjunction. That means, the correct target concept does not have to be in H. In this case, for example Candidate-Elimination returns an empty set.

To assure this will not happen, we could use a representation, where a hypothesis is any subset of X. However, it is not very reasonable, because the power set of X is too large. Moreover, generalization cannot be applied, because the S boundary will always be the disjunction of observed positive examples and G will be the negated disjunction of the observed negative examples. Therefore, we can classify only previously observed examples. Each unseen instance will be classified half positive half negative, because in power set, for each hypothesis that classifies x positive, there will be identical hypothesis except for its classification of x.

The space of all possible hypotheses (power set of X) is called unbiased. When we have an assumption about the hypotheses, the space is smaller and it is called biased.

A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances. In-

ductive bias of an algorithm is a set of assumptions B, such that for a new instances x_i, their classification by the algorithms follows deductively from B and D and x_i, where D is a set of training examples. For example, inductive bias of Candidate-Elimination algorithm is that the target concept is contained in the given hypothesis space H.

2.1.5 Remarks

Find-S and Candidate-Elimination algorithms are iterative, because they just iterate over the set of training examples, which means we can add new examples without starting again. This is not true for Decision tree algorithms.

2.2 Decision tree learning

Decision tree learning is one of the most used methods for the classification task. It is robust to noisy data, it is able to represent and learn disjunctive expressions. Another advantage is that these algorithms are white boxes, which means it is approximately clear, how they work.

2.2.1 The principles

In the classification task, unlike in the concept learning, the target function can have more than two possible output values. The task is to classify examples into one of a discrete set of possible values.

Input: Training examples - a subset of X, where X is the set of all possible instances. An instance is represented by attribute-value pairs (extensions allow real-valued attributes) and the label (the output value of the target function). For each attribute we know its possible values and so do we for the label.

Output: Decision tree. Using decision tree, we can easily classify unseen instances. Each node of the tree represents a test of some attribute of the instance. Each branch from that node corresponds to one of the possible values for this attribute. An instance is classified by starting in the root node, testing the attribute specified by this node, moving down the branch corresponding to the value of the example and this is recursively repeated until it reaches a leaf node, which provides the classification.

We can divide the algorithm into 2 main parts:

1. Creating the tree.
2. Post-pruning the tree, which deals with noisy data.

2.2.2 Creating the tree

The tree is constructed top-down. It finds such attribute that is the best to test as the first, in sense, how well it can distinguish the training examples. It is selected as the root node and for each its possible value, its descendent is created recursively with appropriate examples (whose values correspond to the branch).

2.2.3 Choosing the best attribute

We will define a statistical property, called information gain, that measures how well a given attribute separates the training examples according to their target classification.

At first, let us define the Entropy, that characterizes the purity of the examples.

$$Entropy(S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

This is a definition useful for positive/negative classification. Only if the entropy is 0, all members belong to the same class. The entropy is 1 if the collection contains an equal number of positive and negative examples. For unequal numbers, it is between 0 and 1.

One interpretation says that *Entropy(S)* specifies the minimum number of bits of information needed to encode the classification of an arbitrary member S. For example, if all are positive, no message is needed, and entropy is 0. If the numbers of positive and negative is equal, 1 bit is required for each classification. Finally, if positive (or negative) ones are more frequent, in average less than 1 bit is needed, using shorter codes to the collections of those more frequent.

In general, if the output can have c different values, entropy is defined as follows.

$$Entropy(S) \equiv \sum_{i=i}^{c} p_i log_2 p_i$$

The logarithm is still base 2, because we are interested in expected encoding length in bits. The maximum of the entropy for c-valued target function is $log_2 c$.

What we want is to know the efficiency of splitting examples with respect to the attribute. Information gain measures the expected reduction of entropy caused by the split.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)}^{k} \frac{|S_v|}{|S|} Entropy(S_v)$$

As the best attribute, we select the one with highest information gain, so, in fact, we search for the minimal sum over values of A.

2.2.4 Inductive bias in decision tree learning

Although there are many decision trees consistent with the examples, the algorithm chooses one of them, the first acceptable it finds. In its search, it prefers shorter trees and selects the trees that have the attribute with highest information gain in root node.

As we mentioned, there has to be some inductive bias if we want to classify unseen instances. The inductive bias of decision trees reminds of Occam's razor, which says:

> "Prefer the simplest hypothesis that fits the data."

More precisely, the inductive bias of decision tree algorithms is: "Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not."

2.2.5 Post-pruning the tree

Post-pruning is a method used to avoid so called overfitting the data. We say that a hypothesis overfits the training examples if there is some other hypothesis that fits the training data less well, but performs better over the entire distribution of instances. This is caused either by errors in the training data, or because the training set is too small.

The most common approach is to divide the training set into two parts. Two thirds serve as the training examples and the rest as a validation set, used for the post-pruning. We can remove a subtree and make it a leaf node assigning it the most common classification of the training examples affiliated with the root of the subtree. We remove a subtree only in case that the resulting pruned tree performs no worse than the original over the validation set.

2.3 Naive Bayes classifier

The Bayesian learning methods are based on a probabilistic approach.

Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct, unlike algorithms that completely eliminate a hypothesis if it is inconsistent with any seen example.

New instances can be classified with a certain amount of probability, such as "there is 90% chance that the event is interesting for this user".

Naive Bayes classifier is a very popular method and its performance has been shown to be comparable to decision trees and neural networks.

2.3.1 The algorithm

We assume a classification task as we have defined before. The label can take on values from some finite set V.

The approach to classifying an instance a is to assign the most probable target value v_{MAP}, given the attribute values $\{a_1, ..., a_n\}$ that describe the instance.

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j | a_1, ...a_n)$$

We use the standard notation $P(x|y)$ which means the probability of x assuming y. Now, using the Bayes theorem, we can rewrite the expression as

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, ...a_n|v_j)P(v_j)}{P(a_1, ...a_n)} = \arg\max_{v_j \in V} P(a_1, ...a_n|v_j)P(v_j)$$

It is easy to estimate $P(v_j)$, however, it is too difficult to estimate $P(a_1, ...a_n|v_j)$ without any assumption. The main idea of Naive Bayes is to assume that the attribute values are conditionally independent given the target value. That means,

the probability of this conjunction is just the product of the probabilities for the individual attributes. This leads to the final equation

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{i=1} P(a_i|v_j)$$

where v_{NB} is the target value output of the algorithm.

Furthermore, we can return the probability of this result. Let us say we have calculated the term for each v_j in V as $f(v_j)$. The most probable is v_k. Then the normalized probability is $\frac{f(v_k)}{\sum_i f(v_i)}$.

2.3.2 Estimating probabilities

There is a difficulty when estimating the probabilities as the simple fractional count (n_c/n). We get poor estimates when the number of examples with a certain value of some attribute is very small. If a probability of an attribute's value is very small, it is very probable that we have 0 training examples with that value and our estimate is 0. This causes underestimation and moreover, it affects other terms because we use multiplication in our equation.

To avoid this, we use the m-estimate:

$$\frac{n_c + mp}{n + m}$$

Where p is our prior estimate of the probability we wish to determine and m m is a constant called the equivalent sample size. Typically, we choose p to be the same for all possible values of the attribute. We can interpret this estimation as extending the n actual observations by an additional m virtual samples distributed according to p.

3. The library

This chapter provides information about the library, we have implemented. It describes our considerations about the representation, the most important issues of implementation of the algorithms and briefly talks about how the library can be used.

3.1 Representing the data

The library shall be implemented in the Java programming language, since it is the only one directly supported by the Android platform.

3.1.1 Representing the task

The first thing to consider is the representation of a classification task. It is defined by a set of attributes, where each attribute has its possible values. There are algorithms in machine learning that can use real-valued attributes, but the ones we have chosen don't support them directly. Therefore, we will operate with discrete-valued attributes. The user of our library can divide a real-valued attribute into some intervals on his own. The second thing we leave to the library user is the data types of the attributes. In a classification task, the values can be strings, various numbers, etc. As these are discrete, we have decided to use a common format of the values - each attribute has its values represented as integers from 0 to $m_i - 1$, where m_i is the number of the values (we shall call it *range*). Note however, that the ordering is arbitrary and has no impact on the classification. The conversion is made by the user, or it is made by the GUIHelper class from our library, which offers a comfortable way to declare the classification task in XML format.

The advantage of the integer representation is clear - the efficiency. A question we dealt with was if it is possible to use smaller data types, e.g. bytes, as integers could be wasteful if only a few values are needed. There are generic types in Java, however they cannot be applied on the primitive types. We could use wrapper types for numbers when declaring a generic class, but it would not bring any saving, because a wrapper type is a class and therefore needs 4B reference, which is the size of the primitive integer type.

Our first approach was to represent the task by the ranges and the index of the target attribute. The motivation was a task described by some attributes, where one of them is the one we are to predict. However, especially for the implementation of the Naive Bayes classifier, it is more convenient to have the target attribute separated.

In conclusion, the main class that represents the task is called *Classifier*. It is initialized by an integer array describing ranges of the attributes and an integer representing the range of the target attribute (which is not included in previous ranges). This class provides the methods for adding examples and classifying new instances. These are represented as integer arrays of values corresponding to the ranges given in the initialization.

Inside the class, we use *Example* class containing the values and the label.

3.1.2 Representing the decision tree

We represent the tree by abstract *Node* classes, which can be an *InnerNode* containing the attribute number and an *ArrayList* of *Node*, or a *LeafNode* containing the label. As the ArrayList is indexed directly by the attribute values (due to the representation), the classification can be very fast.

Since we need the training data and the learnt models to persist across sessions, we consider a few options. For the training examples we could use a database approach (SQLite is directly supported by the Android platform). However, there would be difficulties in saving the decision tree. We have thus opted for the serialization of the *Classifier* class, which includes the learnt Naive Bayes data and the decision tree. An application that uses the library should serialize the *Classifier* object, which costs about 3 lines of code in Java, and save it on the file system. When making the class serializable, we use standard Java serialization mostly. The only thing the standard serialization cannot manage is the tree. Therefore, we override it in the Node class.

3.2 Implementing the algorithms

Here, we shall describe the standard ID3 algorithm (Quinlan 1986) [5] and Naive Bayes with a few implementation details we used.

3.2.1 Decision tree - ID3

The algorithm ID3 recursively computes a (sub)tree of the decision tree. Its parameters are:

- ArrayList< *Example* > examples–a list of examples relevant to the current subtree, i.e. examples from the training set, which satisfied all values of attributes tested during the descent in the tree to the current subtrees's root. The main call of ID3 includes the whole training set.
- ArrayList< *Integer* > attributeIndexes–a list of indexes of the attributes, which have not been tested yet. At the start it is all attributes.

The algorithm:

1. If all examples are with the same label, return a new leaf node with this label.
2. If attributeIndexes is empty (i.e. no more attributes to be tested), return a leaf node with the most common label among the current examples.
3. Choose the attribute, which best classifies the example (see Section 2.2.3).
4. Create a root node of the subtree labeled with the chosen attribute and create branches for all its possible values.
5. Create S_vSet according to the current examples and the attribute a, which is a list of lists of examples. The v-th member of the S_vSet is a list of examples, all of which have v as the value of the attribute a.

6. For each member of the S_vSet (i.e. for each possible value of the attribute a)
    - run ID3 recursively with examples of the S_vSet for v-value and cloned attributeIndexes with a removed. Assign the output of this recursive call to the corresponding branch.
7. Return root.

We have not used any post-pruning method in our implementation.

3.2.2 Naive Bayes learning

The learning phase computes the probabilities, which are used in the classification. They are stored in Classifier fields:

- *probabilities*–a 3 dimensional array of double. The first dimension is the attribute number, the second is its value and the third is the label value.

$$probabilities[i][j][k] = P(attr_i = val_j | label = label_k)$$

- *labelProbabilities*–an array of double, indexed by the label value. $labelProbabilities[2]$ contains the probability of an example having its label equal 2.

The algorithm:

1. let *subsets* be the set of subsets of the examples, where examples in a subset have the same label v for each possible label.
2. initialize the probability arrays according to number of attribute and ranges.
3. for each label value v:
    (a) $labelProbabilities[v] = subset.size/totalExamplesCount$
    (b) Count the occurrences of each possible attribute value. This is done proportionally to all attributes of all training example. We have counters for each attribute value represented by the two dimensional array
    $$counts[attrIndex][attrVal]$$
    (c) Count the probabilities using the counters and adopting the m-estimate with uniform prior probabilities and m equal to the number of the attributes.

This algorithm works in

$$O(LabelRange.AllPossibleValues + ExamplesCount.NumberOfAttributes)$$

Here, the classifying is not so straightforward as it is in the decision tree. As we explained in 2.2.3, the most probable label of a new instance is acquired as $argmax$ from some multiplication of probabilities. However, it can lead into floating point underflow easily, as all the probabilities are between 0 and 1 and the multiplication iterates over all attributes. Therefore we compute the logarithm of the multiplication, which is the sum of the individual logarithms. The argument of the highest sum is the most probable classification, because the logarithm function is monotonic. This modification is described in online version of [6].

3.3 The GUIHelper

What we have introduced so far, has been a pure Java part of our code. All the code is fully portable to any desktop with the Java Virtual Machine.

A part of the library we also developed is a helper class for creating a simple GUI for Android. The idea is to have a helper class that enables to create a simple classifying application quickly and that is flexible enough to be used in more advanced applications.

The GUIHelper class can create combo boxes (called spinners in the Android GUI) that represent the attributes of a classification task. Moreover, they provide methods to retrieve the data prepared to be used by the Classifier class (integer representation of the selected values).

The GUIHelper does not communicate directly with the Classifier. That enables the user of the library to extend the attributes from the GUI with some custom attributes which could be retrieved from the Internet, the device resources or a custom GUI.

The input of the class is a simple XML document that describes the task. It can look like this:

```
<task>
        <attribute name="genre">
                <value>Jazz</value>
                <value>Rock</value>
                <value>Pop</value>
                 ...
        </attribute>
        <attribute name="company">
                <value>Alone</value>
                <value>With friends</value>
                <value>With family</value>
        </attribute>
        ...
        <attribute name="interesting" isTarget="true">
                <value>Yes</value>
                <value>Medium</value>
                <value>No</value>
        </attribute>
</task>
```

The outputs are: A GUI component with spinners describing the input attributes. The target spinner can be retrieved separately, so the application may put some GUI between the input spinners and the target spinner. Alternatively, the application can implement the classification GUI by some buttons instead of the target spinner.

To create a Classifier instance, we need the attribute ranges. The GUIHelper's method returns this array that can be directly passed to the Classifiers's constructor. When we want to use custom attributes, (not defined in the XML file), we prepare an additional array and pass the concatenation of the two arrays to the constructor.

Similarly, when adding a new example, there's a method in GUIHelper that returns the selected values prepared for the Classifier (i.e. an array of integers).

3.4 Using the library in other projects

A programmer can use either the Java base of the library or the full Android version. The attached CD (see Appendix A) includes a *jar* file, which can be imported to any Java project and provides all the functionality of the Classifier. The *androidClassificationLib* uses this *jar* and adds the *AndroidGuiHelper* class.

This approach enables a programmer to create a system consisting of two parts–an Android application (mainly to collect the data and show classifications) and a desktop application (mainly to learn the model). These parts shall be compatibile.

Using Eclipse[1] and Android ADT 17 plugin[2] or higher, a recommended (and standard) way to use the full library is the following:

- In your application project, create a folder called *libs* and add the *androidClassificationLib* there.
- Import *androidClassificationLib* from *libs* to your workspace.
- In the application project properties, select Android and add a reference to *androidClassificationLib*.

If you use an XML file to describe the task, insert it into the *assets* folder of your application project.

The summary of the main methods is in Appendix B.

[1] http://www.eclipse.org/

[2] http://developer.android.com/sdk/eclipse-adt.html

4. Testing of the library

This chapter describes the evaluation method we have used, our testing task and the results we have reached.

4.1 Evaluation of trained models

The library provides a complex method for evaluating given a collection of labeled examples. Its parameters are the evaluation set[1] size (a portion of the whole dataset) and the sizes of training sets to be tested.

The evaluation set is fixed (randomly selected collection of examples). Then, for each training set size, a training set disjoint from the evaluation set is randomly selected from the examples. The both learning algorithms are called for each of these. As the evaluating set is in fact a part of the labeled examples, we can measure the success. This procedure is repeated r-times (r can be set by a parameter, too) and the average success is returned.

While performing that, the learning and classifying times are measured. The times are measured and returned in milliseconds. The classifying time is the average time spent for classifying whole evaluating set as classifying a single instance is typically extremely short.

Note that the real performance in terms of classification accuracy can be better as it uses the whole training set.

4.2 The testing task

We have tested the library's performance on the following problem: We have the data from a linguistic project CzEng [4]. The data contain parallel sentences in English and Czech. The information which word of a sentence is mapped (translated) into which word is included. Now, we are only interested in these word pairs. Each word is described by a set of some linguistic attributes. In our task, we need a few of them - the formeme of the English word, the formeme of the father word of the English one and the Czech formeme. The task is to predict the third one from the first two mentioned. We can see the most common examples from the data in table 4.1. The linguistic meaning of the attributes is not important for our experiment, so we do not provide any detailed explanation.

A simple representation therefore is: A training example is a word pair. It has two input attributes, $myFormeme$, $fatherFormeme$, both of which can have values from a set called $engformemes$. The label is the $czFormeme$ that can have values from the set $czFormemes$. These sets are not identical, but they have some common values and both are relatively large. We chose just a part from the full CzEng data (they are already shuffled and thus distributed uniformly) which lead to 100 000 examples, 431 possible values for $engformeme$ and 288 values for $czformemes$. That means 288 possible labels.

Already from the representation, we can see that the decision tree will be very wide (many possible values for the attributes) and its depth will be small (only

[1] It is not a set, because an element can be included more times

myFormeme	fatherFormeme	czFormeme
v:fin	no father	v:fin
n:subj	v:fin	n:1
n:	no father	n:1
n:obj	v:fin	n:4
v:fin	x	v:fin
n:attr	n:attr	n:???
n:obj	v:fin	n:1
adj:attr	n:obj	adj:attr
n:subj	v:rc	n:1
n:attr	n:	n:???

Table 4.1: The most common examples from the CzEng data.

two attributes, max. depth is two).

4.3 The testing procedure

We would like to see the progress while enlarging the training examples. However, the step of choosing the mentioned 100 000 examples is very important. If we started with a smaller part (and then enlarge it), it would mean a different task, because the vocabularies would be different and the smaller task would be easier. Therefore we could not compare the results.

Having defined the task firmly, we can use the Evaluate method properly. We called the Evaluate method with various training set sizes and a fixed evaluation set size of 30% of the 100 000 examples.

4.4 The results

The Naive Bayes learning is faster, but the classifying is slower than it is in the decision tree method. It's clear when talking about the classifying, because when the decision tree is already constructed, we just need to go down through it. Choosing the branch is very fast, because they can be directly indexed by the attribute value. Moreover, the tree is never deep in this representation. On the other hand, the Naive Bayes classifier has to count the probability for each possible label (and even the logarithms). One of the reasons for the slower Decisions tree's learning performance is the memory allocation which is not performed once, but each time when creating an inner node.

The performance of both methods has reached 52%. As the number of labels is 288, a random prediction would succeed just in 0.34%. If the predictions were fixed to the most frequent label (known as Zero classifier method), it would succeed in 18% of cases. The results are summarized in tables 4.2 and 4.3.

We've tried to compare the results to the results from the Weka application using its implementation of the Naive Bayes classifier and the ID3 algorithm. The success hasn't reached more than 51%. The learning times were similar to ours,

Number of training examples	Bayes success in %	Decision tree success in %	Bayes learn time in ms	Decision tree learn time in ms
50	35.68	36.61	9	19
100	40.87	40.93	11	22
500	45.52	45.87	7	103
1 000	47.07	47.42	11	198
5 000	49.89	50.04	10	890
10 000	50.54	50.89	12	1662
20 000	50.87	51.51	30	3076
40 000	51.26	52.03	22	5417
60 000	51.51	52.21	28	7191

Table 4.2: The results of the testing of the library on linguistic data. The procedure has been performed on a desktop and repeated 10-times.

Bayes classification time (30 000 examples) in ms	2531
Decision tree classification time (30 000 examples) in ms	9

Table 4.3: The average classification times in the test.

but slightly better. The success rates with less training data were a little worse when using Weka.

4.5 An alternative representation

Apart from assessing the library's performance, we shall demonstrate the importance of task representation. Let us consider another representation of the task. Instead of two input attributes with larger ranges, we will use many attributes with small ranges. For each English formeme, there will be an attribute with the possible values as follows:

1. neither the word nor its parent have the formeme set to the given value
2. the word (and not its parent) has this formeme
3. the parent (and not the word itself) has this formeme
4. both the word and its parent have this formeme

The label could stay the same as in the previous representation, or we could add a special value that says "it is the same as in the English word". With this representation, we see that the algorithms run much slower.

The worst of all is the classifying time in the Naive Bayes method. The learning phase is "only" 2 times slower, but the classifying is about 200 times slower with our task size. We see it in the theory already: The learning in the Naive Bayes counts the probabilities for each possible value of each attribute for each label. That is $|engFormemes| \cdot 4 \cdot |czFormemes|$ ($|engFormemes|$ attributes, 4

possible values for each, $|czFormemes|$ possible labels), while in the previous representation it is $2 \cdot |engFormemes| \cdot |czFormemes|$ (2 attributes, $|engFormemes|$ possible values for each, $|czFormemes|$ possible labels).

However, the classifying iterates through each attribute for each label value. That is $|czFormemes| \cdot |engFormemes| = 125000$, while in the previous representation, it is $|czFormemes| \cdot 2 = 576$. Note, that it is the same task, just represented alternatively.

Finally, if we want to use the information about the agreement of myformeme and fatherformeme, we can do it effectively by having 3 attributes

- $isSame \in 0, 1$
- $myformeme \in engformemes$
- $fatherformeme \in engformemes$
- $label \in czformeme \cup SAME\_AS\_ENGLISH$

The times and successes of this representation were close to the first approach. Note, that in a real testing it could be slightly better, considering the case when the prediction is a correct, but concrete formeme, but not labeled as SAME_AS_ENGLISH. The default Evaluate method doesn't count that as a success.

5. The application

Here, we examine the application that we have created using our library. The chapter starts with motivation, then describes the architecture of the application and focuses on the most important issues of the implementation. Finally, it explains how the machine learning is used and shows some patterns for using our library in non-trivial cases.

5.1 Introduction

The main aim of our sample application built using our machine learning library is to provide information about culture events that could be interesting for the user.

There are many web servers in Czech Republic which contain huge amounts of data about culture events. However, they typically don't provide any advanced filters. The first task of our application is to bring the data into the mobile device and serve them in an appropriate easy-viewable format instead of using a mobile web browser, which is not very convenient. As the users don't always have the Internet connection, having the data offline can be very useful. However, the most important feature is the intelligent filter based on the machine learning, which sorts the events by their importance for the user.

The user documentation can be found in Appendix C.

5.1.1 Advantages and problems

Typical situations solved by the application

- I feel like attending a culture event in the moment, but I can't find anything suitable, because I get lost in the multitude of the data.
- I miss something what could be interesting for me, because I don't have the energy to search for the events regularly.

Summary of the main features

- comfortable access to the data in a mobile phone
- having the data offline, no need to have the connection
- effective filtering and sorting events customized to the user
- taking current conditions into account (location, occupation, etc.)
- grouping the data from various servers together
- simple GUI

The problems

Often, one of the most important factors in making the decision whether to attend an event is the price. However, this is rarely a part of the data provided on the most servers. Usually, it is quite hard to find this information and you have to search the proper venue's website or even further to a ticket portal.

The application's approach is to filter as well as possible and then to provide links to the event's details.

5.2 Source data and servers

We asked several Czech information servers for an access to their data. The *informuji.cz* server allowed us to use the data for their partners, which is periodically updated on a certain URL. Then we added the data from the *last.fm* server that provides full API to various data (albums, interprets, events), but for our purpose it is sufficient to get the data about the concerts.

Both servers allow to use a HTTP request to gain the actual data in XML. The *informuji.cz* server events for the Czech Republic. *last.fm* is a worldwide server, so in the URL we can specify some attributes. We use the data for Prague, but we possibly could add a parameter which would specify the diameter (in kilometers) how far an event can be be accepted, too. We could also use another city, but it would need a change in the front-end.

The main idea is to have a class that represents the data about an event and is independent from the source servers. We have an abstract class called *Event*. Each server will implement its own descendant of the Event class and therefore provide public methods like *getName*, *getShortDescription*, *getLongDescription*, *getPlace*, etc. Moreover, the specific *ServerClassifiers* can take the advantage of having their specific Event descendants and use some extra data in the classification task.

For instance, *informuji.cz* usually provides longer text descriptions, which is very uncommon for the *last.fm*. Instead, *last.fm* has a data node for the list of the performing artists. So, the output of *getDescription* is generated from the first 300 characters of the text description in the *informuji.cz* case and from the list of the artists and tags (keywords) in the *last.fm* case. Due to this, the logic is separated from the specific data parsing.

We wanted the application to be able to load the data offline from a file. It was intended mostly for the testing and it wouldn't be very convenient for a common user as they would have to get the files somehow. Therefore we have revoked this functionality.

Possibly, the release version could perform the updates periodically in the background (it is already implemented asynchronously) and the user would not have to do anything.

5.2.1 Loading the data

The data about the available events are stored in a Map indexed by the event *Id* ant the values are instances of the Event class. The Id is a string created this way: $< server\_name\_code >< internal\_event\_id\_on\_the\_server >$. It should

prevent from the Id collisions when having multiple servers events Ids stored in the front-end's structures.

The application is able to update the data online. We use simple HTTP requests to URLs with the XML data. Then we can apply some helper methods for getting an instance of Document, which is a Java class that represents an XML file and provides some parsing methods.

The specialized classes *LfmEvent* and *InformujiEvent* can parse a Document instance in their constructors.

Both data downloading and XML parsing can take a longer time (a few seconds), therefore it is reasonable to run them asynchronously. That is nicely supported by the Android class *AsyncTask*, which also cooperates with the GUI thread to pass the progress messages. The user can keep using the application while updating the data.

5.3 The main design concepts

The whole application's front-end is managed by a single Activity. The layout consists of a Pager, to which we define an Adapter that describes how to display an event according to the current mode.

The activity has an instance of the EventToClassifierAdapter class (referred to as adapter) that provides methods for getting the data about the events and working with the machine learning (i.e. addExample, classify,...).

As we support multiple servers, the task of the adapter is to unite them. The activity works with the events using the abstract Event class that guarantees the specific server's event to have methods like *getShortDescription*, *getName*, etc. Similarly, there is a separate Classifier instance for each server.

The adapter stores a structure of all the specific classifiers. The classifier for a server is contained in a class that inherits from the *ServerClassifier* class, which is also responsible for providing and updating the data about the events.

Therefore, the front-end activity has a complete support for filtering the events.

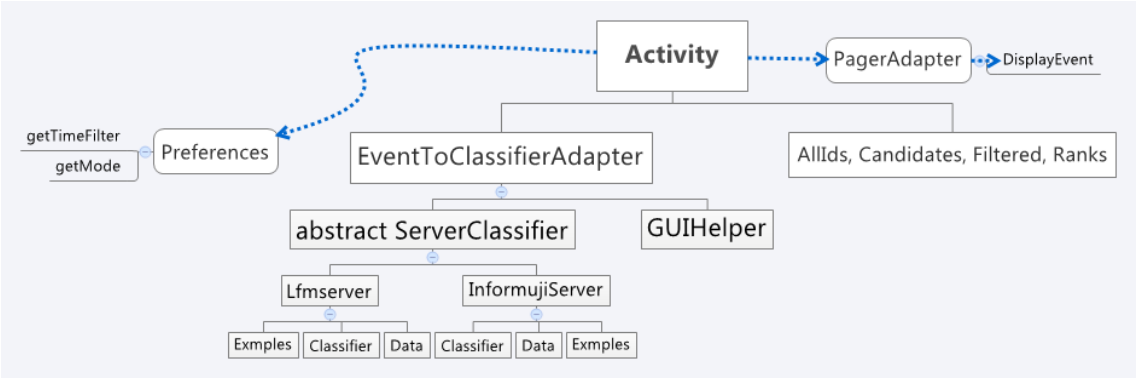


Figure 5.1: The main design concepts of the application. Separating the servers.

For instance, let us show how the *addExample* call is propagated. The user clicks the Yes/Medium/No button. The *ButtonListener* in the activity sets the GUIHelper's target spinner to the selected value. A new instance of Example is

created–we have the Id of the Event and the conditions values. This instance is passed to the adapter that chooses the right *ServerClassifier* and calls its *addExample*.

When learning, each *ServerClassifier* specifies the classification task according to the collected examples and the available data, transforms the examples into integer values and passes them to the "low level" *addExample* of the Classifier.

5.3.1 Separating the servers–advantages and disadvantages

Due to the separation of the servers, we have the application easily extensible. One can just implement a class that inherits from the *ServerClassifier*. Only a few (abstract) methods must be implemented and typically can follow the patterns we used in the 2 default servers. The methods include updating the data and preparing the classification task.

The second advantage is that a unification of various source servers could be unefficient in case of the classification. For example, the *informuji.cz* server provides various types of events (e.g. cinema, theater, music, exposition), which is an important attribute for the classification of these events. However, there is no way to map the *last.fm* examples to this, as they all are about music (which is more precisely described by the tags). If we set this attribute to "music" for all *last.fm* examples, it can lead to wrong classifications. It happens for example when we collect more negative examples from the *last.fm*. In such case, it appears to the Classifier as the value "music" from the attribute "type" indicated the example is not interesting.

When we tested the unifying approach, the problem proved to be serious.

On the other hand, the separation of servers brings a little disadvantage, too. We lose portions of information common for all servers. At least, it is the values of the conditions. There can be some conditions indicating the user's decision, but our approach splits the information into the individual classifiers. However, if there is such a rule in the user's decisions, the problem will be solved by adding more examples.

5.3.2 Event importance ranking

For showing the actual events ordered by their importance to the user, we define a ranking system. It is based on the results of the two available classifying methods–Decision trees (DC) and the Naive Bayes classifier (NB).

Each event that satisfies the time filter is classified by both of the methods. Then, the rank is computed as follows:

if $bothmethodssayNo$ **then**
  return
else
  if $DCsaysYes$ **then**
    $rank \leftarrow 100$
  end if
  if $DCsaysMedium$ **then**
    $rank \leftarrow 50$
  end if

$rank \leftarrow rank + 100 * P(Yes)byNB$
$rank \leftarrow rank + 50 * P(Medium)byNB$
end if

The main advantage is that we get a larger scale than just Yes, Medium or No. Mainly due to the probabilities of the NB we can see the slight differences between the importance of events. The training phase is unaffected and we still ask the user to select from three simple options.

5.3.3 Being user friendly

One of the main approaches of the application is to be accessible for ordinary people. That is partially done by the rank of the events, because it hides the machine learning background.

The second task is to develop a user friendly GUI. The main feature is the pager that enables the easiest way to switch events.

The pager class is a class from the Android API and provides the functionality of viewing some data in pages that can be switched by dragging them horizontally. This pattern is used for instance in the Android Market application. Note, that for a horizontal scrolling, the Gallery widget can be used too. However, the Gallery is intended mainly for viewing pictures and there have been several problems in implementing the GUI this way.

It is important to realize that the application is to be run on a mobile device where a click is more difficult for a user than on a desktop. Therefore we try to minimize the clicks needed for an operation. For instance, as there are just 3 possible labels in the classification, we don't use the library's target spinner. Instead of clicking on the spinner, choosing the value and submitting, the user can click just once on one of the 3 buttons (Yes, Medium, No). This action adds the example and switches to the next event.

5.3.4 The pager and front-end's structures

The whole application's GUI, except for the dialogs, is wrapped in the Pager. We have to implement a class that extends the PagerAdapter class. Its most important method is *initLayout*.

The *initLayout* method gets the position of the pager and should return a *View* to be displayed when the user scrolls to the position. Therefore, we must have a mapping from the positions to the events, which depends on the current mode. For each mode, we have a list of event Ids that defines the mapping directly–we index the list by the pager position. When we have the event, we can easily prepare the layout.

Let us examine how these ArrayLists are constructed.

- allIds–should contain all events that can be shown in the training mode. After a startup and updating, we set the list a as random permutation of all the available data.

- candidates–after a change of the time filter (or change of allIds), we refresh this list to contain such Ids from allIds that satisfy the time filter.

- advisedEvents–after changing the candidates list, we apply the machine learning classification for each member of candidates (as an Example with current conditions), use the ranking algorithm, set the helper Map with the ranks and create the advisedEvents list that contains the event Ids with the $rank > 0$ sorted from the best one.

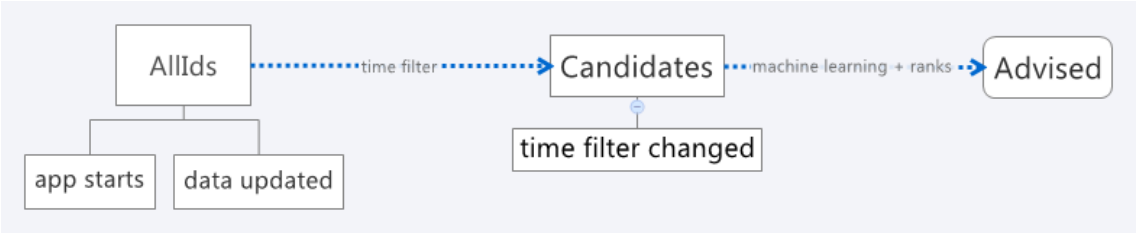


Figure 5.2: Three ArrayLists for storing culture event Ids for each mode and the events that cause refreshing the ArrayLists.

In the classifying mode, we show the events from the candidates along with their ranks (not sorted), i.e. the user can see also the actual events with the rank 0 to retrain the classifier.

5.3.5 Using the application preferences

For saving the data connected with the classification, we use the serialization of the EventToClassifierAdapter object which includes the examples, learned model and the data about the events. However, there are some more data to preserve, for example the current mode, time filter and the flag which says if the application is running for the first time.

Such data are recommended to be represented as *application preferences*. There is a mechanism for working with application preferences in Android API, which enables us to save key-value pairs which persist over multiple user sessions. We encapsulate this key-value approach in our Preferences class with custom methods like *getMode* or *setMode*.

5.4 Developing an adapter for the Classifier

When developing a little more advanced application that needs to use the Classification library, we recommend to create an adapter class for the Classifier.

It should separate the application logic from the Classifier usage mechanisms. For instance, the input (and the output, too) of the Classifier class is represented as integers, which is good as it is universal and efficient, However, the adapter class should receive some custom data (e.g. keywords from an event). We shall describe some of the patterns that are used in the EventAdviser application.

5.4.1 Principles

Each time the data is updated, the classification task changes. It is because the new available events may contain new values for some attributes (e.g. a new city)

that lead to a change in the attribute ranges, or new attributes can be added (e.g. a new keyword). As we usually have some examples collected during an update, and there is not a way how to re-create them for the new task from the old one (explained later), we store the examples separately from the machine learner. That means we have a representation of an example and the classifier adapter (in its *ServerClassifier*) contains a list of Examples.

Example

An instance of Example is created each time the user chooses Yes, Medium or No for an event (being in certain conditions). Duplicities are allowed. The Example class consists of 3 parts:

- conditions–we gain this easily from the GUIHelper
- eventId–we store just the Id for the classified event. The advantage is that we get all the attributes for the classification later, when learning and they can be different when learning after uploading the data, i.e. changing the task.

  We just remember which event has been classified and then we get the information that is needed. For instance, a word from the tags might not be considered as the classification attribute if it occurs only in one event, but later, after an update, it can be important as a keyword.
- the label (interesting – yes, medium or no).

5.4.2 Preparing the classifier

When the classification is needed after a data update, we should re-create the Classifier instance and prepare some data structures for the newly formulated task.

Let us precisely define, what the classification task is in the case of our source servers.

- The possible labels are Yes, Medium and No.
- The attributes are:
  1. conditions–describing the user's conditions when he added the example. We have the array of integer values already prepared in the Example instance that received it from the GUIHelper.
  2. custom values
     (a) event type (only for the *informuji.cz* case)
     (b) clubId
     (c) city

     When preparing the Classifier, we have to determine all the possible values for these attributes. Therefore, we iterate through all the available data and construct enums for each. Actually, we construct Maps with the attribute values (strings) as keys and an increasing integer

representation for the Classifier as values. When creating a new Classifier instance we simply set the corresponding ranges to the size of the Map. When adding the examples to the Classifier, we use the Maps. For instance, to add an example of an event in Prague, we find Prague in the Map for the cities (in constant time).

3. keywords (only for the *last.fm* case). These are boolean attributes that just specify if a keyword is present. We consider the tags given by the *last.fm* API (notice, that we can use the specific data). We count their occurrences in all the available data. The keywords considered as classification attributes will be the ones that satisfy both of the following:
   (a) they are contained in the training examples (otherwise, we could not learn anything from them)
   (b) they appear in the available data more than 5 times (we can classify more instances using it)

Possible improvements

We could search for the keywords in the title and/or description of the event, even in the *informuji.cz* case (no tags, just a longer text), assuming we have a set of keywords prepared somehow. Possibly, the *ServerClassifier* could cooperate when choosing the keywords, which would require several design changes. However, we suppose these two servers not to have much in common.

6. Conclusion

In the thesis, we have introduced the classification task and examined two algorithms–decision trees and naive Bayes classifier. There are many more algorithms for machine learning, for example neural networks, genetic algorithms or support vector machines. None of them is the best for any task and data.

We have implemented a machine learning library for the Android platform that provides basic and required functionalities–defining the task, adding training examples, learning a model, classifying new instance and evaluation of the model. We have tested and documented its performance including an example how the performance but also computational costs increase when adding more training instances.

We have demonstrated the ways of using our library in a real application. The advantage of the library is its small size. That could play a major role in the decision of a developer whether to use it or not. It is because the most of Android applications are targeted to the Google Play service, where the users download the applications online from their devices and mobile Internet connections are often slow. Moreover, the memory for applications is limited on the Android platform. There is a port of Weka to Android, but it is much bigger and maybe too complicated for a programmer with no experience in machine learning.

The approach of the thesis has been to point out the importance of using advanced algorithms in practice. Nowadays, a user of the information technologies is flooded by vast data and it is important to simplify his orientation. It is often not possible to program it directly, however programs can learn to follow regular patterns on their own.

Besides the development of a learning algorithm, it is important to come up with a useful task, and then to define and represent it well. For example, in classification of culture events according to the degree of interest of the user, it is important to decide what the input attributes will be. Definitely, it would be the keywords describing the genre and the venue, but we could take the advantage of the mobility of the target devices–consider the current location of the user, ask him whether he is alone, with friends or his partner and so on.

The sample application, which is a part of the thesis, seems to be practically usable, mainly due to the capability of working with real online data. Before a public release, there are still a lot of necessary things to improve, mostly in terms of graphics and user interface. The users are especially perceptive to these. It is expected that an application respects the newest UI standards of the Android platform and provides backward compatibility for the older Android versions as well.

We believe that not only our main result, the machine learning library, but also the example application aimed at pre-selection of culture events has a potential to be widely used and we are going to continue with their development.

Bibliography

[1] Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. *The WEKA data mining software: an update*. SIGKDD Explorations, 11(1):10–18, November 2009.

[2] Mitchell, Tom M. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 0-07-042807-7.

[3] Russell, Stuart J. and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 1st edition, January 1995.

[4] Bojar, Ondřej and Zdeněk Žabokrtský. *CzEng0.9: Large Parallel Treebank with Rich Annotation*. Prague Bulletin of Mathematical Linguistics, vol. 92. Prague: Charles University, 2009. ISSN 0032-6585.

[5] Quinlan, J. R. *Induction of Decision Trees*. Machine Learning. Mar. 1986, 81-106.

[6] Manning, Christopher, Prabhakar Raghavan and Hinrich Schütze. *Introduction to Information Retrieval*, Cambridge University Press. 2008. Online at `http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html`.

[7] `http://www.android.com/about/` as of May 2012.

[8] `http://developer.android.com` as of May 2012.

A. Content of the CD

The attached CD contains the following items:

- sources - complete Eclipse projects with sources and javadoc for:
  1. classificationLib - Java project with the base of the library.
  2. androidClassificationLib - Android library, i.e. the full version of the library with helpers for Android GUI.
  3. EventAdviser - the Application using the library.
- bin
  1. jar file for the library base (not Android-specific)
  2. apk file with the application (installation file for Android device)
- pdf with the thesis

B. Summary of the main methods of the library

The complete documentation with more detailed (and all) methods of the library is included on the CD. Here, we mention the most important parts.

B.1 Base methods (Java)

```
public class Classifier implements java.io.Serializable
```

- The main library class. Represents a classification task. Enables the user to add examples, process learning algorithms and classify new instances. Attribute values are required to be represented as integers from 0 to n-1 where n is the number of possible values for the attribute, specified for each attribute when constructing the Classifier.

```
public Classifier(int[] ranges, int labelRange)
```

- Constructor for a new instance of classifier with no training data.
- ranges: the ranges of the attributes NOT including the label.
- labelRange: the number of possible labels.

```
public void addTrainingExample(int[] attributes, int
    label) throws Exception
```

- Adds a training example.
- attributes: The values for each attribute.
- label: The label of the example.

```
public void learn(Method method) throws Exception
```

- Learns from the available training examples using the given method.
- method: The method to be used (from enum).
- Throws an exception if there are no training examples.

```
public int classify(Method method, int[] attributes)
    throws Exception
```

- Classifies a new instance by the given method.
- method: The method to be used (from enum).

- attributes: The values of the attributes of the instance to be classified.
- returns: The label of the instance according to the method used.
- Throws an exception if the classification cannot be processed. That is typically caused by not calling the Learn method before.

public double probabilityOfLabel()

- Returns a number between 0 and 1 that determines the probability of the recent classification result.

public double[] probabilitiesOfLabels()

- Returns the probabilities of all label values for the recent classification. The probabilities are represented as double values from 0 to 1 and ordered the same way as the label value.

public boolean canClassify(Method method)

- Checks if the Classifier is ready to classify a new instance by the given method (i.e. Learn for this method has processed).

public boolean needReLearn(Method method)

- Checks if some new training examples have been added since the last Learn process for the given method.

B.2 Android extension

public class AndroidGUIHelper

- A class which helps to create a GUI for Android and provides data representing the selected values in GUI compatible with the Classifier class.

public LinearLayout makeForm(Context ctx, String fileName, **boolean** inAssets)

- Parses the given XML document, prepares the internal structures to be able to provide the rest of the public methods and produces the form.
- ctx: Android context (the Activity where the form should be).
- fileName: the file name of the XML document.
- inAssets: true if the document is located in assets, otherwise it is saved on the device.
- returns: LinearLayout with the form. For each field there is a TextView with its name and a Spinner with the possible values. The target attribute is not included (you can get GUI for it by using GetTextViewForTarget and GetTargetSpinner.

public int[] getSelectedValues()

- Gets the selected values as integers (ready to be passed to the Classifier).

public int getTargetValue()

- Gets the selected target value (label) as an integer (ready to be passed to the Classifier).

public Spinner getTargetSpinner()

- Gets the GUI for the target spinner (ComboBox).

public int[] getRanges()

- Gets the ranges of the attributes (ready to be passed to the Classifier)

public int getLabelRange()

- Gets the ranges of the attributes (ready to be passed to the Classifier)

public String getLabelValue(**int** i)

- Gets the string for a given label number.
- i: the integer typically returned from a classifying method.
- returs: a string value for the label as defined in XML.

public int getNumberOfAttributes()

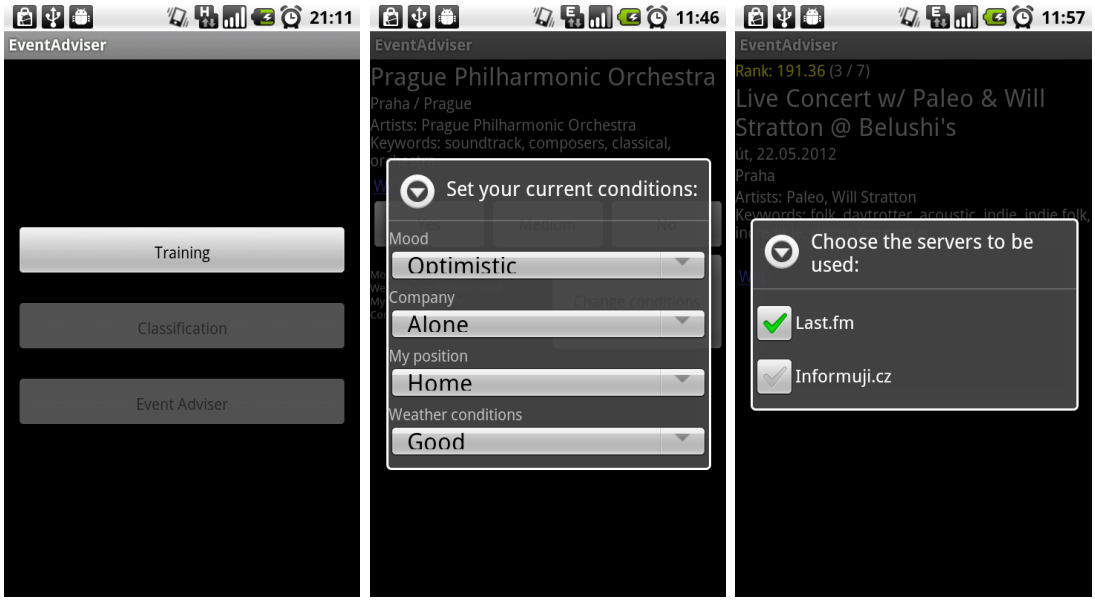
- Gets the number of attributes NOT including the label.

C. User documentation of the EventAdviser application

C.1 The first run

The first screen provides the menu, when you can select one of three modes. There is just one of them enabled at the beginning, because no training examples are available from the user.

The Set conditions dialog is shown when running the training mode for the first time. It can be open again from the options menu. The options menu is launched by the options menu button on the device and in every mode provides important functionalities (update data online, set source servers, reset training and reset data).

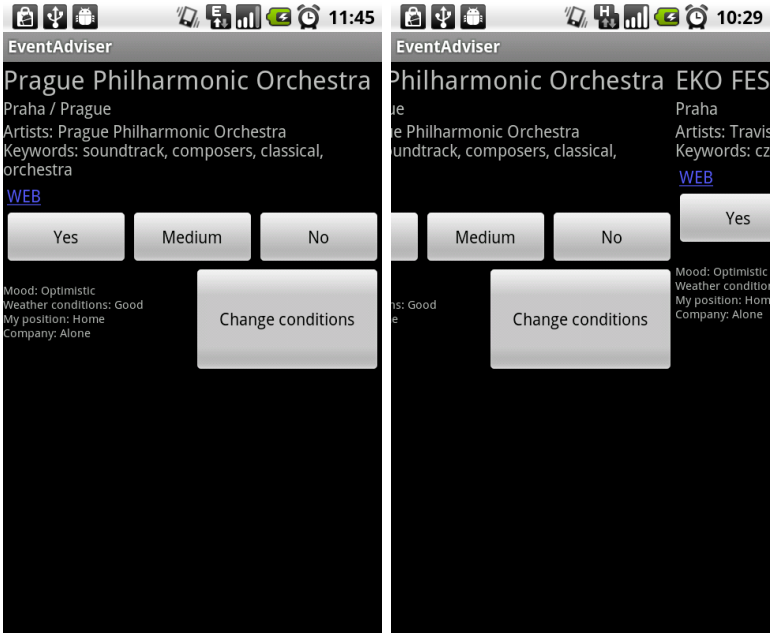


When running any mode for the first time, a help for the mode is shown.

The application has some default offline data about culture events, so you can start without the Internet connection.

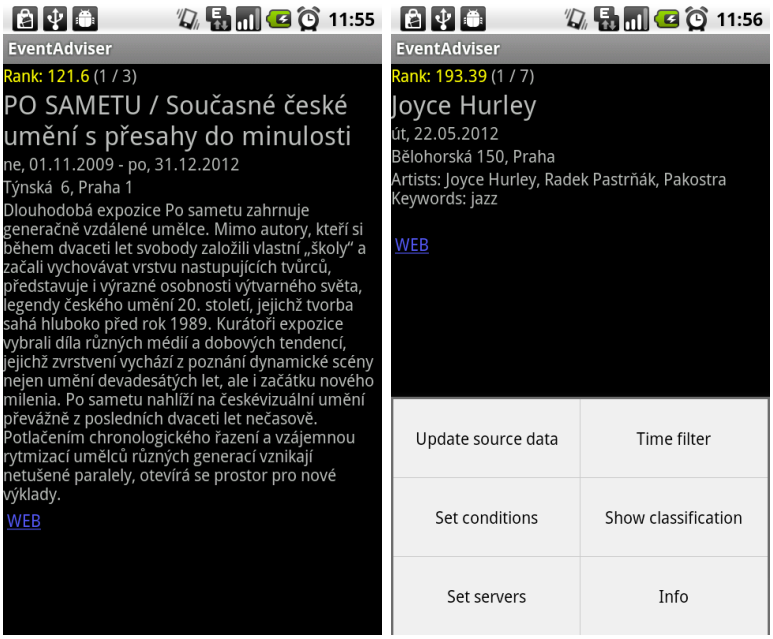
C.2 Training mode

In the training mode, the dates for the events are not shown, because it is not important–you should teach the Adviser about the character of events which are (un)interesting for you. This is learned together with the current weekday and set conditions. This means, such an event is (or is not) interesting for you when it is Friday and you are in the conditions you've set. The training mode lets you classify all the available events. They are displayed randomly and you do not have to classify each one, for example when the description is not sufficient. In such case you can just switch event by dragging the screen horizontally (like switching the Android desktop).



C.3 Main usage: Adviser mode

Now, let us see the result of the learning. By the back button you can return to main menu, where the rest of the menu items should be yet enabled. Select EventAdviser mode. You shall get the events for the next 2 days ordered from the probably most interesting one. They are displayed more detailed. Moreover, the events which seem to be really uninteresting for you are not shown at all.



What more exactly happens in the Adviser mode? The available data are firstly filtered by the time filter, which is set to "next 2 days" by default. You can change this in the options menu. These candidates are then classified according to the learned model. Remember, your current conditions are taken into account. For instance, if you have classified some rock concerts as interesting when you are alone and in the city center (conditions), some actual rock concerts will probably

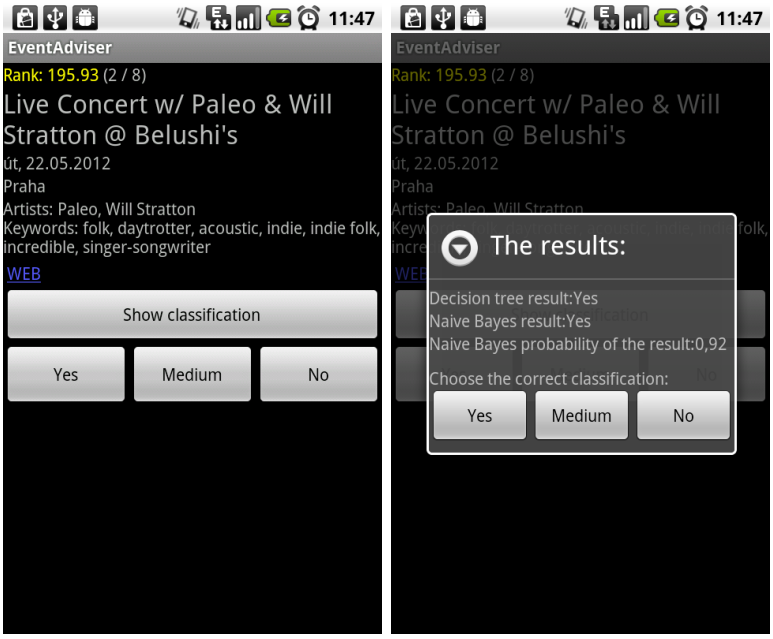
be advised. They may not be so interesting when you're at home and the weather is bad, but classical concerts could be good no matter your conditions. The Adviser can learn more complicated variations, because it classifies according to some keywords, which can describe the event more precisely.

When training, you say just Yes, Medium or No, but the application assigns a rank from 0 to 200 to each event. This expresses some probabilities and so on. You can see the rank at the top of the screen above each event.

In each mode, you can click on a link (if available) to be redirected to your web browser with a page containing more details about the selected event.

C.4 Optional details: Classification view

Let us examine the last mode - Classification. This is something between. You see just the time filtered events, but the ones with rank 0 are displayed, too. You can check the classification process of each event (meaning event + current conditions) more closely. It shows the results of 2 methods used by the application.



This does not have to be interesting for you, but after you see the results, you can set the correct answer and it will be added to the training examples and considered in the next advising. It can be used also when the answer is correct, to make the application "more sure". You can display this results dialog even in the Adviser mode from the options, but the Classifying mode has a specialized button to save one click.