DISSERTATIONES
MATHEMATICAE
UNIVERSITATIS
TARTUENSIS
111

ANNA LEONTJEVA

Using Generative Models to Combine Static and Sequential Features for Classification

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

111

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
111

ANNA LEONTJEVA

Using Generative Models to Combine Static and Sequential Features for Classification

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on 29th March, 2017 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors:

Prof. Marlon Dumas
University of Tartu, Estonia

Prof. Jaak Vilo
University of Tartu, Estonia

Opponents:

Prof. Alessandro Sperduti
Università Degli Studi Di Padova, Italy

D.Sc.(Tech.) Jaakko Hollmén
Aalto University, Finland

The public defense will take place on May 22, 2017 at 16:15 in J.Liivi 2-405.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu and supported by Internationalization Program DoRa carried out by Archimedes Foundation.

European Union
European Regional
Development Fund

Investing
in your future

ARCHIMEDES

DoRa

ISSN 1024-4212
ISBN 978-9949-77-404-3 (print)
ISBN 978-9949-77-405-0 (pdf)

University of Tartu Press
http://www.tyk.ee/

Abstract

The accuracy of a classification model is highly dependent on the feature set employed to train it. Extracting a suitable set of features from a dataset can be a challenging task, particularly when the data captures observations of phenomena that occur over time, such as a user interaction with a computer system, a business process or a human movement. Such data usually consists of features of an object that evolve over time (multivariate sequential features) and features that capture inherent properties of the object(s) under observation (static features). Most existing machine learning algorithms are designed to deal with either static or sequential features but not both. Yet, in real-life scenarios, both types of features co-exist. This thesis investigates the question of how to combine static and sequential features in order to increase the accuracy of a classification model. The thesis advocates a hybrid generative-discriminative approach to build classification models from sequential data and a framework that incorporates the hybrid model to make an early prediction for an incomplete sequence. The key idea is to use generative models (Hidden Markov Models and Recurrent Neural Networks) to extract a set of features from multivariate sequential data, and then to combine them with static features in order to build a discriminative model. The main hypothesis is that the combination of generative and discriminative models allows one to extract complementary patterns from the data and to integrate them effectively in the resulting classification model. The proposed hybrid approach is applied to three case studies from different domains and involving different data types, specifically: (i) prediction of deviant cases of a business process; (ii) detection of stealthy fraudulent users in a social network; and (iii) discrimination of imaginary movements in electrocorticography data. The latter case study illustrates the use of the approach in the context of post-mortem classification of completed sequences, while the former two involve early classification, where the goal is to predict the label of an incomplete sequence based on a model built from a set of complete sequences. The advocated approach is able to discriminate between deviant business process executions and normal ones after only 5 events with an Area-Under-the-Curve (AUC) of 90%, to detect 68% of stealthy fraudulent users with a 5% false positive rate, and to achieve an accuracy of 80% for movement classification on electrocorticography data. The thesis empirically compares the proposed approach against baseline approaches and elicits properties that a dataset should exhibit in order to benefit from the advocated approach.

Contents

Abstract **5**

List of abbreviations **10**

List of notations **11**

1 Introduction **14**

1.1 Contributions 17

1.2 Outline 18

2 Background **20**

2.1 Classification 20

2.1.1 Classification pipeline 22

2.1.2 Evaluation measures 24

2.1.3 Splitting strategies 26

2.1.4 Classification tasks in the case studies 27

2.2 Data types 30

2.2.1 Sequences 32

2.2.2 Graphs 33

2.2.3 Data types in the case studies 33

2.2.4 Summary 38

2.3 Classification models 39

2.3.1 Discriminative classifiers 41

2.3.2 Generative classifiers 43

2.4 Ensemble learning 55

3 Hybrid models and progressive index-based framework 60

3.1 Feature extraction from mixed data types 60

3.1.1 Feature extraction from sequences 60

3.1.2 Feature extraction from graphs 65

3.2 Hybrid models 66

3.3 Notion of earliness 75

3.4 Progressive index-based framework 76

4 Case study I: PBPM 81

4.1 Introduction 81

4.2 Related work 82

4.3 Complex symbolic sequence encodings in the PBPM domain 84

4.4 Evaluation 89

4.4.1 Datasets 89

4.4.2 Evaluation procedure 91

4.4.3 Results 91

4.5 Discussion 95

4.6 Conclusions 99

5 Case study II: Fraud detection 101

5.1 Introduction 101

5.2 Related work 103

5.3 Fraud detection classifier and its inputs 104

5.3.1 Evaluation procedure 108

5.3.2 Results 109

5.4 Conclusions 112

6 Case study III: ECoG signal discrimination 114

6.1 Introduction 114

6.2 Prerequisites 115

6.3 Methods 116

6.3.1 Stand-alone models for unicomponent data 116

6.3.2 Stand-alone models for bicomponent data 117

6.3.3 Multiple models on bicomponent data 118
6.4 Evaluation 122
6.4.1 Evaluation procedure 122
6.4.2 Datasets 125
6.4.3 Results 128
6.5 Conclusions 131

7 Conclusion and future directions 132
7.1 Summary of contributions 132
7.2 Future directions 134

Appendix 137

Acknowledgements 138

Kokkuvõte (Summary in Estonian) 140

Bibliography 143

Curriculum Vitae 159

List of original publications 161

List of abbreviations

AR	autoregressive
ARMA	autoregressive moving average
AUC	area under ROC curve
BCI	brain computer interface
BPI	business process intelligence
BPM	business process mining
ECoG	electrocorticography data
FPR	false positive rate
GB	gigabyte
GHz	gigahertz
HMM	hidden markov models
kNN	k-nearest neghobrs
LLR	log-likelihood ratio
LSTM	long short-term memory unit
MA	moving average
ME	mixture of experts
ML	machine learning
MRI	magnetic resonance imaging
MSE	mean squared error
PBPM	predictive business process monitoring
RAM	random-access memory
ReLU	rectified linear unit
RF	random forest
RMSProp	root mean square propagation
RNN	recurrent neural networks
ROC	receiver operating characteristic curve
RQ	research question
SVM	support vector machines
TPR	true positive rate

List of notations

In this list we present the common notations used in the work. We aim to use the most common and consistent conventions, but notation systems differ significantly in the literature and across domains. Therefore, the same symbol may sometimes have multiple meanings, as well as the same meaning can have different notations. Moreover, in some cases we keep the difference to highlight the domain-specific value of it. In general, lowercase letters denote indices and uppercase letters denote constants (for example, timepoints are $t = 1, \ldots, T$). Boldface lowercase letters indicate vectors (e.g. $\mathbf{x}$), while boldface uppercase is a matrix (e.g. $\mathbf{A}$).

Notations for heavily-notated topics (e.g Baum-Welch algorithm and LSTM unit) are described directly in the corresponding sections.

Symbol	**Meaning**
$\mathcal{D}$	a (training) dataset
$\mathcal{D}_{test}$	a test dataset
$\mathbf{x_i} = (x_{i1}, \ldots, x_{im})$	a feature vector for i-th instance (with m features)
x_i	a value of a single feature for i-th instance
y_i	an output for i-th instance
N	total number of instances in a dataset $\mathcal{D}$ with i-th instance $i \in 1, \ldots, N$
M	total number of features in a dataset $\mathcal{D}$, $m \in 1, \ldots, M$
M_d	total number of sequential features in $\mathcal{D}$, $m_d \in 1, \ldots, M_d$
M_s	total number of static features in $\mathcal{D}$, $m_s \in 1, \ldots, M_s$

C	total number of classes/levels in categorical feature, c-th level is $c \in 1, \ldots, C$
$\hat{y}$	prediction of an output, either of a class or a score
$\mathbb{I}_c(x)$	indicator function for c-th level
Ω, $\boldsymbol{\Omega}$	a vocabulary for simple and complex sequences, finite set of elements
$\mathbf{x}_i^{seq}$	a simple symbolic sequence of instance i and length t
$\mathbf{X}_i^{seq}$	a complex symbolic sequence of instance i, of length t and with m features
$\mathbf{x}^{static}$, $\mathbf{x}^{seq}$	feature vector static and sequential (explicitly highlighted)
T	length of a sequence, t-th timepoint is $t = 1, \ldots, T$
$p(x)$	probability
$X \sim p$	random variable X distributed according to some distribution p
θ	parameter vector
$\mathbb{L}(\theta)$, $l(\theta)$	likelihood and log-likelihood function
$s, s^{'}$	states, $s, s^{'} \in S$
o	observations, $o \in O$
K	number of hidden states in HMM or number of clusters or number of layers in a network
$\mathbf{w}$, $\mathbf{W}$	weight vector, weight matrix
$L(\hat{y}, y)$	loss function
$\mathbf{s_t}$	hidden node at timestamp t
$\odot$	elementwise multiplication
$\mathbf{x} \oplus \mathbf{x}^{'}$	concatenation of two vectors
$\mathcal{L}_k$, $\mathcal{L}_E$	baselearner and ensemble learner
$h(\mathcal{L})$	aggregation functions for ensemble, $\mathcal{L}_E = h(\mathcal{L}_k)$

Domain-specific notations:

Symbol	**Meaning**
$\mathbf{BPL}$	a set of historical traces in a BPM log (different versions are indexed: $\mathbf{BPL}_1$)
σ	a sequence corresponding to a business process
$index$	similar to a timepoint in BPM
e	event class in a trace
$\mathbf{SKY}$	a set of users represented via some set of features
$\mathbf{ECOG}$	a set of recorded trials in ECoG dataset

Chapter 1

Introduction

Classification is one of the most heavily used machine learning (ML) approaches as many real-life problems can be defined in terms of the classification task. Discrimination between spam and non-spam emails based on the content, detection of a disease based on the symptoms, categorization of cells as benign or malignant based on MRI pictures are all prominent examples of the classification task. In addition, several aspects contribute to the classification task popularity among ML practitioners, such as the existence of a well-defined pipeline, ground truth comparison and the variety of available methods. This leads to a number of frameworks that simplify the modeling procedure and, thus, reduce the cost of trying different models in order to choose the best one according to some goodness measure. Whether a model succeeds in solving a particular task mostly depends on the chosen feature space rather than on the chosen model. Despite recent efforts to make the feature extraction from raw data automatic [Kriz 12, Hint 06, Beng 13], it is still considered to be a long, iterative, and a largely creative process, where the crucial goal is to construct informative, independent features that discriminate between the classes and help the model to generalize well. In this step, both the domain knowledge and the machine learning expertise are equally required, which makes the whole process inefficient in terms of the manual effort.

Moreover, raw data are not always representable through independent features in a straightforward manner. Data types such as sequences and graphs do not have a natural feature vector representation. There are multiple ways to aggregate the information or to come up with more advanced features [Xing 10, Ries 10, Lin 03]. The task becomes even more challenging when the data consist of different data types. It often leads to some data being either discarded or heavily aggregated as most of the machine learning methods are not designed to deal with such a mixture.

In this work we show different possibilities how to handle multivariate sequential data along with static features and graph data in a classification task. We demonstrate that the results of common discriminative methods can be improved by applying a generative approach for the feature extraction. Generative models belong to the class of models that optimize joint probability, thus, aiming to reproduce the input data. Applied on multivariate sequences, they create a representation of the data in contrast to the discriminative models, which only seek for the boundary between the classes. The features extracted from the generative models enable the subsequently applied discriminative classifier to access the sequential or structural information in the data. Therefore, the research hypothesis of this thesis is that the use of generative models on the multivariate sequential data together with the static features improves the resulting classification model.

In this work, we propose the hybrid generative-discriminative approach, which combines static and sequential components of a dataset and uses them for the classification task. It is a two-step procedure, where we first train a generative model on the multivariate sequential component of the data and then use the output of the generative model along with the static component to train a classifier. We compare the hybrid approach with another way of model combination – an ensemble. In an ensemble predictions of different models are used as features to train another classifier.

We apply the hybrid approach on three real-life problems from different domains. By putting the problem of combining different data types in the context of each domain we demonstrate the flexibility of the suggested framework. First two case studies explore the possibility of early class detection, where the goal is to predict a label using an incomplete sequence. The third case study uses full-length sequences for the prediction in an effort to discover the latent patterns. We discuss the details further and provide the full machine learning pipeline for each of the case studies.

The first case study comes from the domain of Business Process Mining (BPM) and aims to solve a predictive business process monitoring (PBPM) problem, where the goal is to predict the outcome of an ongoing business case as early as possible (for example, predicting whether the case will deviate from the normal process flow). Commonly, a case is represented as a sequence of events accompanied with the descriptive information that can also be sequential. Such descriptive information in BPM domain is called a payload. Earlier works on PBPM used different levels of aggregation: either by calculating frequencies or using limited amount of payload information. We show that the data can be represented via complex symbolic sequences as well as list different ways of dealing with such a data type. We explore simpler baselines that partially discard the information and compare them to the approaches where all the available information is fed into the classifier. We demonstrate that by using all available information from the logs we significantly improve the results. We propose a general progressive index-based framework that is beneficial for an early prediction of the outcome of a business process. The results indicate that using this framework we achieve high predictive power at early stages of the process – by using the first few events.

The second case study analyses the problem of fraudulent users in Skype. While being a very sensitive problem for the company, it has been studied for some time. On one hand, the existing system already detects a vast majority of fraudulent users. On the other hand, it is mostly semi-automatic with a considerable amount of manual effort. We use available labeled data to fit a machine learning model in order to detect fraudulent users in order to potentially reduce the required manual effort. Furthermore, we improve the existing approaches by finding fraudulent users that had not been detected in first moments of their existence, which means that their behavior is less common. The available data come from different sources, including dynamic social graph and time series of Skype usage. The challenge is to fuse these data types together. We apply hybrid approach with hand-crafted, static features along with the ones extracted by applying the generative approach, and demonstrate the usefulness of such combination.

The third case study is focused on multivariate time series of electrocorticography data (ECoG) with the aim of classifying brain signal when patients perform an imaginary movement of tongue or finger. While the first two case studies explore feature extraction from the combination of data types, the last part of the thesis aims at answering a slightly different question: whether the feature extraction from a generative model – a proposed *hybrid approach* is superior to an *ensemble approach*, where the predictions from the generative and the discriminative models are combined. In addition, we alter different parameters of a simulated dataset and derive a few interesting observations regarding these parameters and how they affect the classification accuracy. We provide insights into the properties a dataset should exhibit in order to benefit from the hybrid method.

We believe that this work can be of particular interest for the practitioners, who are seeking for model improvements in real-life classification problems.

1.1 Contributions

To summarize, the contributions of this work are the following:

- We provide an overview of various ways to process data types such as combination of sequences, static features and graphs in order to use the data for the classification task purposes.
- We demonstrate how the machine learning pipeline can be adopted for the cases when different data types are present. We do so on several case studies.
- We suggest a hybrid generative-discriminative classification model that efficiently combines static and sequential components of a dataset.
- We propose a progressive index-based framework that is beneficial for an early prediction for multivariate sequential data and empirically validate the approach for the task of predictive business process monitoring.
- We define several feature representations and compare them with the suggested framework for an early prediction of process outcomes in the BPM domain.

- We discuss how a combination of such data types as static features, sequences, graphs, and sequences of graphs can enrich the classifier for a fraud detection task.

- We compare hybrid and ensemble approaches, as well as “simpler” models for a classification tasks on ECoG data and several public datasets.

- With the help of a synthetic dataset we provide some suggestions how and when to combine static and sequential components and when to choose more complex methods over “simpler” models.

1.2 Outline

The chapters in the thesis are grouped into three parts: Chapter 2 discusses definitions, principles and basic concepts from machine learning field that help the reader to go through the rest of the thesis. Examples and illustrations are based on the case studies that are presented later.

Chapter 3 provides an overview of the existing solutions to the problem of combining static and sequential features for the purpose of the classification task. The chapter further describes the hybrid models and the proposed progressive index-based methodology as well as reviews the literature in this field. The notion of earliness is outlined in the context of the suggested framework along with the reviewed literature on this topic.

Chapters 4, 5, and 6 belong to the third part of the thesis, where we describe three case studies from different domains. In Chapter 4 we use the classification model to make early predictions about deviant and regular business processes. We demonstrate how historical log data can be represented in terms of complex symbolic sequences, where both static and sequential data types are present. We make a hypothesis that using both the control flow and the accompanying data payload helps to capture patterns earlier compared to other baselines methods. We suggest to use for the classification task index-based encoding and hybrid models that take into account both data types. We support the hypothesis by the experimental pipeline, where the baselines that rely on partial information are compared to the suggested approaches. We show that we are able to achieve good accuracy by making early predictions in case of two data logs and different classification tasks. The results of this chapter were published in [Leon 15] (Publication I).

Chapter 5 is based on the work published in [Leon 13] (Publication II), where the hybrid approach based on Hidden Markov Models is used for the discrimination of stealthy fraudulent users from the regular ones. We also have different types of information at our disposal: static and sequential data, as well as the dynamic social network. The idea is to combine all of the information under the umbrella of a single classification model, but instead of relying on aggregated summary statistics, we propose to utilize generative model properties of Hidden Markov Models. The early predictions are crucial for this task. We demonstrate that the technique is promising by achieving 68% of true positive rate with 5% false positive rate.

The last case study is described in the Chapter 6, which is based on the article [Leon 16] (Publication III). We apply the suggested hybrid model with such generative methods as Hidden Markov Models and Recurrent Neural Networks with Long Short-term Memory units on ECoG data, where the aim is to classify between imaginary movements. We demonstrate that raw sequential data can be decomposed into both static and sequential features. Both of these data types can serve as an input to the classification model if we use the hybrid approach. We use the ECoG dataset and several other datasets in order to experimentally compare hybrid model to ensembles, and to simpler techniques that use only static or sequential data. We provide insights about data-specific parameters that can help to choose a proper model for such a task.

Concluding remarks and directions for future research are outlined in Chapter 7.

Chapter 2

Background

2.1 Classification

According to the common definition, *machine learning* (ML) is the subfield of computer science, which explores and constructs sets of algorithms that "give computers the ability to learn without being explicitly programmed" (Arthur L. Samuel). Most often the term machine learning can be heard in the context of big data as it is unfeasible to analyze huge amount of data with regular analytical means or get useful information unless we apply automatic pattern recognition[Murp 12, Han 11, Alpa 14]. In contrast to the statistical hypothesis testing, in ML it is not necessary to define the result of the analytical task in advance. Nowadays, the data are often collected automatically without any particular goal, and the owner of the data has a very vague understanding of what he wants to achieve with the analysis. The beauty of machine learning is that ML provides methods not only in case of the well-defined hypotheses, but can be applied to seek for any interesting patterns in the data.

Algorithms that "give computers the ability to learn" are essentially ones that can *automatically* detect patterns in the data. Once these algorithms are "learned", they can be applied on new or future data for the prediction purposes or can help to extract digested information for decision making.

The field of machine learning is wide and expanding even more. For example, some universities devote a whole curriculum to this field [Dono 15] as the demand for the ML expertise is increasing. Typically, ML can be divided into three main types: the supervised, unsupervised, and reinforcement learning. The main difference of supervised approach from unsupervised is that there is a "right" answer, or ground-truth; in other words, the labels are provided along with the data. Supervised algorithms are trained to predict that label using information from the data. Therefore, the problem is well-defined in terms of the error between predictions and the real output – the smaller the mistake the better.

In unsupervised methods labels are not provided and the main task is to *cluster* data, which means grouping of objects by some defined similarity. The goal is to achieve such a division into groups, where the objects within the groups are very similar, but not so much between the groups.

In the case of reinforcement learning, the methodology is somewhat different: the learning is performed by so-called agents that take actions in an environment and get feedback either by getting a reward or a punishment. Their goal is to maximize the reward and, thus, to determine an optimal behavior in the given environment. Recently, this subfield of machine learning gained a lot of attention in the light of solving many tasks that were previously unsolved, e.g. learning how to play Atari games. Another attention spike was caused when the reinforcement algorithm won a human professional in the advanced game Go [Mnih 13].

In this work, we mainly concentrate on problems related to the classification task, which is a subtask of the supervised learning approach. Suppose we are given a set of pairs $\mathcal{D} = \{(\mathbf{x_i}, y_i)\}$, $i = 1, .., N$, were $\mathbf{x_i}$ is an input, y_i is an output for a particular *instance* i, and $\mathcal{D}$ is a *training set* of size N. The goal of the supervised approach is to estimate an unknown function from the input object $\mathbf{x}$ to output y:

$$f : \mathbf{x} \rightarrow y \tag{2.1}$$

The estimated function should be such approximation $\hat{y} \sim \hat{f}(\mathbf{x})$ that generalizes the relationship present in the data. In *predictive setting* it is applied to predict output y for new instances by providing only input $\mathbf{x}$ as opposite to the *descriptive setting*, where the goal is to explain assumed generation process of the data.

The type of output y dictates the subclass of the supervised methods: in case of continuous values, the problem is called *regression*, while in case of categorical values of the output, the *classification task* is performed. Two-class problem is called *binary*, but in general case $y_i \in \{1, .., C\}$, where C is number of distinct classes and the problem is called *multiclass classification.*

Continuous and categorical values are the most common types of output with most of algorithms dealing only with these two types. However, more complex problems are related to a non-trivial output type, like y being a sequence (called temporal classification or sequence-to-sequence classification) or any other structure (in general, referred to as structural learning). The same is valid for the input $\mathbf{x}$, so-called *feature space.* Features (also often called variables or attributes) are vectors of size N. Let M be the number of features. Then, for each instance i, x_i is an M-dimensional *feature vector*, and $\mathbf{x}$ is $N \times M$ feature matrix. In its simplest form, features are either real-valued or categorical vectors, for example, the height or the gender of a person, respectively. However, input data can come in various forms, such as pictures, sequences, graphs, or any other objects. They are often referred to as *complex data types* [Han 11]. In that case, it is not immediately straightforward how to represent such structures in a classical ML setup via vector representations [Murp 12]. Even more challenging is the situation when the combination of various complex data types is present.

In this work, we discuss the representations of such complex data types as sequences and graphs, their combination with static features, and how to incorporate them together into the single ML pipeline. In the next section, we discuss the ML pipeline itself.

2.1.1 Classification pipeline

The classification approach is summarized in the Diagram 2.1: features are extracted from the raw data and the resulting feature matrix is split into subsets: training, validation, and test samples. The main goal of such splits is to prevent model from *overfitting*, which occurs when a model tries to learn every fluctuation of the data instead of capturing general underlying patterns. Usually it happens when complex models with many parameters (compared to the number of data samples) are favored over simpler ones. In case of overfitting, the training error becomes very small, but such a model does not generalize to the test set, and, therefore to the real-world data. In that case, estimating the performance on the unseen data helps to diagnose the existence of overfitting. A validation set is required to be able to choose *hyperparameters* – parameters of the models that can affect the performance. The validation set helps to check performance iteratively without seeing the test set. Multiple splitting strategies exist and we discuss several of them in Section 2.1.3.

Next phase (Figure 2.1) is to fit a model to the training data, where the algorithm can be tailored to the needs: models can be either *interpretable*, but in general weaker in their performance, like logistic regression and decision trees, or *black-boxes*, like Random Forest or gradient boosted trees — ensembles of many models that perform generally better, but more difficult to understand the reasoning behind them.

Each fitted model is evaluated on a validation set using some measure of performance. Again, there are various measures of model success that can be used depending on several factors and on the field of application. Most of measures for the classification are based on the *confusion matrix*. For binary output often receiver operating characteristic (ROC) together with Area under ROC curve (AUC) are used [Fawc 06]. We discuss evaluation measures in more detail in Section 2.1.2.

If the resulting model does not perform well, there are many different ways to proceed. For example, we can fine-tune model parameters (or *hyperparameters*), or change the feature space by extracting other features. Another option to consider is to change the algorithm we use for the classification.

Once we find the best model among all the fitted ones according to the defined metric, we apply it to the unseen data to make predictions. Predictions are rarely provided in terms of classes, mostly they are on a continuous scale – for binary classification from 0 to 1, where the cut-off for the final class prediction can be chosen.

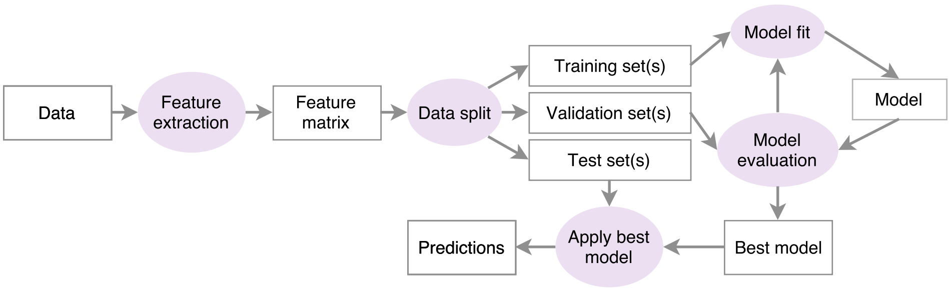


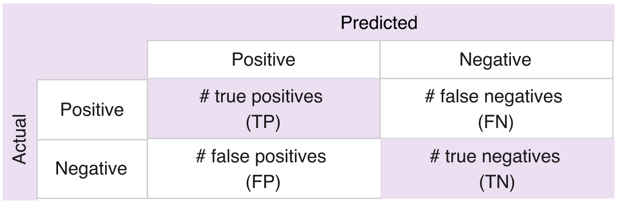
Figure 2.1: Machine learning pipeline

2.1.2 Evaluation measures

One of the most important parts of the machine learning approach is its established experimental pipeline. While the collection of learning methods is overlapping with the existing "classical" statistical approaches (where regression is the most well-known example of such), the experimental pipeline in machine learning is different. Instead of sophisticated *hypothesis testing* in statistics, machine learning adopts the seemingly simple *splitting strategy*, which is, however, theoretically justified by a field of statistical learning [Frie 01]. Such concepts as overfitting-underfitting and bias-variance tradeoff can be considered as central in that regard.

There is no way to make claims about the "best classifiers" without a proper definition of the classifier evaluation measures. These measures have to be carefully chosen depending on a problem and the data at hand as there is no single measure that "fits all". We introduce the measures that are used in the experimental part of this work.

Recall that for the classification task we train a classifier by providing the ground-truth outcome $\mathbf{y}$. The algorithm uses a feature matrix to learn a model and calculates the predictions $\hat{\mathbf{y}}$ — for a given set of features it outputs the answer. It can be either on a same scale as the outcome (binary for the binary task and predicted classes for multilevel classification) or correspond to class probabilities (or other continuous scores) that sum to 1 for all predictions for a given instance x_i. It is straightforward to choose a threshold that would translate a score on the continuous scale to a class representation. Given a fixed threshold, we can calculate a *confusion matrix* that is the basis for many evaluation measures. It is defined as following:



		Predicted	
		Positive	Negative
Actual	Positive	# true positives (TP)	# false negatives (FN)
	Negative	# false positives (FP)	# true negatives (TN)

Figure 2.2: Confusion matrix

Note that in a machine learning task the positive cases correspond to the class of interest (encoded as 1), while the negative cases belong to the class of regular instances or they represent the majority of the cases (encoded as 0). Based on a confusion matrix we define:

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

$$\text{TPR (True Positive Rate)} = \text{TP}/(\text{TP} + \text{FN}) \text{ and}$$

$$\text{FPR (False Positive Rate)} = \text{FP}/(\text{FP} + \text{TN}),$$

where Accuracy is the number of correctly classified instances out of all. It shows the general goodness of a classifier handling negatives and positives with equal importance, while pair TPR-FPR is often used to detect rare instances. TPR shows the proportion of correctly classified positives and FPR shows the proportion of misclassified negatives.

In all measures that are based on a notion of the confusion matrix the threshold for continuous score is already fixed. However, the choice of such threshold is rarely straightforward. When the dataset is balanced with respect to the number of positives and negatives, it is usually a good choice to define the threshold as 0.5. Some prior knowledge of the classification problem can shift it to a smaller value, making the classifier more conservative (e.g. finding all suspicious users is of high priority, while labeling regular users as suspicious is less harmful), while shifting a threshold to a larger value than 0.5 leads to a more relaxed classifier towards positive (e.g. it is of higher priority not to block regular users from making transactions and only when the classifier is very certain, system blocks the transaction). But once the dataset is not balanced (e.g. fraudulent users, deviant cases, rare diseases represent only tiny percentage of the whole population), the choice of the threshold becomes more tricky. In order to assess the classification performance without the choice of the threshold the ROC (Receiver operating characteristic) curve is constructed.

ROC curve visualizes all possible tradeoffs of TPR and FPR, which allows us to inspect the results across different thresholds. The area under the ROC curve (AUC) condenses the information provided by a ROC curve into a single measure of performance [Brad 97] . A classifier of the random guess, expressed as a ROC curve, is represented by a diagonal line with AUC of 0.5, while the perfect classifier would score AUC of 1 and is represented by the ROC curve crossing the coordinates $(0, 1)$ - where FPR = 0 and TPR = 1. In other words, the AUC represents the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. Also, in addition to comparing models using their AUC, it is possible to fix the FPR to some small value of practical interest (for example 1% or 5%) and compare the TPR between the models.

2.1.3 Splitting strategies

In order to properly assess the quality of a classifier we need to adopt proper techniques in such a way that no overfitting occurs. Otherwise the measures are overly optimistic and may lead to data *memorization* rather than *learning*, or in other words, we want the estimation of the accuracy exhibit low variance and low bias. The main rule to keep in mind is not to evaluate the performance on the same data that the classifier was trained on. The basic splitting strategies are the train-test split (holdout) and k-fold cross-validation, which are depicted in Figure 2.3. However, bootstrap, leave-one-out cross-validation and resampling are also widely spread [Efro 97, Koha 95]. We describe briefly the main ideas behind the main strategy schemes.

train-test split is a very basic machine learning procedure, where the data are randomly split according to some proportion. More frequently are used $50\% - 50\%$ or $80\% - 20\%$ divisions. It is very simple procedure, but it uses the data inefficiently. Therefore, it is not recommended to use it when the dataset size is small. Moreover, the scheme is not suitable when we need to learn hyperparameters of the model. In that case either data has to be split equally on three disjoint subsets: training, validation and test sets or we can use k-fold cross-validation.

k-fold cross-validation is a technique, which is used either to estimate the measure of accuracy more reliably or to search for hyperparameters. Data is split on k folds and model is trained on $k-1$ folds and evaluated on the remaining one. The procedure is repeated until all the k folds were used for testing.

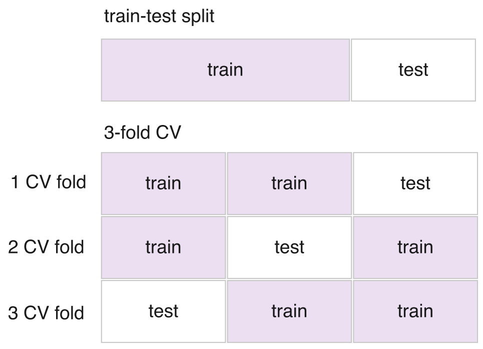


Figure 2.3: Basic splitting strategies: holdout split with 2/3 of the training set and 1/3 of the test set; and k-fold cross validation with $k = 3$.

Often, more complex splitting strategies are required, especially when the data has various deviations from the golden standard. For example, when the data exhibits temporal shifts or is highly imbalanced. In the case studies we describe in this thesis the evaluation procedures differ depending on the domain, data and the goal of the study. We describe each of the evaluation procedures separately. Please refer to Sections 4.4.2, 5.3.1, and 6.4.1 for the detailed descriptions of the procedures.

2.1.4 Classification tasks in the case studies

In this section we define the classification problems with respect to the various domains. Each of these problems refers to a particular case study that we later analyze in full. We use the case studies throughout the thesis to illustrate the ideas discussed previously.

Classification of business process outcomes An interesting example of a classification task can be found in the field of predictive business process monitoring (PBPM) [Leon 15]. For the classification task purposes a business process is analyzed from the perspective of *event logs*, consisting of *traces*. Each trace represents one execution of a process (a case). A trace is a sequence of events, each referring to an execution of an activity. Often, logs can be accompanied with the metadata, either on a trace or event level (*payload*). One of the tasks of PBPM in this case is to predict at runtime the most probable outcome of an incomplete case. The outcome in the context of a classification task (y) can be the fulfillment or the validity of some constraint (for example, a time constraint) or any other condition under interest of a business owner. For example, in sales process a possible outcome y is the placement of a purchase order by a potential customer, whereas in medical processes a possible outcome is the recovery of a patient upon the completion of the treatment (see [Leon 15] for more details)). In case of binary classification we refer to two possible outcomes as *deviant* and *non-deviant*, where the first does not satisfy the required constraints.

The task of a classification learner in this domain is to predict the outcome of an ongoing unlabeled case using the historical logs with the known outcomes. Note that the ongoing case is not yet completed, while the historical cases are. The formula 2.1 can be written as following:

$$f : \mathbf{BPL} \to \{\textit{deviant}, \textit{non-deviant}\}$$

where we denote with **BPL** a set of all historical traces available for the training. We need to extract for each case the feature vector $\mathbf{x_i} = (x_{i1}, \ldots x_{im}, \ldots, x_{iM})$, $i = 1, \ldots, N$ from the historical data **BPL**.

Classification of fraudulent users Valid financial transactions are often burdened by some amount of illicit transactions with the intention to cause parties to suffer a loss. Such financial institutions as banks, money transfer and e-commerce companies experience the problem of *fraudulent* transactions. The fraud detection task has been receiving a lot of attention [Fawc 97b, Bolt 02, Duma 16]. In general, the problem at hand is difficult to solve for multiple reasons. One of them is the variety of applications with different data sources depending on a particular application. More importantly, different applications lead also to the great variety of fraudulent schemes that are hard to detect without knowing what to look for. Depending on a specific application, there is a whole ontology of various fraudulent user types.

Even the detection levels can be many: sometimes it is important to prevent just suspicious transactions, sometimes a particular user or client or even whole groups of people involved in the suspicious activity should be blocked. Another aspect is that the problem cannot be solved once and forever as new schemes may be invented when the discovered fraudulent schemes do not work anymore.

We bring an example of classification task for the problem of fraudulent users detection in the context of Skype [Leon 12, Leon 13], one of the largest providers of Internet communication software. Skype is targeted by fraudulent users, who intentionally deceive other users or providers in many ways. The types of fraudulent users relevant to Skype include (but not limited to) credit card and other online payment fraudulent users, account take overs and account abusers.

As Skype already implements very sophisticated techniques to capture such users, it has collected a database with the verified labeling: for each user there is information together with the labeling. In the simplest setting, label (y) is binary: $flag = 1$ indicates that the user is fraudulent and $flag = 0$ otherwise. However, the problem can be seen as a multilabel classification, where users have a higher granularity division: different types of suspicious activities as well as different types of regular usage (e.g. whether a client makes payments or uses only free services). Also, naturally there are many choices over the feature space $\mathbf{x}$ as we can extract information from various aspects: data related to payments, social activity of users, or whether the provided personal information is in accordance to the user activity.

In the context of this example, the function 2.1 can be defined as:

$$f : \mathbf{SKY} \rightarrow \{fraud, regular\}$$

where each user $u_i \in \mathbf{SKY}$ is represented as some feature vector $\mathbf{x_i} = (x_{i1}, \ldots x_{im}, \ldots, x_{iM})$ and $\mathbf{SKY}$ is the dataset of Skype users.

Classification of electrocorticography signals (ECoG) The last illustrative example of a concept is based on the data of electrocorticography for brain-computer interface (BCI). According to the experimental setup [Lal 04], a subject had to perform imagined movements either with a finger or the tongue. During these movements the time series of the electrical brain activity were recorded with the help of the platinum 8×8 grid of electrodes on the right motor cortex. All recordings were performed with a sampling rate of 1000Hz. Every trial was recorded either for imagined tongue movements or an imagined finger movement, but not the both. The duration of each trial was 3 seconds.

The classifier here has to learn from the set of recorded trials and predict whether a trial corresponds to a imagined tongue movement or finger movement:

$$f : \mathbf{ECOG} \rightarrow \{finger, tongue\},$$

where **ECOG** is a set of recorded trials. The challenge here is to come up with the suitable feature vector $\mathbf{x}$, which is non-trivially extracted from the signals of 64 electrodes.

2.2 Data types

In the previous section we discussed how the task of supervised learning depends on two types of output y: numerical and categorical. Classification task expects the output of the categorical values. Recall that in general ML algorithms expect the data in a form shown in Figure 2.2:

$$\begin{array}{ccc} & features & labels \\ instances & \begin{pmatrix} x_{11} & \dots & x_{1j} & \dots & x_{1m} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \dots & x_{ij} & \dots & x_{im} \\ \vdots & & \vdots & & \vdots \\ x_{n1} & \dots & x_{nj} & \dots & x_{nm} \end{pmatrix} & \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix} \end{array} \tag{2.2}$$

In this section, we define and discuss different types of input $\mathbf{x}$. The most straightforward types of features are *continuous*, *discrete*, or *categorical.* Continuous features have numerical values that can be characterized by an infinite number of values between any two values. Discrete features consist of a countable numerical values, while categorical features have a finite number of values – *categories*. Categorical values might not have a particular order. But if they do, they are called *ordinal*. where the values can be put in a one-to-one correspondence with the discrete values. Most of the ML algorithms deal with these types of features (see the description of classification algorithms in Section 2.3.1).

Often data do not come in any of these types, but in more complex types – for example, we might want to classify pictures by objects of various shapes that are represented on them, or music by the hierarchy of genres, or get sense of networks, either biological like protein structures or social networks like Facebook. These are all examples of *complex data*. Complex data types are schematically shown in Figure 2.4. It appears that in real-life scenarios even more frequently we have to deal with the combination of different data types. We refer to the combination of various data types, complex or non-complex, as *mixed data types*. In this section we introduce the notions and definitions of such complex data types as sequences and graphs as prerequisites for the following case studies.

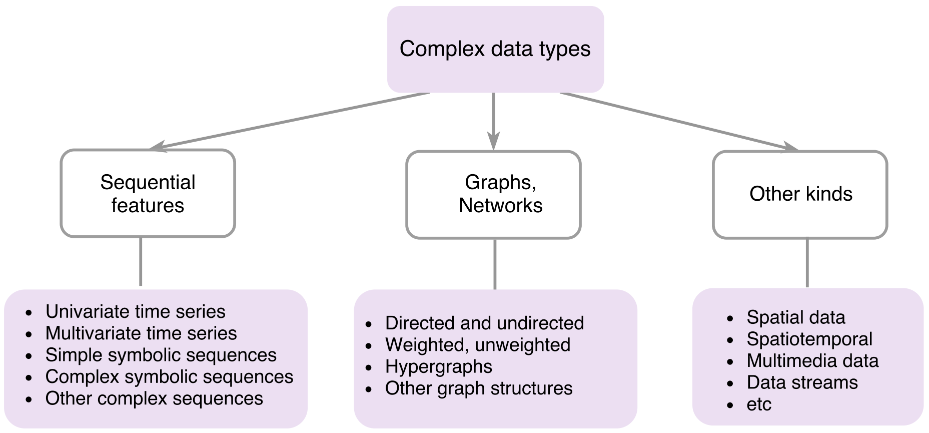


Figure 2.4: Complex data types. The illustration adopted from [Han 11].

In the aforementioned case studies (see Section 2.1.4) the most common type of input is *a sequence* that is one of the main focuses of this work.

2.2.1 Sequences

In the general case, a sequence is defined as an ordered list of elements, where an element can have a numerical real value or a symbolic representation. The main difference from the feature vector representation is that elements are not independent nor identically distributed (i.i.d). It is expected that a sequence exhibit a strong correlation between successive elements. One well-known and well-studied subtype of a sequence is *time series* – a sequence with the assigned time interval. Adopting the onthology from Xing, et al [Xing 10], we define the following types of sequences:

- *simple symbolic sequence* consists of ordered elements drawn from a finite set of elements called a vocabulary (or alphabet) Ω. $\mathbf{x}_i^{seq} = \{x_1, \ldots, x_t, \ldots, x_T\}$ is called a *sequence* if $x_{it} \in \Omega$, where t is the length of a sequence, $t = 1 : T$ and $i = 1 : N$. The dataset for N samples and T timestamps is a matrix of size $N \times T$.

- *complex symbolic sequence* consists of ordered vectors, where each vector is from a vocabulary (which consists of vectors itself). The sequence $\mathbf{X}_i^{seq}$ is then a matrix $\{x_{11}, \ldots, x_{mt}, \ldots, x_{MT}\}$, where $\mathbf{x}_{it} \in \boldsymbol{\Omega}$. For N samples, M features and T timestamps the dataset is a 3-dimensional matrix of size $N \times M \times T$.

- *univariate time series* are sequences of a single feature m, collected over equal time intervals.

- *multivariate time series* are sequences of multiple features, M, collected over equal time intervals.

- sequences of complex objects are sequences of complex data types. For example, the dynamic evolution of a network can be seen as a sequence of graphs in time.

In this work we consider a sequence as a *sequential* feature as an opposite to *static* features or *feature vector* as shown in Figure 2.2. In both time series and sequences we refer to discrete timepoints $t = 1, ...T$, which may also refer to positions or locations in the sequence.

2.2.2 Graphs

Social Network Analysis (SNA). has drawn a lot of attention due to the rise of social networks popularity, which have become a part of our everyday life. A graph is a common complex data structure that is fundamental to the SNA. Graph theory and applications are in heavy use in bioinformatics, chemistry, information networks and many more. Various tasks include calculations on graph structures such as link predictions, community detection, and network modeling. In this work we discuss the use of graph structures as a part of input data for the classification task — in the study of fraud detection (see Section 2.1.4) we adopt the social network combined with other data types in order to enrich the feature set.

Recall that a graph is a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is a set of vertices (or nodes) and $\mathbf{E}$ is a set of edges, where each edge connects a pair of vertices (nodes), i.e. $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. Often in large networks it is not feasible nor necessary to take into account the whole network. In that case we can compute metrics and seek for patterns taking into account only a local neighborhood of a particular vertex v. For a particular vertex $v \in V$, often referred to as *ego*, the first-order *egocentric network* focuses on the subgraph of G, where the direct connections of v and ties of this direct neighborhood are taken into account.

In this work we apply SNA to extract useful features like Page Rank and clustering coefficient in order to improve the classification accuracy of the fraud detection case study (for more details see Section 2.2.3.2).

2.2.3 Data types in the case studies

Now, we return to case studies introduced in Chapter 2.1.4 and explore the data types in each of these examples.

2.2.3.1 Data types in PBPM

In the example 2.1.4 we deal with the data of business process event logs [Leon 15]. For the classification task of deviant processes we use two different real-life logs, but of the same kind and structure: the BPI challenge 2011 [Dong 11] log (herein called $\mathbf{BPL_1}$) and an event log (herein called $\mathbf{BPL_2}$) of an Australian insurance company. The former log pertains to a healthcare process and describes the executions of processes related to the treatment of patients diagnosed with cancer in a large Dutch academic hospital. Each case refers to the treatment of a different patient. The second log relates to the handling process of insurance claims and covers about one year of completed cases.

Each case in the logs corresponds to a sequence of events describing its *control flow* or the ordered execution of events of the process. Each event is associated with data in the form of attribute-value pairs. All the associated data in the log is also referred to as *payload*. Moreover, each completed case is associated with an outcome - a *label*, which can assume binary or categorical values. We represent a case in the following form:

$$\mathsf{sequence}_i(\mathsf{event}_{i1}\{\mathsf{associated\ data}_{i1}\},\ldots,\mathsf{event}_{in}\{\mathsf{associated\ data}_{in}\}) : \mathsf{label}_i$$

Figure 2.5: Complex symbolic sequence

Note that cases of logs are represented as **complex symbolic sequences**, where features related to the whole case are *static*, but features related to events are *sequential*. Therefore this can be considered as a mixed data type of sequential and static features. In Figure 2.6 a case corresponds to one example from a hypothetical hospital log. Age is a static feature – it does not change throughout the case, while an event itself and a resource, corresponding to the procedure and the responsible party, respectively, are sequential data.

$$\begin{aligned}&\mathsf{sequence}_i(\mathsf{consultation}\{\mathsf{age{:}33,\ resource{:}general\ practitioner}\},\ldots,\\&\qquad\ldots,\mathsf{ultrasound}\{\mathsf{age{:}33,\ resource{:}nursing\ ward}\}) : \mathsf{false}\end{aligned}$$

Figure 2.6: Example of a complex symbolic sequence.

Table 2.1 summarizes the characteristics of these two logs (number of cases, number of events, and number of event classes).

Log	# Cases	# Events	# Event Classes
dataset $\mathbf{BPL_1}$	1,143	150,291	624
dataset $\mathbf{BPL_2}$	1,065	16,869	9

Table 2.1: Case study datasets.

In Section 3.1 we explore different options how to map complex symbolic sequences to a feature representation and empirically demonstrate which of the methods yields better classification accuracy.

2.2.3.2 Data types in fraudulent user detection

Earlier we mentioned that the classification task for this case study is to detect fraudulent users by training on the existing examples. Each instance requires a description that should be provided as a vector of features. The dataset **SKY** [Leon 13] represents a carefully anonymized snapshot that contains the Skype contact network, time series of the utilization of Skype products, and time series of the social/contact request activity of users. It does not contain information about individual calls and messages nor their contents. The data types that are extracted from this dataset can be divided into four groups (summarized in Table 2.2):

- *Profile set*: this set contains information about users, such as gender, age, country and OS platform. The information for this set is extracted from the profile a user provided at account registration time. Such information is not compulsory and is not verified by Skype. For many users it is incomplete or totally missing. However, it is the simplest feature set as it represents directly a feature vector with **static features** as these features do not change in time (or change very little).
- *Skype product usage*: this set of features is in the form of activity logs representing the total number of days per month a user was active using a specific product. None of the Skype usage data contained information about individual Skype communications, such as the parties involved in a communication, the content of communications, or

the time and date of communications. Rather, it merely contained the number of days in each month that a Skype user used a particular communications feature, such as Skype chat, Skype video calls, and Skype In and Skype Out calls. For example, such product as *"connected days"* represents the number of days per month a user was logged in Skype, while the product *"video calls"* shows how many days in total the user initialized a video call. Due to the limitations of the data, activity logs do not indicate which pairs of users actually communicate, nor distinguish a user who makes hundreds of calls from a user who makes just one. Note that all activity logs are in the form of monthly aggregated **time series**.

- *Local social activity*: relying on the user-to-user contact requests, we build a directed social graph with 677.8 million nodes and 4.2 billion directed edges with the accompanied timestamps of edge creations and deletions. Edge creation shows that a user (the sender) sends a a friendship request to another user (the receiver). If the receiver accepts the request, it is counted as another edge creation. Edge deletions are handled by the same principle. For each user we capture information about his social activity through the egocentric graph (one-hop neighborhood) and observe it over time. The data type we deal with in this case is **a sequential egocentric graph**.

- *Global social activity*: we compute such measures as PageRank and clustering coefficient. This is the most computationally expensive feature set – in order to compute these features a full social graph of users is taken into account. Here the data type is a regular, static, **graph** due to the computational reasons. Only the latest snapshot of the social network graph is utilized. We compute a PageRank score for every user on the Skype communication graph (see Section 3.1.2). Another input to our classifier that comes from the full contact graph is the local clustering coefficient [Watt 98] (see Section 3.1.2). As fraudulent users often seem to add unrelated users to their contact lists, their clustering coefficients are often lower than those of normal users.

Table 2.2: Sets of features with the corresponding data types and examples of features

Profile set	**Skype product usage**	**Local social activity**	**Global social activity**
static	**sequential**	**sequential graphs**	**static graph**
gender, age, country, OS	connected days, audio call days, video call days chat days	additions by a user, deletions by a user, additions of a user, deletions of a user, accept rate (%), degree	page rank, clustering coefficient

As we can observe, in this particular classification task various types of data are present. In some cases there is a straightforward way to use the provided data, but in others there are multiple ways to pre-process the data to turn it into a feature vector. Depending on a particular pre-processing we might lose a lot of information and miss patterns if we do not care about the connections or temporal aspects of the mixed data types. Later we demonstrate how such types can be combined together and discuss empirical results of such combinations.

2.2.3.3 Data types in ECoG movement discrimination

In the last example in 2.1.4 we described the experimental setup and the problem of detecting imaginary movements of tongue and finger. The ECoG recordings are provided in the form of *multivariate time series.* The data is three-dimensional: $(10584, 64, 300)$, where first dimension corresponds to samples, second — to channels and last one to time. Therefore, we have 10584 recordings of 300 ms duration for 64 channels, where each channel recordings originate from one of the 8×8 ECoG platinum electrode grid. The grid was placed on the contralateral (right) motor cortex. For each sample a corresponding label provided: whether it is a tongue or finger movement. In Figure 2.7 we depict two samples of all 64 channels for the whole duration of 300 ms, where the upper figure corresponds to the recordings during a finger movement and the lower figure — during a tongue movement. As we can observe, the data is quite noisy. The transformation of such data into a feature vector is not straightforward. In Section 3.1 we suggest different ways to extract features from multivariate time series that allow to capture both sequential and static components.

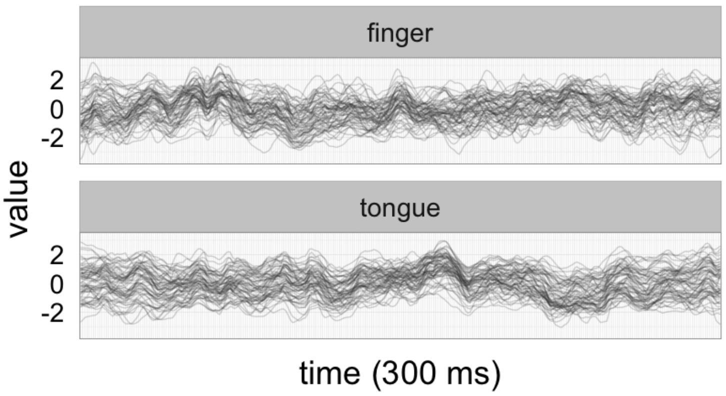


Figure 2.7: Two examples of ECoG multivariate time series for each of the classes.

2.2.4 Summary

In Table 2.3 we provide a brief summary of mixed data types that serve as the input for the classifiers in the case studies. As we can observe, all of the above mentioned tasks require non-trivial approaches to represent data in such a way that it fits ML pipeline for the classification task.

Table 2.3: Summary of complex data types presented in the case studies

SKY	time series of egocentric graphs, graph, time series
BPL$_1$, **BPL$_2$**	complex symbolic sequences
ECOG	multivariate time series

Next, we proceed with the background information about classification techniques, followed by the overview of the generative models as these are building blocks for the suggested frameworks.

2.3 Classification models

There is one essential division of machine learning methods that we use in this work: the *generative* and *discriminative* models. "Typical" classification methods belong to the discriminative family, while most of complex data types exhibit dependencies that are better captured by the generative approach.

As we mentioned earlier the ultimate task of the classification problem is to predict a class y. In that context *the discriminative* methods are specifically crafted maximizing class prediction accuracy directly, by learning the boundary between the classes of y. As opposite to *generative* models, these methods do not try to *capture* the underlying data. On a contrast, *the generative* approach is oriented finding the underlying processes that "generate" the data. In other words they model the underlying distribution, or distributions for each of the classes. We refer to the latter case as a *generative classifier*. The discriminative classifiers model the conditional probability $p(y|x)$, usually by maximizing $\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \theta)$, where θ corresponds to the parameters of the model, while generative classifiers maximize joint log likelihood $\sum_{i=1}^{N} \log p(y_i, \mathbf{x}_i|\theta)$ first and then can be used to calculate $p(y|x)$ by applying Bayesian rule.

Intuitively, the generative method has to extract more information from the data by modeling the distribution of the features, including all the temporal or structural connections between them. On one hand, a discriminative classifier yields better results in the prediction task as it is modeled directly [Vapn 98], on the other hand, it does not capture the information hidden in a complex data structure.

In the book [Murp 12], the author discusses pros and cons of the two approaches. We name just a few that are relevant in our context:

- Generative models are often easier to fit and under proper assumptions [Jord 02] they converge faster.
- Generative classifiers fit the distributions for each class independently, therefore, they do not need to be re-trained if we want to add an additional class. For example, if we describe three types of fraudulent users by a generative model and then discover another type, we train

an additional generative classifier. In case of a discriminative classifier introducing an additional class requires retraining.

- Generative classifiers are often useful when the problem is semi--supervised – some data are unlabeled.
- It is easier to handle missing features using generative approach. Discriminative methods are more sensitive to such problems.
- Discriminative methods can handle correlated features and be easily updated, while in generative models the violation of some assumptions can lead to incorrect results.

In Figure 2.8 we demonstrate a toy example with the two approaches. For the exact same data, in the left plot a generative model has to learn the distributions for each class, which are not straightforward and difficult to capture. That is, once the generative model is trained for each class, we should be able to generate a new data samples from the same distributions. For example, the distribution $p(x|y = 0)$ depicted with the red line is multimodal. In contrast, on the right plot the same data is used. But for the discriminative classifier the goal is to find a boundary between classes $y = 0$ and $y = 1$. As we optimize the conditional probabilities, to find the simple boundary of $x = 0.54$ with simple sigmoid function is quite easy. This toy example demonstrates that fitting generative models might be a challenging task compared to the training of discriminative models. However, a benefit from the generative methods can overcome a hurdle.

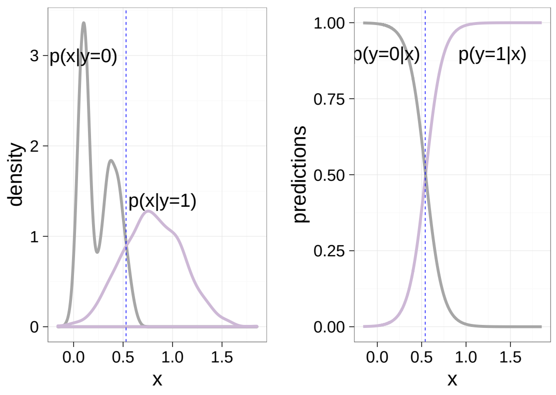


Figure 2.8: Data distributions for two classes. Adopted version of [Murp 12] and [Bish 06].

We would like to take best from the two worlds for the mixed data types. Discriminative approaches are widely used with *hand-crafted* features from complex data. On the one hand, one can extract various features that describe the structure or temporality, at least to some extent. For example, if we have a recording of consecutive timepoints, we implicitly might introduce the first order differences between them. However, it is time-consuming to come up with useful features as well as extract them. We suggest another approach, where we still use discriminative models, but combine them with the features coming from generative models. The generative model's features (see Section 2.3.2 for more details) can be learnt directly from the data and provide more information compared to the manually designed features, especially without any prior knowledge of the problem at hand. Potentially, they can better reflect the structure of complex data types.

In the next sections we describe a few different machine learning algorithms from both generative and discriminative families.

2.3.1 Discriminative classifiers

There is a variety of different algorithms for solving classification problem and countless different modifications of them. We limit ourselves to the well-known decision trees and to the Random Forest [Brei 01]. The decision trees are essential building blocks for more complex methods, while Random

Forest is a discriminative algorithm that we use in all case studies that provides state-of-the-art performance.

2.3.1.1 Decision tree

Decision tree is a common simple algorithm that learns decision rules, i.e. conditional "if-else" rules inferred from the data features to predict the output. The algorithm to construct the decision trees usually uses top-down best-split strategy, meaning that at each step the algorithm chooses a feature that splits data in the best way. "Best split" is defined by some measure, which dictates what version of the decision tree algorithm to apply: for example, ID3 and C4.5 use the information gain, while in CART the best split is defined by Gini impurity [Loh 11]. Decision tree method is easily interpretable, but has multiple issues regarding stability and tendency of overfitting. Even interpretability may be an issue – unpruned trees may grow so deep that simple interpretation of the decision rules is no longer possible. In addition, the dependencies in the data may not be representable in a "decision rule" nature. However, it serves as one of the best *base algorithms* (see Section 2.4 for more details) for more complex methods as for example, Random Forest.

2.3.1.2 Random Forest

We adopt Random Forest [Brei 01] as our main classification algorithm as it usually either outperforms other candidates or comes close to the top rankings, and it requires almost no of hyperparameter optimization [Caru 06]. This algorithm, first introduced by Breiman and Cutler, is a modification of bagging that builds an ensemble of decision tree classifiers. The idea behind the algorithm is to reduce variance by reducing the correlation between the trees. This reduction is achieved by growing unpruned trees with random feature selection. The final classification result is derived by combining the classification results of individual trees through majority voting. Since each tree is trained independently, the training procedure can be easily parallelized to be applicable for large data sets with many features. The most important parameters (*hyperparameters*) are: 1) number of trees in the forest (`ntree`). The larger the forest, the higher is the probability that every input instance of a training set gets predicted for a reasonable number of times, 2) number of features that are randomly sampled as candidates at each split (mtry). Essentially, it controls the amount of randomness. When mtry equals to all the features in the data, it becomes identical to the deterministic decision tree. According to Zhou et al. [Zhou 12], Random Forest has a better generalization ability compared to regular bagging (see Section 2.4) with the same amount of baselearners, and, in general, should be preferable over bagging technique. In more details the generalization ability of ensembles is described in Section 2.4.

2.3.2 Generative classifiers

Generative classifiers are considered to be weaker predictors as they optimize the joint probability, trying to capture the distribution of the underlying data instead of aiming at the discrimination between the classes. They are called generative as we can use the fitted model to generate new samples of data. In this work we use generative models as feature extractors rather than classifiers. Next, we describe the two models we apply in our work in more details.

2.3.2.1 Hidden Markov Models

Hidden Markov Models (HMM) have been the most frequently used technique for modeling sequence data since the 1980s [Good 16]. They belong to the class of graphical probabilistic generative approaches, which means they are used to generate samples from a joint distribution of observed and unobserved features. In case of HMM there is an assumption that an observed sequence is generated by some process that needs to be uncovered via probabilistic reasoning. The idea behind HMM is that a sequence consists of observed events generated by some hidden factors.

Despite many existing variations of HMM, the most commonly used is the first-order Markov process with discrete hidden states [Rabi 89]:

Definition 2.3.1 (first-order Markov model).

$$p(S_{1:T}) = p(S_1)p(S_2|S_1)p(S_3|S_2)\ldots = p(S_1)\prod_{t=2}^{T} p(S_t|S_{t-1})$$

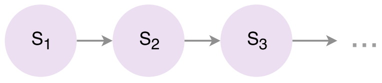


Figure 2.9: A first-order Markov model.

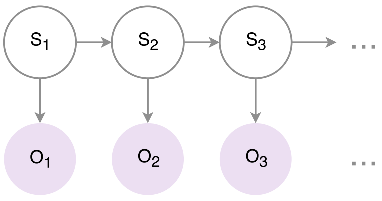


Figure 2.10: A first-order HMM

The assumption of a Markov process states that the probability of a given state depends only on the previous state, while ignoring the rest. See Figure 2.9 for a graphical representation of the first-order Markov model).

The conditional distributions of Markov model $p(S_t|S_{t-1})$ can be represented using *transition matrix*, where each element of the matrix is the probability of going from state s to state s': $A_{ss'} = p(S_t = s'|S_{t-1} = s)$. An HMM is a further extension of the Markov model, where assumed hidden process can be modeled as Markov process, but we see only the observations, not the hidden states themselves. The type of observations dictate the type of HMM. If observations are discrete, we can specify HMM with the following components:

1. the probability distribution of the initial states: $\boldsymbol{\pi} = p(S_1 = s)$, for all realizations of states, $s \in S$,

2. the transition probabilities between states: $\mathbf{A} = p(S_t = s'|S_{t-1} = s)$, for all $s, s' \in S$,

3. the emission probabilities (of the observable data) given a specific state: $\mathbf{B} = p(O_t = o|S_t = s)$, for all states $s \in S$, and observations $o \in O$.

If observations are not discrete, but continuous, Gaussian Mixture Models [Reyn 15] can be used for the parametrization so that each hidden state is described using the means and covariance matrices. The underlying assumption is that the data is generated as a mixture of finite number of Gaussian distributions with the unknown parameters.

In order to specify, train, and apply HMM for the purposes of feature extraction we perform the following steps:

1. Specification of the discriminative training of HMM.
2. Choosing the number of hidden states $K = |S|$.
3. Definition of the vocabulary Ω or distinct set of observation values.
4. Initialization of the model $\lambda = (\boldsymbol{\pi}, \mathbf{A}, \mathbf{B})$.
5. Training of the specified model λ.
6. Extraction of HMM-based features for further use in the next training phase.

Next, we describe each of the steps in more details.

1. The discriminative training of HMM. HMM is an example of a generative model, which means it can be fitted to unlabeled data. However, for the purposes of this work it has to take into account the information about the classes. Potentially, we can maximize the conditional likelihoods instead of the joint distribution, which may improve the accuracy of the classification. But it comes at the cost of more expensive procedure where standard Baum-Welch algorithm (see the algorithm below) cannot be applied. As we use HMM as a feature extractor, we argue that the standard, generative, HMM is better suited than the optimization of conditional likelihoods. Therefore, a more common approach is used, where HMM is fitted separately for each class. For example, using HMMs for the fraudulent user detection, we need to fit one HMM for fraudulent users (or each model for a different type of fraudulent users) and one for regular users. Note that in this case the number of hidden states may differ for each of the models. Next outlined steps should be applied in this case for each of the class-specific models separately.

2. The number of hidden states is a hyperparameter and should be specified in advance. However, similarly to such unsupervised methods as K-means, experience or domain knowledge are of no help here. We apply the most common approach to select the hyperparamer by using a grid-search over possible range of number of hidden states. Other possible ways of specifying K include the particular application of variational

Bayes [MacK 97, Murp 12] or the special version of HMMs based on the hierarchical Dirichlet process [Teh 12, Murp 12]. One important note is that instead of choosing the best number of hidden states by maximizing the likelihood or the discriminative accuracy of HMMs, we directly optimize for the accuracy of the final classifier that includes features from the HMM (for more details see 3.2).

3. The vocabulary Ω. There are several problems with the specification of the vocabulary of observations in advance. Often, the complete set of vocabulary is not known and therefore, the estimation of Ω on the training set leads to the problem where some observations in the test set are not specified. The problem is called *out-of-vocabulary observations.* The second problem occurs when the number of hidden states K increases resulting in the sparse emission distribution of the training set. It means that some observations appear only in one hidden state, but absent in others, which leads to the multiple problems with the model. The first problem can be solved by adding the vocabulary element that collects all the unknown observations. The second problem can be solved by applying different smoothing procedures, where one is to add small random noise to the emission probabilities in the training. Other smoothing techniques exist that do not assign equal weights to the rare observations, but they require some prior knowledge of the observations. Therefore, we resort to the simplest one without the use of priors.

4. The model initialization. In order to train a HMM we need to specify the initial components of the model λ, namely, $\boldsymbol{\pi}$, $\mathbf{A}$, $\mathbf{B}$. Again, there are different ways to do so. It is not recommended to initialize the model with zeros, as it has a danger of getting into local minima [Murp 12]. It is suggested to use either random initialization or use K-means to mitigate this problem (to some extent). The estimation of the parameters is later used to train EM with maximum likelihood on the same data. In our experiments we apply random initialization with several restarts.

5. Training: Baum-Welch algorithm. The Baum-Welch algorithm is a version of Expectation-Maximization (EM) method, where model λ

parameters are sought via maximum likelihood. It consists of two steps: **E** step, which uses forward-backward algorithm to fit parameters, and **M**-step, where we update the model parameters. We optimize the following function $\lambda^* = \operatorname{argmax}_\lambda p(O|\lambda)$. In other words we find such parameters of the model that maximize the probability of observations. We proceed in the following way:

- Initialize $\boldsymbol{\pi}$, $\mathbf{A}$, $\mathbf{B}$ randomly as discussed earlier.
- Apply *forward* procedure, namely, define the $\alpha_s(t)$ as the probability of observing the sequence till time t: $o_1, \dots, o_t$, where $t = 1, \dots, T$ and being in state s at time t: $\alpha_s(t) = p(O_1 = o_1, \dots, O_t = o_t, S_t = s|\lambda)$ We can find it recursively:
  1. $\alpha_s(1) = \pi_s b_s(o_1)$, where $s \in S$ and $b_s(o_1)$ is an element of emission matrix $\mathbf{B}$.
  2. at time $t+1$: $\alpha_{s'}(t+1) = b_{s'}(o_{t+1}) \sum_{s=1}^{K} \alpha_s(t) a_{s,s'}$, where $s' \in S$ and $a_{s,s'}$ is an element of transition matrix $\mathbf{A}$.
- Apply *backward* procedure. Define the probability of moving backward from time T to t given the state s at time t: $\beta_s(t) = p(O_T = o_T, O_{T-1} = o_{T-1} \dots, O_{t+1} = o_{t+1}|S_t = s, \lambda)$. Now, we can calculate the probability of moving backward recursively:
  1. $\beta_s(T) = 1$
  2. $\beta_s(t) = \sum_{s'=1}^{K} \beta_{s'}(t+1) a_{s,s'} b_{s'}(o_{t+1})$
- Update the parameters: $\pi_s^* = \gamma_s(1)$, $a_{s,s'}^* = \frac{\sum_{t=1}^{T-1} \xi_{s,s'}(t)}{\sum_{t=1}^{T-1} \gamma_s(t)}$, $b_s^*(v_j) = \frac{\sum_{t=1}^{T} \mathbf{I}(o_t = v_j) \gamma_s(t)}{\sum_{t=1}^{T} \gamma_s(t)}$ where:

$$\gamma_s(t) = p(S_t = s|O, \lambda) = \frac{\alpha_s(t)\beta_s(t)}{\sum_{s'=1}^{K} \alpha_{s'}(t)\beta_{s'}(t)}$$

$$\xi_{s,s'}(t) = p(S_t = s, S_{t+1} = s'|O, \lambda) = \frac{\alpha_s(t) a_{s,s'} \beta_{s'}(o_{t+1})}{\sum_{i=1}^{K} \alpha_i(T)}$$

$$\mathbb{I}(o_t = v_j) = \begin{cases} 1 & \text{if } o_t = v_j\text{, where } v_j \text{ is a letter from a vocabulary: } v_j \in \Omega \\ 0 & \text{if } o_t \neq v_j, v_j \in \Omega \end{cases}$$

The provided calculations may seem confusing at first, but essentially, the updated $\mathbf{A}$ and $\boldsymbol{\pi}$ in that case are normalized expected counts. New transition matrix intuitively means the sum of expected numbers of transitions from s to s' divided by the total number of transitions from s to anything else. For updated $\mathbf{B}$ we sum the expected number of times we are in state s and observe a symbol v_j divided by the expected number of times we are in state s.

6. Extraction of features from HMM Recall that we build separate models for each of the classes. Namely, we maximize the likelihood $\mathbb{L}(\lambda) = \prod_{i=1}^{N} p(\mathbf{x}_i|y = c, \lambda)$. Let us define λ_{pos} and λ_{neg} as generative models for each of two classes in a binary classification task (it is trivial to extend it to a larger number of classes) and $l(\lambda_{pos}), l(\lambda_{neg})$ as log-likelihoods for each of the models. Then, a discriminative setting of HMM is achieved by applying the rule:

$$\text{if } l(\lambda_{pos}) \geq l(\lambda_{neg}) \text{ for an instance } i \Rightarrow$$
$$\text{assign the positive class label to the instance } i.$$

As we discuss later, we exploit the models to form new, informative features rather than apply them as stand-alone discriminative models (see Section 3.2 for more details). One way to proceed is to use purely the log-likelihoods for each of the classes. However, our experiments suggest that use of log-odds ratios is a better choice (see Chapter 6):

$$r_{HMM}(x_m^{sec}) = \frac{\mathbb{L}(x_m^{sec}|\lambda_{pos})}{\mathbb{L}(x_m^{sec}|\lambda_{neg})} = l(x_m^{sec}|\lambda_{pos}|) - l(x_m^{sec}|\lambda_{neg}), \tag{2.3}$$

where x_m^{sec} is the m-th sequential feature.

2.3.2.2 Generative Long Short-Term Memory Recurrent Neural Network

Despite the heavy use specifically for sequential data, traditional HMMs have various limitations [Suts 13]. First, the hidden states should be discrete and due to the dynamic programming involved, the size of the hidden state space increases the computational time quadratically [Vite 67]. The most common use of HMMs is to model first order Markov processes, which makes it difficult to capture possible long-term dependencies of the data. Using HMMs of higher order also increases the hidden state space, which becomes infeasible for an efficient computation [Grav 13]. Though different versions of HMMs are introduced to tackle the problems to some extent, another class of models called *Recurrent Neural Networks* (RNNs) that is also specifically designed for sequences naturally provides the solution. RNNs also have a layer of hidden nodes, but this layer is capable of representing a higher number of distinct states thanks to distributed state representation. Previously, HMMs were either preferred over RNNs or they were equally considered, mostly as RNNs had other limitations [Lipt 15]. One of them, for example, is the problem of vanishing/exploding gradients, which occurs due to the effect of multiplying gradients multiple times. An enhancement of the RNN structure, called *Long Short-Term Memory unit* (LSTM) [Hoch 97], solved this problem, while the development of optimization methods and further research of deep learning allowed to train RNN models efficiently. The core idea of LSTM lies in replacing a usual artificial neuron by a more complicated structure, called *an LSTM unit.* This structure makes an LSTM network capable of learning long-term dependencies. LSTM has gained popularity by showing state of the art performance in many fields [Grav 13, Suts 14, Sriv 15, Lang 14]. Next, we introduce the general idea of RNN and explain the particular extension to the LSTM version.

Basics of artificial neural nets. All artificial neural networks consist of *units* or nodes (see Figure 2.11). A unit takes an input $\mathbf{x}$, computes the weighted sum of the input and passes it further to the *activation function* φ producing non-linear output. There are different choices of activation functions, where the most commonly used activation functions are *sigmoid* $\sigma(z) = \frac{1}{1+e^{-z}}$ and *tanh*: $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. In recent works it was shown that using the rectified linear unit (ReLU): $\xi(z) = \max(0, z)$, instead of sigmoid and tanh, can often improve the performance of the network [Kriz 12].

In general, the variety of different neural networks depends on the structure that combines units together. The common way to represent *the architecture* of a network is using nodes and edges between them. The network consist of *layers* of units, where the first is the *input layer*, last is the *output layer* and any number of *hidden layers* in-between. Intuitively, the artifi-

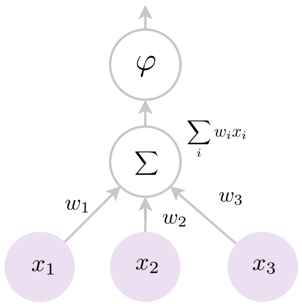


Figure 2.11: An example of an artificial unit with 3 inputs x_i, weighted sum $\sum$ and an activation function ϕ.

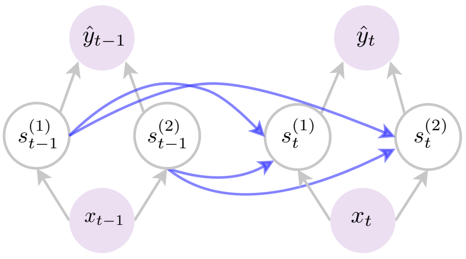


Figure 2.12: An example of a simple RNN spanning two timestamps t that has one hidden layer with two hidden nodes s_t.

cial network with some unit architecture is activated with an input data, and activations are propagated through the network until the output $\hat{y}$ is produced. The training of neural networks consists of iterative updating of the weights $\mathbf{w}$ to minimize some loss function $L(\hat{y}, y)$. The most common algorithm for the training is *backpropagation* [Rume 88]. The idea of the backpropagation is to use forward and backward passes, where during the forward pass the weights w and the loss function L are calculated and during the backward pass the partial derivatives with respect to each parameter are calculated using the chain rule. Intuitively, such an approach shows the error contribution of each parameter. The weights are then adjusted by gradient descent.

RNNs have the architecture that introduces the notion of temporal dynamics by allowing the cycles between nodes. In the “classical” version of RNN (see Figure 2.12) the edges connect the adjacent timestamps. At the time step t the hidden nodes $\mathbf{s_t}$ at that timestamp receive the information from the input $\mathbf{x_t}$ as well as from the hidden nodes $\mathbf{s_{t-1}}$of the previous timestep $(t-1)$. This leads to some sort of information sharing in the sequential data.

RNN can have any type of the classification task from the sequence classification discussed in this work with many-to-one connections, but also many-to-many, where the input sequences can be trained to predict the output sequences of any length. This is particularly useful in language modeling, where translation from one language to another may not have the input and output sentences of equal length.

LSTM. The vanishing/exploding gradient problem caused the initial RNN to perform poorly when training longer dependencies in sequential data. It occurs as the weights are updated proportionally to the gradient of the loss function, and propagating the gradients through a few layers of the network and across the time results in exponentially reduced (or increased) updates. Therefore, the input many steps ago may not affect the output at the end of the sequence. The solution to the problem was introduced by enhancing the regular artificial unit (see Figure 2.11) with the *LSTM unit* depicted in Figure 2.13. We describe the LSTM unit by adopting the notions of

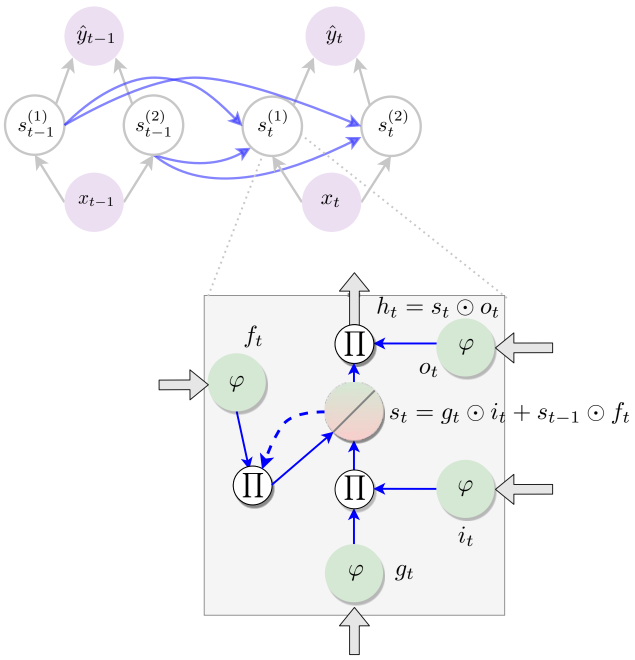


Figure 2.13: LSTM unit. Figure adopted from [Lipt 15]. Dashed edge shows the recurrent edge to itself from step t to t-1

Lipton et al [Lipt 15]. The input data at time $t (x_t)$ is fed to *the input node* g_t. The input node takes also the information from the hidden layer at the previous time step $h_{(t-1)}$ as it is depicted in Figure 2.12. Then, either tanh or sigmoid σ is applied as the activation function on the sum of weighted output. The other gates of LSTM units are *input gate* i_t, *forget gate* f_t and *output gate* o_t that proceed in the exact same manner. More precisely they are defined as follows:

$$g_t = \varphi(W_{gx}x_t + W_{gh}h_{(t-1)} + b_g)$$

$$i_t = \varphi(W_{ix}x_t + W_{ih}h_{(t-1)} + b_i)$$
$$f_t = \varphi(W_{fx}x_t + W_{fh}h_{(t-1)} + b_f)$$
$$o_t = \varphi(W_{ox}x_t + W_{oh}h_{(t-1)} + b_o)$$

where W_{kj} is the weight matrix from the layer j to the layer k and b_k is the bias term. The names of the gates are given according to the way they are combined. In *the internal state* s_t:

$$s_t = g_t \odot i_t + s_{t-1} \odot f_t$$

the outputs of input node and the input gate are elementwise multiplied. Intuitively, the input gate controls how much information should be allowed further. If the value of the input gate is zero, the information is considered to be irrelevant for the task. The forget gate is similarly used to decide how much information to forget from the previous internal state. The internal state sums them together. It is important to notice that one of the solution to the vanishing/exploding gradient is introduced here by the constant weight across the different time steps [Lipt 15]. Even with long time span the errors are passed further with a constant weight. The resulting value of the hidden LSTM unit output is calculated as:

$$h_t = \varphi(s_t) \odot o_t$$

The output gate o_t regulates how much information to let out of the node.

Summarizing, an LSTM unit manages the information flow in the forward pass by regulating with the gates how much information to let in and out and how much to forget. In terms of the backward pass it regulates how much error to propagate back.

Extraction of features from LSTM In order to be able to combine static and sequential data for hybrid models as described in Section 3.2 we are also interested in the extraction of the information from the LSTM network in a form of a feature. One way to proceed is to follow the same principle as in the HMM case (see Section 2.3.2.1). In other words we can use some notion of accuracy like mean squared error (MSE) between the predicted sample and the true sequence for continuous values of a sequence and define the new feature for each sequential feature x_m^{sec} as

$$r_{LSTM}(x_m^{sec}) = \log(\frac{MSE_{\text{NEG}}}{MSE_{\text{POS}}}) \tag{2.4}$$

where MSE_{POS} and MSE_{NEG} are the mean squared errors between the true output sequence and the generated sequence.

Note that compared to Equation 2.3 we use inverse *MSE ratio*. It is mere a simple convenience — while *MSE* is an error, the inverse ratio shows how likely it is that a sample comes from the generative model for positive cases rather than from a generative LSTM model for negative cases. We assume that the smaller the error, more likely the new sample belongs to the corresponding model. Also, the MSE is defined for the continuous values of a sequence, but it is straightforward to use another measure of accuracy in the case of discrete values.

In order to clarify the procedure, we demonstrate how the MSE ratio produced on a simplified example on a set of sequences in Table 2.4.

Table 2.4: Toy sequences for MSE ratio calculation

seq_nr	**sequence**	**class**
1	(2.1, 4.8, 6.6, 2.9)	1
2	(5.4, 5.1, 6.6, 4.8)	1
3	(5.5, 2.6, 2.1, 5.3)	0
4	(2.0, 6.2, 5.9, 5.3)	0
5	(2.0, 5.4, 5.6, 4.3)	?

First four sequences are used for the training and last is left for the prediction. We train a separate LSTM model on sequences 1 and 2 ($LSTM_{\text{POS}}$)

and an LSTM on negative samples 3 and 4 ($LSTM_{\text{NEG}}$). Once the models are trained, we can use them to generate a new sequence number-by-number and calculate the MSE ratios. It is an iterative process, where we give each LSTM as an input the value of a sequence for the first timepoint and predict the next symbol. For example, for the 5th sequence we provide as an input a value 2.0 and produce the next value by $LSTM_{\text{POS}}$ and $LSTM_{\text{NEG}}$, on the next iteration we provide $(2.0, 5.4)$ as an input for both models. We continue to do so until the end of the sequence. Next, we calculate the MSE for each of the models. For the sake of example, let LSTM_{POS} produced for the 5th sequence $(3.0, 5.0, 5.9, 4.7)$, while $LSTM_{\text{NEG}}$ gave as a result $(5.3, 7.4, 6.6, 2.0)$. The corresponding MSE ratio according to the Equation 2.4 would be: $log(5.295/0.3525) = 1.17$. This is the values of MSE ratio for the 5th sequence; similarly, we obtain MSE values for each of the unlabeled sequences. Later, we explain how the ratios are used to enrich the feature set in the hybrid models (see Chapter 3 for more details). Note that the "ground truth" that we compare with is not the label, but the sequence itself.

2.3.2.3 Other methods of feature extraction from generative models.

There are several other possibilities in addition to the one outlined in sections 2.3.2.1 and 2.3.2.2 that can reflect the information in time series.

Extraction of activations from LSTM. In case of LSTM there is a possibility to use the activations of the last hidden layer of LSTM at the last iteration as feature representation that an RNN model had defined and use the activations as new features [Yosi 14]. The log-likelihood ratios are almost the most aggregated form of information about the data samples and predictions of a model are the highest level of aggregation. LSTM activations $\mathbf{s_t}$ in this hierarchy can be considered as less aggregated ones, meaning that they contain more information about the input. Later we will explore how and when such new features from the LSTM network can help to improve the result of the classification (see Chapter 6).

Fisher score. In our case studies we use log-odds ratios from the generative models that correspond to posterior probabilities to belong to one class or another. One interesting line of research is to use the Fisher scores as suggested in the work of Jaakkola et al. [Jaak 00]. The idea is to fit HMM (or other generative model) for each of the classes and then, for each new sequence to compute the derivative of the log-likelihood with respect to the model parameters such as transition and emission matrices (in case of HMM). The motivation behind such representation is that the HMM can output the same score for different sequences, while the representation through the gradient provides a natural way to compare two sequences with respect to the same model. More precisely, Fisher score for sequence x is defined as:

$$U_x = \nabla_\theta \log P(x|c, \theta)$$

where θ are model parameters and $c \in C$ is a c-th class. For further information how to use Fisher score for the classification see Section 3.2.

2.4 Ensemble learning

In this work we often operate with the concept of *ensemble learning.* In this section we take a closer look what ensemble learning is and why it is so powerful. The concept of ensemble learning is very simple: multiple classifiers can be trained for the same task and then combined together for the final prediction. Sometimes ensemble learners are referred to as *multiple classifier systems.* Figure 2.14 depicts the high-level architecture of an ensemble learner. Ensembles consist of a set of *baselearners.* If the baselearners are of same type (e.g. decision tree) an ensemble is *homogeneous*, while in case of different types of baselearners (e.g. combination of a decision tree, a logistic regression and a neural network) it is called *heterogeneous ensemble.* In general, ensembles can generalize better than the baselearners. It was shown that 1) the prediction of an ensemble is more accurate than the prediction of a single best classifier and 2) the accuracy of weak classifiers can be boosted to the level of accuracy of the strong ones. Moreover, it is suggested that the powerful ensemble is the one which consists of diverse and accurate base learners [Zhou 12]. Currently, ensembles are considered to be a common approach in solving practical machine learning tasks that

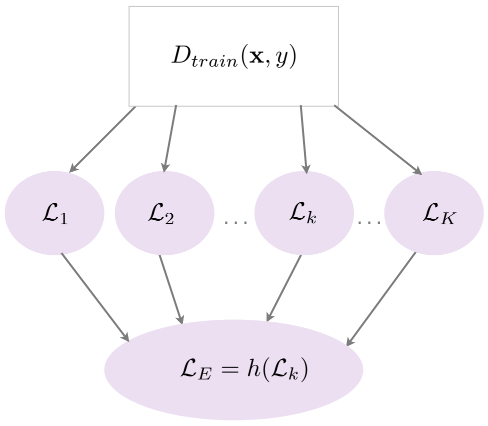

$D_{train}(\mathbf{x}, y)$
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_k$
$\mathcal{L}_K$
$\mathcal{L}_E = h(\mathcal{L}_k)$

Figure 2.14: The general representation of ensemble learning: multiple baselearners $\mathcal{L}$ are fitted on the train data or their subset and then combined into one ensemble learner $\mathcal{L}_E$ by using simple average, major voting or another learner.

aim at higher accuracy and do not require much of interpretability. One of such examples is the Kaggle competitions [Gold 10], where the winner is chosen according to the highest score and where even a small gain in the prediction accuracy can be crucial. Majority of the winners in the recent years use ensembles with hundreds of baselearners [Wind 14].

As there are no limits to how the classifiers are combined, there are many ensemble versions and architectures. The taxonomy of ensemble types is not well-defined. Various authors propose three, four, or even eighteen basic schemes for ensembles [Sewe 08]. We describe here briefly *bagging*, *boosting* and *stacking*, where the first is from the group of *sequential* ensemble methods, the boosting is representing the *parallel ensemble* family, while the stacking demonstrates the idea of more complex combination methods for the baselearners rather than just average or major voting.

Before going into details about these particular examples of ensembles we introduce simple combining techniques, where instead of second-level learning some aggregation function $h(\cdot)$ is applied to combine predictions of baselearners (see Figure 3.2a). Several popular aggregation functions $h(\cdot)$ are often used for the classification task:

- *indicator aggregation function* is defined as:

$$\mathbb{I}(\mathcal{L}_k) = \begin{cases} \mathcal{L}_k & \text{if } k = T \\ 0 & \text{if } k < T \end{cases},$$

  where T is a current timepoint. Indicator aggregation function is a special case of ensembles. In other words, it is a *gating function* that selects which baselearner to choose based on current time step.

- *averaging* is the most popular combination method, where the class probabilities from the different models are combined. Let us define a baselearner as $\mathcal{L}_k$, $k = 1 \dots K$ with the predictions of such a baselearner being $\hat{y}_{\mathcal{L}_k}$. Then,the metalearner prediction is:

$$\hat{y}_{\mathcal{L}_E} = h(\mathcal{L}_k) = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_{\mathcal{L}_k}$$

- *majority voting* is an aggregation method that we briefly mentioned when describing Random Forest in 2.3.1.2. Given a sample, each base classifier $\mathcal{L}_k$ votes for a particular class label for that sample. The final class label is the one that gets the majority of the votes. Sometimes *the rejection option* can be given, when there is a tie or in the case of multilabel classification a winner class does not receive enough (above a threshold) votes. The rejection option allows to introduce some notion of uncertainty for the predictions, when a sample can be left unclassified if the model is not "certain".

Next we describe briefly the core ideas behind the common ensembles:

Boosting is sometimes described as a version of models' weighted average. More precisely, in boosting base classifiers (often weak ones) are built incrementally, such that each next base classifier's focus is on the previously misclassified training instances. The final prediction is made by using a combination of the weighted base classifiers. The classical example of boosting is AdaBoost [Freu 95].

Bagging is short for Bootstrap Aggregation. Random Forest described in Section 2.3.1.2 is the most famous example of that ensemble type. In bagging baselearners are applied on different "versions" of a dataset, where each version is sampled from the original dataset with replacement. The final prediction is made by averaging in case of regression or by major voting in case of classification. Random Forest is in this sense an extension of bagging, where it also uses random selection of the features for each node split of each baselearner — a decision tree in this case.

Stacking is usually referred to as a more general procedure of baselearner combination, where instead of combining predictions of the baselearners in simple fashion with weighted averaging or major voting, the outputs of baselearners are *trained* again. In other words, it is a multiple step procedure, where firstly the baselearners are trained. Next, the output of first-level baselearners are handled as an input for the second-level learner or *a metalearner*. Often it is beneficial for stacking to be a homogeneous ensemble within the first level and heterogeneous – between the levels. Later, in our case study we compare stacking architecture of ensemble with the proposed hybrid approach (see Section 3.2). In Figure 3.2a we demonstrate the essential steps of the ensemble algorithm.

It is worth highlighting a few key differences and the consequent recommendations. Boosting is aimed to decrease bias of a classifier by exploiting the dependence between the baselearners (the performance is boosted using the residual decrease), while bagging decreases variance by exploitation of the independence between the baselearners. According to Kuncheva [Kunc 03], weaker, simpler learners with large training sample sizes are preferred for the boosting and more complex, but unstable and weak ones are better for bagging. According to Breiman [Brei 01], as bagging operates on samples with replacement, the overlap of the instances in such samples is somewhere around 63.2.%[1] On one hand, it allows one to estimate the performance during the training on *out-of-bag* instances, e.g. on other 36.8%,

[1]It is expected that for the i-th instance being sampled with replacement (where the sample size m is equal to the size of the original dataset) the probability distribution of being selected 1,2,... and more times is approximately Poisson ($\sim Pois(\lambda)$), therefore, the probability of i-th instance occurring at least once is $1 - (1/e) \approx 0.632$.

but on the other hand baselearners still have a large overlap in the datasets they are using, and, therefore, using more stable baselearners in bagging does not improve, in theory, the final classifier (bagging is not working with the kNN as the base classifier [Zhou 12]). Unstable classifiers such as unpruned trees are best suited for such an ensemble.

It is suggested by Ting and Witten [Ting 99] that the class probabilities as an input to metalearner are more valuable than the predicted class as they provide more information. However, we argue later that in the case of mixed data types the information extracted from the sequences is better served as a feature combined with other static features rather than the independent model prediction (see Chapter 6). In Section 3.2 we suggest a hybrid model that can be seen as a more complex ensemble architecture.

The notion of ensembles is important in the context of this work for multiple reasons: 1) in case of discriminative approach we use Random Forest that is ensemble in its nature, 2) this powerful structure is compared against the proposed hybrid approach and 3) last but not the least, we propose an ensemble scheme for *early classification* in the progressive index-based framework (see 3.4).

Chapter 3

Hybrid models and progressive index-based framework

Previously we discussed what are the common feature types and what are considered to be the data with mixed data types — when the data comes in a different form from the static feature matrix and violates the notion of independence. In Section 2.2 we list various data types in the case studies that are challenging for the machine learning flow. How can we handle them to keep as much information as possible in the final feature representation? In this chapter we first focus on several common ways of handling mixed data types, then propose a hybrid approach that boosts the performance of a classifier in certain scenarios by combining sequential and static data together (see for details Section 3.2), and last, we outline the progressive index-based framework that allows one to perform early classification by taking advantage of hybrids and ensembles.

3.1 Feature extraction from mixed data types

In this section we outline different ways of feature extraction from both sequential and graph data types that are used later for the case studies.

3.1.1 Feature extraction from sequences

When the desired information comes both in static and sequential forms, the possible next step is to pre-process it in some way that makes it possible to apply machine learning pipeline. Recall that *static* features correspond to standard vector representation for classifiers and *sequential* features are features that violate notion of i.i.d (see Section 2.2).

Sequential-to-static transformation. In case of such mixed data types as static and sequential data, the most straightforward approach is to represent sequential features as static ones. The sequences are either aggregated or transformed to static representation and can serve as the input along with other static features to the machine learning model. Next, we list some of the sequential features transformations. The goal is not to exhaustively list all the possibilities, but rather demonstrate some basic principles of transformations. Most of the methods serve as baselines for the later empirical research.

Consider a single **univariate symbolic sequence** (or simple symbolic sequence) for i-th instance $\mathbf{x}_i^{seq}$ with T timepoints. Note that sequences can be of various length for different instances. We can translate a symbolic sequence $\mathbf{x}_i^{seq}$ into a feature vector by summarizing information according to a predefined vocabulary Ω. Thus, the size of the resulted feature vector is the size of the vocabulary: $\|\Omega\|$ with each feature corresponding to a particular letter in the vocabulary. Depending on the aggregation function that we apply, we get the following representations:

- *Boolean representation* is the simplest form of such aggregation, where each feature is expressed through the indicator function. In other words, if the letter Ω_i is presented in the symbolic sequence, the binary feature corresponding to this word is one, otherwise zero.

- *Frequency representation* encodes a bit more information: instead of recording the existence or absence of a letter in the vocabulary, we count them. Note that in both of these cases the size of a feature vector can be very large which results in sparse matrix $\mathbf{X}$. The frequency-based approach also suffers from the heavy-tails of count distributions, where some letters are largely underrepresented. The assumption of both methods is that the vocabulary should be specified in advance.

- *Index representation* is an approach, where we take into account the information about the order in which the elements occur in the sequence. Each resulting feature corresponds to a position in the sequence with the value of the element in that position. The size of the feature vector is thus the length of the longest sequence.

- *n-gram representation* is a type of aggregation. It can be encoded in terms of boolean or frequency representations, but the size of vocabulary is increased as well as the feature space. Essentially, instead of features corresponding to a single element in the vocabulary, we take n-size tuples of elements. The elements of boolean and frequency representations mentioned above can be considered as *unigram* features, while *bigrams* would correspond to all unique pairs of vocabulary elements.

Consider an example with the vocabulary $\Omega = \{A, B, C\}$ and one single sequence $x^{seq} = \{ABB\}$. Then:

	A	B	C
Boolean representation	1	1	0
Frequency representation	1	2	0
	1st	2nd	3rd
Index representation	A	B	B

For 2-gram representation the same example would be:

	AA	AB	AC	BA	BB	BC	CA	CB	CC
Boolean repres.	0	1	0	0	1	0	0	0	0
Frequency repres.	0	1	0	0	1	0	0	0	0

In case of **complex symbolic sequences** (see Section 2.2), the vocabulary consists of vectors of symbols, not single symbols. Complex symbolic sequences are *multivariate sequences.* Such data object can be represented via *tensor*, or n-dimensional matrix, where one dimension is samples, one – features and the last dimension is timepoints. We can apply the same approaches as for simple symbolic sequences, – boolean, frequency, index or ngram representations – but for each of the sequences separately. The resulting feature matrix is therefore equal to the size of the corresponding simple representation multiplied by number of features. For example, for the boolean representation it is the size of the vocabulary of each feature multiplied by the number of features. This way, the resulting matrix can be used for the machine learning pipeline. In order to obtain index representation for multivariate sequences, *time spatialization* is applied. Let M_d be the number of sequential features in a data sample, each of them of length T_d. In this case, every sample can be represented by a matrix of size $M_d \times T_d$:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1T_d} \\ x_{21} & x_{22} & \cdots & x_{2T_d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M_1} & x_{M_d2} & \cdots & x_{M_dT_d} \end{bmatrix}$$

By flattening the matrix we obtain a feature vector of length $M_d \cdot T_d$ thus transforming a data sample from the time domain to the feature space:

$$(x_{11}, x_{12}, \dots, x_{1T_d}, x_{21}, x_{22}, \dots, x_{2T_d}, \dots, x_{M_d1}, x_{M_d2}, \dots, x_{M_dT_d}). \tag{3.1}$$

In this work we discriminate between symbolic sequences that consist of ordered observations of discrete values – symbols, and **time series** – sequences with continuous observations. One way to project time series into a feature vector is to *aggregate* them. For time series the simple statistics as mean and standard deviation can serve as static features, but many other methods exist [Lin 12, Lin 07]. Next, we mention a few approaches that are later applied in the case studies.

The *discretization*, or division of continuous values into pre-defined bins is the basic way to project continuous values onto discrete scale. Division into bins on its own has different versions: one common method is to divide time series into K equal intervals or into bins with equal frequencies. Inevitably, such transformation leads to a discretization error [Kots 06]. Once the time series are discretized, they take a symbolic representation and all methods that were mentioned previously for sequences can be applied. Here, methods for univariate time series can be easily extended to multivariate features. However, the problem remains — we lose information not only by discretization, but also as we do not account for a time-dependent information.

Time-frequency transformation is another approach for discretization, where time series are mapped from time domain into frequency domain. Some of the most common methods include Fourier transform and wavelet transform [Chui 92]. For example, recall the ECoG data time series (see Figure 2.7). In Figure 3.1 we depict time series of only one channel and its corresponding discrete Fourier transformation into 50 frequency components. Once such transformation is applied, Fourier components can be incorporated into the classification model as regular discrete features.

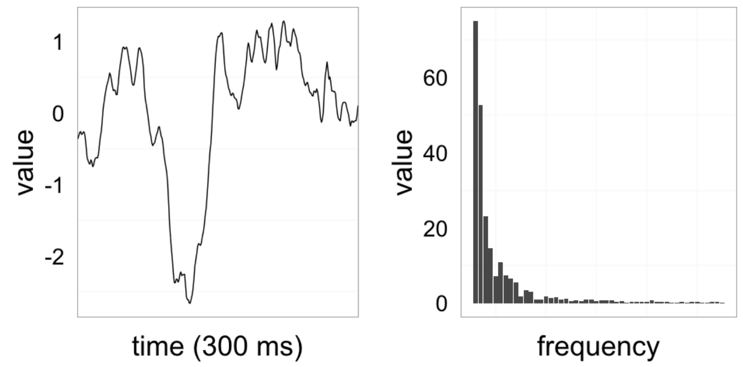


Figure 3.1

Static-to-sequential transformation. Another possible way to proceed in order to take mixed data types into account is to apply models that work on sequential data (like generative ones in a discriminative setting, see Section 2.3.2) to both static and sequential data simultaneously. In that case static features should be transformed into sequential ones. Static features can be *padded* with constant values, in other words we construct sequences of constant values and feed them to a sequential classifier along with the sequential features. For example, if a sequence of sample i consists of three observations, a static feature for i-th sample would be represented as the single value repeated three times.

To conclude, there are ways to map static features to sequential or vice versa. However, in both cases we do not take into account either dynamic nature of the sequential features or we misrepresent the static information.

3.1.2 Feature extraction from graphs

Previously we discussed how to deal with the combination of sequential features together with static feature vectors. In our *fraud detection case study* (see Chapter 5) both sequential and graph data types are present. We need to exploit the structure of a graph in order to obtain additional information. There are two types of graph information: local and global, where the local is obtained by extracting features manually from the ego-centric graph, while the global features use the information from the whole network. Next, we describe two global graph features that are used in the case study.

PageRank The PageRank algorithm is widely used for ranking web pages based on web link structures [Page 99]. Pages with high PageRank scores are usually authoritative pages, with either many incoming links or links from other important pages. Pages with low PageRank scores are usually of low importance or spam pages. Recently, PageRank has also been used for identifying spammers on social graphs [Chir 05, Huan 13]. In that work, the PageRank algorithm is run on a reversed email graph. More specifically, if user A sends an email to user B, one places an edge from B to A in the reversed email graph. A spammer that sends many spam emails but receives very few emails will have a high number of incoming edges in the reserve email graph, hence will likely have a high PageRank score.

In our work, we adopt a similar approach and compute PageRank scores on the reversed Skype user contact graph. More specifically, each user represents a node in the graph. If user u_i sends a friend request to user u_j, we place a link from user u_j to user u_i. Thus, users with higher PageRank scores are likely to be those that send out a large number of friend requests. On such a reversed contact graph, we assign an initial uniform score to each of the users and then perform iterative PageRank computation until the PageRank scores converge.

In each iteration, each user u propagates her scores to neighbors (friends). At the end of the iteration, a user u's new PageRank score $R_{u,i+1}$ is computed as:

$$R_{u,i+1} = 1 - d + d \sum_{\{X: e_{Xu} \in E\}} \frac{R_{X,i}}{outdegree(X)}$$

where d is the damping factor usually set to be 0.85 [Brin 12], $R_{X,i}$ is the score of the user X after the previous iteration, and $\{X : e_{Xu} \in E\}$ is the set of users in the graph having directed edges pointing to u (friends who received contact requests from u).

The local clustering coefficient Another input to our classifier that comes from the full contact graph is the *local clustering coefficient* [Watt 98]. The local clustering coefficient is the ratio of the number of connections in the neighborhood of a user to the number of connections in a fully connected neighborhood. Intuitively, it is a measure of how tightly the neighborhood of the user is connected. In terms of the Skype network, each user's contacts constitute her neighborhood. If the neighborhood is fully connected, the clustering coefficient is 1; on the other hand, if the clustering coefficient is close to 0, there are no connections between contacts of the user.

More formally, let k_i be the size of the neighborhood of user u_i. Let n_i be the number of directed links between those k_i users. Then, the clustering coefficient $cc(u_i)$ of user u_i is defined as:

$$cc(u_i) = \begin{cases} 0 & \text{if } k_i < 2 \\ n_i/(k_i(k_i-1)) & \text{otherwise} \end{cases}$$

It was demonstrated that clustering coefficient is greater in social networks than in a random network [Watt 98].

Next, we discuss more advanced ways of static and sequential data combinations that can potentially extract more information, suggest the hybrid approach and compare it with the ensembles.

3.2 Hybrid models

In the previous section we discussed how the dataset with mixed data types such as sequences and regular static features can be represented as a proper input for a classifier. The main goal is to extract information from both static and sequential types of data so that it contributes to the classification power. In this section we describe a more advanced approach to this problem, where discriminative classifier with static features is enriched with the new features from generative models trained on the sequential data. We refer to such approach as *hybrid*. By combining two different models, we achieve a higher accuracy than *singleton* models as singleton models discard or summarize the information to a great extent. We propose the hybrid approach and discuss its similarities and differences with other combination methods.

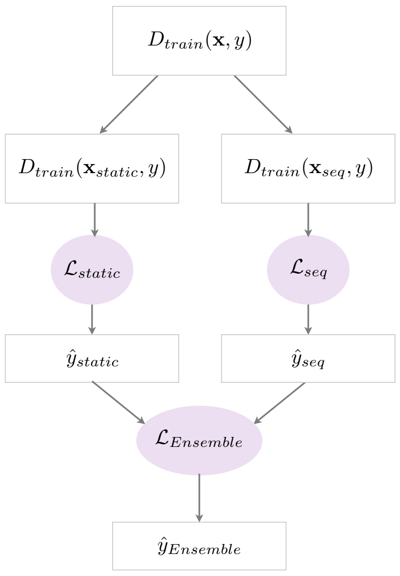
It has become common knowledge that a single classification algorithm even of high complexity is outperformed by many classifications combined together [Diet 00]. The base classifiers of a multilayered classification system should have enough diversity in order to exploit different patterns. When the data consist of different data types like static and sequential features, it leads already to a desired diverse information. However, it is still very common to transform such mixed data to a single data type and apply the regular classifier. We suggest a hybrid scheme for combination of data types, which uses generative models within a discriminative pipeline. Moreover, in hybrid models instead of using compressed sequence information represented as a prediction score (compared to ensembles in Figure 3.2a) we rely on an intermediate step of generative models.

In Figure 3.2b we depict the schematic pipeline of the hybrid approach. On a high level of abstraction, the provided data are divided according to the type of features into a subset of data with static features and another subset of either time series or symbolic sequences. Next, we use generative models to learn a representation of the sequential features, where we train models separately for each of the classification classes, i.e. use generative models in a discriminative manner. We mentioned in Section 2.3.2 the benefit of generative models for sequential data, which is to capture additional signal from the available data.

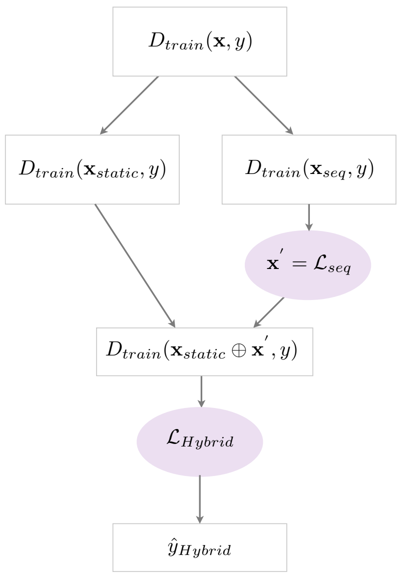
Once we fit generative models for each of the classes, we calculate for each instance the likelihoods that it belongs to each of the class-specific generative models. We use this information as new features $\mathbf{x}^{'}$ that describe the instance. New feature matrix $\mathbf{x}^{'} \oplus \mathbf{x}$ is created, which becomes a combination of the static features and the new features. This resulting feature matrix is passed to a new classifier to be trained.

However, the likelihoods from the class-specific generative models are not the only possibility to obtain new features. We discuss activations of LSTM model as potential candidates for this role in Chapter 6. In this thesis for a hybrid approach we experimentally analyze two types of the new features $\mathbf{x}'$: in case of HMM we use log-odds ratios (see Equation 2.3), and in case of LSTM we explore LSTM MSE ratios (as outlined in equation 2.4) and LSTM activations. Intuitively, log-odds and MSE ratios as the new features $\mathbf{x}'$ show for each instance to which class it belongs more likely. The interpretation of LSTM activations is less intuitive in this case, but potentially may provide valuable insights.

In Figure 3.2 we compare hybrids to ensembles. Note the key differences between the two methods: in an ensemble the metalearner uses predictions of baselearners as the new features, in other words in ensembles baselearners *cooperate* — the sequential and static models are trained for a weights to be assigned to predictions of these models. Hybrid models represent the cascading type [Heit 09] — static features are combined together with new features of sequential models.



(a) A schematic representation of an ensemble


$D_{train}(\mathbf{x}, y)$
$D_{train}(\mathbf{x}_{static}, y)$
$D_{train}(\mathbf{x}_{seq}, y)$
$\mathbf{x}' = \mathcal{L}_{seq}$
$D_{train}(\mathbf{x}_{static} \oplus \mathbf{x}', y)$
$\mathcal{L}_{Hybrid}$
$\hat{y}_{Hybrid}$

(b) A schematic representation of a hybrid approach

Figure 3.2

Ensembles are very powerful methods, but in some situations hybrids have more flexibility. Consider a toy example, where the underlaying process of data generation consists of two features, one static – $\mathbf{x_1}$ and one sequential – $\mathbf{x_2}$. The class of the instance is determined by the following rules:

$$\mathbf{x_1} \geq 7 \;\&\; \mathbf{x_2} = \mathsf{ARMA}(p = 1, q = 1) \Rightarrow \mathsf{class}_0$$

$$\mathbf{x_1} \geq 7 \;\&\; \mathbf{x_2} = \mathsf{ARMA}(p = 2, q = 2) \Rightarrow \mathsf{class}_0$$

$$\mathbf{x_1} < 7 \;\&\; \mathbf{x_2} = \mathsf{ARMA}(p = 1, q = 1) \Rightarrow \mathsf{class}_1$$

$$\mathbf{x_1} < 7 \;\&\; \mathbf{x_2} = \mathsf{ARMA}(p = 2, q = 2) \Rightarrow \mathsf{class}_0,$$

where ARMA stands for Autoregressive Moving Average model with p autoregressive terms and q moving average terms. For the comparison of two methods we use the same baselearners for the ensemble and hybrid methods. For the sake of brevity we also simplify the baselearners to provide a binary output. If we are to classify instances:

Table 3.1

$\mathbf{x_1}$	$\mathbf{x_2}$	class
5	ARMA(1,1)	1
5	ARMA(2,2)	0
8	ARMA(1,1)	0
8	ARMA(2,2)	0

a baselearner for the static features has problems classifying two first instances and therefore, ambiguous. Now, recall that the metalearner of the ensemble uses the predictions of two methods, while in the case of the hybrid approach we use initial static features in conjunction with the output of the sequential model. In Table 3.2 we can observe that for the ensemble first two cases can have either first or second class, which leads to the uncertain classification results and would depend on an applied threshold for the static baselearner rule. If first sample happens to be classified as 0, then in the new feature space, samples 1 and 3 are identical, thought they have different classes. For the hybrid model the input is not ambiguous anymore and can be learned easily. Despite being overly simplistic, this toy example demonstrates that the ensemble provides less flexible combination of baselearners.

Table 3.2: Input for the metalearner in case of ensemble (left) and hybrid (right) models.

Ensemble			Hybrid		
static pred	seq pred	real	static feat	seq pred	real
0/1	1	1	5	1	1
0/1	0	0	5	0	0
0	1	0	8	1	0
0	0	0	8	0	0

Not only ensembles provide ways to incorporate different models. Various algorithms have been developed to combine different machine learning algorithms [Kunc 04, Diet 00, Zhou 12]. There are a few other concepts that are similar to the suggested hybrid approach. We already mentioned multiple times one of the most prominent type: an ensemble of classifiers (see Section 2.4). Others include *mixture of experts* and *cascading classifiers.* Another interesting direction is *kernel methods.* Next, we provide an overview of the literature in this field and discuss the similarities and differences with our approach.

Ensemble methods. The main difference between general ensemble and hybrid is that in the case of ensembles we operate at the level of models' predictions, where metaclassifier combines the predictions with different weights. In the case of ensemble base methods learn to *cooperate* [Kunc 04, Zhou 12]. In the hybrid approach we operate at the level of features, where valuable information in a generative model is extracted and passed to the discriminative learner along with the static features. In case of ensembles one can tune only the contribution of (in most cases) multivariate models in the final ensemble, while in the suggested hybrid approach generative models contribute to the prediction along with other features.

Mixture of experts. A prominent method for classifiers' combination is the mixture of experts (ME) [Nowl 90, Jaco 91]. The biggest difference with the ensembles is the ability to *specialize* on each case instead of the *cooperative ability* like in ensembles. Intuitively, mixture of experts consists of base classifiers, each of them specializing on a separate region, and a separate *gating function* that "decides" which classifier should be applied to which region. The gate function is also learned along with the base classifiers. Figure 3.3 shows a schematic representation of mixture of experts.

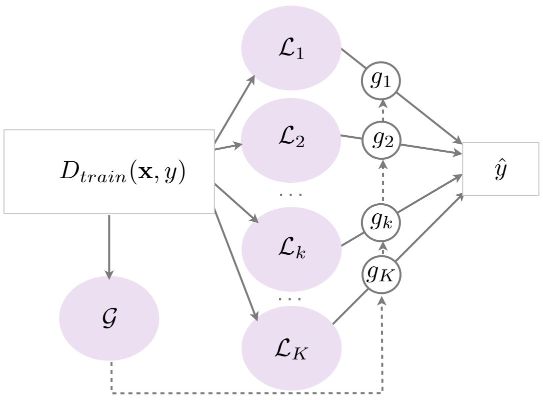


Figure 3.3: Mixture of experts. $\mathcal{G}$ is a gating function, $\mathcal{L}_k$ are experts/base classifiers

In the most conventional ME both gating function and experts are linear classifiers. However, later other non-linear functions were proposed that gained a better performance with the most prominent examples of neural networks [Maso 14]. Moreover, each of the experts, in turn, can be a mixture, which gives rise to a *hierarchical ME* type [Murp 12].

The suggested hybrid approach is different from ME in various ways. It transforms predictions to new features and does not operate on the level of predictions nor have a trained gate network. However, the way ME uses each model to the region that the model knows the best, the hybrid approach applies specific models to data types they excel at the most.

Cascade of classifiers. Another combination technique, which resembles the hybrid approach is *a cascade of classifiers* [Viol 01, Rayk 10, Negr 08]. The basic idea is to use classifiers sequentially. Each classifier either passes the instance to the next, increasingly complex classifier, or rejects it immediately if some threshold is reached (see Figure 3.4). The first classifiers are either less complex or trained with simpler features. With each new classifier increases the complexity or/and features' calculation cost. Such systems are especially useful when the data is imbalanced and the focus is on the detection of minority class. One prominent example is a boosted cascade of classifiers [Viol 01], where AdaBoost is used in a sequential manner to rapidly perform face recognition. Interestingly so, cascades of classifiers were applied mostly in the context of image recognition and scene understanding in particular [Viol 01, Li 10, Heit 09].

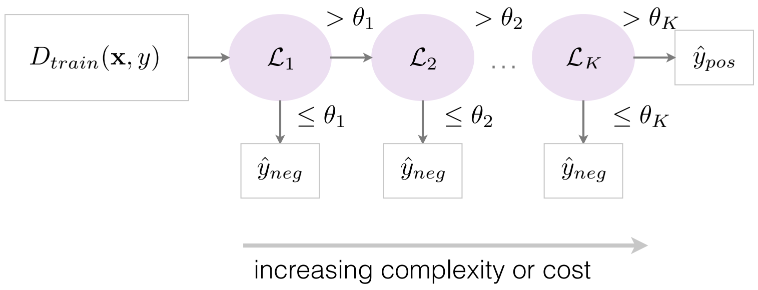


Figure 3.4: A cascade of classifiers

The similarity with the hybrid approach lies in the sequential way of classifiers' combination. However, in the hybrid approach we do not increase the complexity of the classifiers, but apply different types of models: generative models are used for sequences and passed in a cascading manner to the discriminative classifier. The idea of the combination of generative and discriminative classifiers as a cascading model is introduced in work of Negri and colleagues [Negr 08] with the application to vehicle detection. However, their work differs from ours in many ways. They deal with the images and extract features using rectangular filters along with the histograms of oriented gradient. Essentially, all of the data is static. Moreover, in terms of the approach we do not reject any instances in advance.

Other hybrids. A work similar to ours is introduced by Lester et al. [Lest 05], where the main goal is to recognize human activities. Their idea is to first select the useful static features with boosting similarly to the work of

Viola et al. [Viol 01], use an ensemble of static classifiers (classifiers that deal with static features) and apply Hidden Markov Model in order to smooth activities and capture temporal dynamics. Altogether, their method is very similar to the one proposed in this work with the difference of the order. In our case we extract features from a general model and plug them into a discriminative classifier, while in the work of Lester et al. discriminative ensemble is used to find a small set of static features that is then used as inputs to the HMM. This difference is justified by the different task at hand. In their case they want to produce a generative model of activities, while in our case we are interested in the classification task for mixed data types.

Kernel methods from generative models. There is a well-studied line of research that investigates *kernel methods* that are derived from probabilistic generative models [Jaak 99, Murp 12, Jaak 00, Gart 03]. Despite defining the appropriate kernel function is not straightforward, there are several methods that suggest valid kernel functions for the sequential data. Kernels provide a way to use generative model for the classification task and are especially useful when the length of the sequences varies. Let us first define a *kernel function*, which is relevant to the notion of similarity [Sonn 05]. It is defined on the set X as: $K : X \times X \to \mathbb{R}$ such that for all $x, z \in X$

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle, \tag{3.2}$$

where ϕ is a mapping from X to an (inner product) feature space F, $\phi : x \to \phi(x) \in F$. The use of kernel methods is motivated by the Cover's theorem, which states that given a set of training data that is not linearly separable, it can be transformed into a training set that is linearly separable by transforming it into a higher-dimensional space using some non-linear transformation [Cove 65]. Informally, the so-called *kernel trick* allows to compute a function K without the need of explicitly computing the function ϕ. Once the appropriate kernel function is defined (which satisfies the Mercer's condition [Merc 09]), any kernel method can be applied to any type of data [Da S 10].

Various kernels have been suggested for the sequential data from general-purpose kernels such as string kernels and k-spectrum kernels [Xing 10] to domain-based kernels, like the Independent Domain kernel (IDK) based on

Pfam database [Alla 08, Ben 03]. However, the most relevant kernels in the context of this work are *Fisher kernels* [Jaak 99, Jaak 00] that are based on HMMs. Simply put, the key idea of Fisher kernel is to use the gradient of log-likelihood with respect to the parameters of the HMM as the features in a discriminative classifier [Gart 03]. In paragraph 2.3.2.3 we already defined a Fisher score U_x that extracts a feature vector from a generative model. It is trivial to enrich the static feature vector of Fisher scores with other static features and pass it to the discriminative classifier. At this stage, this idea fits into our suggested hybrid approach, where the Fisher scores can be seen as new features x' (see Figure 3.2b). In our approach we use log-odds as such features produced by a generative classifiers, but the Fisher scores can be also applied. It is an interesting direction that is left for the future work.

However, if we want to kernalize the approach, we can define various kernels on the vector of Fisher scores and pass it, for example, to SVM as it is done in the work of Jaakkola et al. [Jaak 00]. But in principle, any other generative-discriminative pairs of models can be used the same way, for example an RNNs with random forest.

The commonly known Fisher kernel is defined as $k(x,x^{'}) = U_x^T I^{-1} U_{x^{'}}$, where x and $x^{'}$ are two sequences. If we use the Euclidean distance function between the Fisher scores as:

$$D^2(x,x^{'}) = \frac{1}{2}(U_x - U_{x^{'}})^T F^{-1}(U_x - U_{x^{'}})$$

where F is the Fisher information matrix, we can use a Gaussian kernel

$$k(x,x^{'}) = e^{-D^2(x,x^{'})}.$$

In our work we did not use Fisher scores nor Fisher kernels. Fisher scores can be used with other static features and thus, incorporated in the proposed hybrid approach naturally; Fisher kernels for the kernalized methods where both static features and Fisher scores are present is another possible approach, however, it does not fit to the suggested hybrid framework.

Our results demonstrate that the suggested in this work hybrid approach with its differences from previous works is beneficial especially with the state of the art generative models such as LSTM, where the model plays the role of a generative feature extractor (see the empirical investigations in Chapter 6). Nonetheless, the incorporation of the Fisher scores in the hybrid approach as well as the line of research towards Fisher kernels can be a fascinating future work.

3.3 Notion of earliness

Previously we described the hybrid models that handle sequential data along with the static features for a classification task. By default it is assumed that the final decision for the class is made at the end of the complete sequences. In other words, it is a *whole-sequence based classifier.* It requires to observe all the information in advance before the classification is made. However, often an early classification of an ongoing process is highly desirable especially in time-sensitive domains. In such a case it is especially necessary to extract all the information both from static and sequential parts of the data as the amount of data is limited. In this chapter we introduce a framework that provides a more flexible usage of classifications for mixed data types, where the prediction can be made at early stages. We discuss the aspect of earliness in the classification task and how it is handled in the proposed framework.

The notion of earliness can be expressed as follows: in some classification tasks the earlier we make the predictions about the outcome, the more valuable it is under the constraint of the sufficient accuracy. Multiple examples of such tasks can be found: one of the examples is the prediction of the diagnosis from the medical history of a patient, where the disease progresses over time. The earlier a medical practitioner is able to diagnose a disease the higher are chances to cure the patient. Many methods for sequential data classification do not account for the earliness aspect by extracting features from the full sequences; and constructing classifiers that are capable to optimize for earliness by maintaining the level of accuracy is not straightforward [Xing 11]. There are several studies that investigate the problem of time series early prediction. Most of the works define some threshold over the accuracy that triggers the final decision [Hata 13, Xing 12, Anto 15]. In essence, there is the trade-off between earliness and accuracy — by postponing the decision the accuracy increases. In work of Xing [Xing 11], instead of threshold, the special features, shapelets, are extracted and the classification is performed when some of the highly discriminative shapelets are discovered. In some works the notion of the confidence/reliability is also introduced, which shows the amount of uncertainty in the prediction [Parr 13, Anto 15]. Another direction of the research is to investigate the optimal t, where the prediction should be made. It is defined as the optimal point between the misclassification cost and the delay cost and it is adaptive [Dach 15, Tave 16]. The work of Hatami and Chira, [Hata 13] suggest to use the rejection option, where the ensembles of classifiers should agree above a certain threshold or the decision should be postponed. Xing et al. [Xing 12] introduces the method based on nearest neighbors that accounts for stability of the predictions. It is worth mentioning that most of the works investigate univariate time series. In paper of He and colleagues [He 15], early prediction is discussed in the context of multivariate time series. Also, the proposed procedure of the progressive index-based system is similar to the work of Ishiguro [Ishi 00], where they apply each weak classification algorithm for each time step. However, they explore the problem of multi-label classification specifically, while we focus on the combination of static and sequential features. Moreover, the *progressive index-based framework* we introduce in the next section can also handle multi-label classification naturally. Our work provides enough flexibility to use a threshold measure as the accuracy is recorded at any timepoint. We do not specify in advance when the optimal decision should be made and it is left for future work. The suggested future direction would be to add earliness as an additional hyperparameter and to estimate particular timepoints. In Chapter 4 we experimentally show on the PBPM case studies that most of the predictions can be made at very early stages.

3.4 Progressive index-based framework

In this section we propose a progressive index-based framework that has several improvements over the existing approaches: it handles multivariate sequential data (1), it naturally incorporates the earliness of the prediction (2), as well as handles sequences of different length (3). Moreover, it takes into account both static and sequential features (4). Also, we are not limited to the type of algorithms used in baselearners and the metalearners.

The suggested approach has two phases: learning phase and application phase:

- (Initialization) Define a form of a baselearner $\mathcal{L}_t$ and a metalearner $\mathcal{L}_E$ or a form of an aggregation function $h(\mathcal{L}_1, \dots \mathcal{L}_t)$.
- (Learning phase)
  1. Collect static features $\mathbf{x}_{static}$ of a dataset $\mathcal{D}$.
  2. Align sequential features $\mathbf{x}^{seq}$ of a dataset $\mathcal{D}$ so that they have a startpoint $t = 1$ and an endpoint $t = T$. In case of complex sequences handle each of the sequential features separately by flattening the matrix (see Section 3.1 and Equation 3.1).
  3. For each t train a separate baselearner $\mathcal{L}_t$ by combining static features with the sequential ones up to a fixed timepoint t. See Sections 3.1 and 3.2 to see how static and sequential features can be combined using different encodings or the hybrid approach.
- (Application phase) For each new running case $\mathbf{x_i}$ in $\mathcal{D}_{test}$ and either for each timepoint t or for the latest available t use the metalearner $\mathcal{L}_E$ to obtain a prediction $\hat{y}_t$ as depicted in Figure 3.5.

The above framework is very flexible as it allows one to choose various ways to handle the problem of the classifier with the combination of multivariate sequences and static features as it consists of different building blocks that we defined in the preceding sections. We experimentally confirm the framework's usefulness and demonstrate the improvements over existing approaches in the following chapters, where we apply it on the case studies.

In Section 3.2 we discuss how to choose a baselearner for the framework and in the following chapters we experimentally show how to apply index-based method or combine it with more sophisticated sequential methods like HMM and LSTM. For the results and the discussion see Chapter 6, where we present the larger study of different combination methods. The metalearner aggregation functions are listed in Section 2.4. In Chapter 4.1 we show how this framework is fully realized on a particular example from the BPM domain and discuss different modifications.

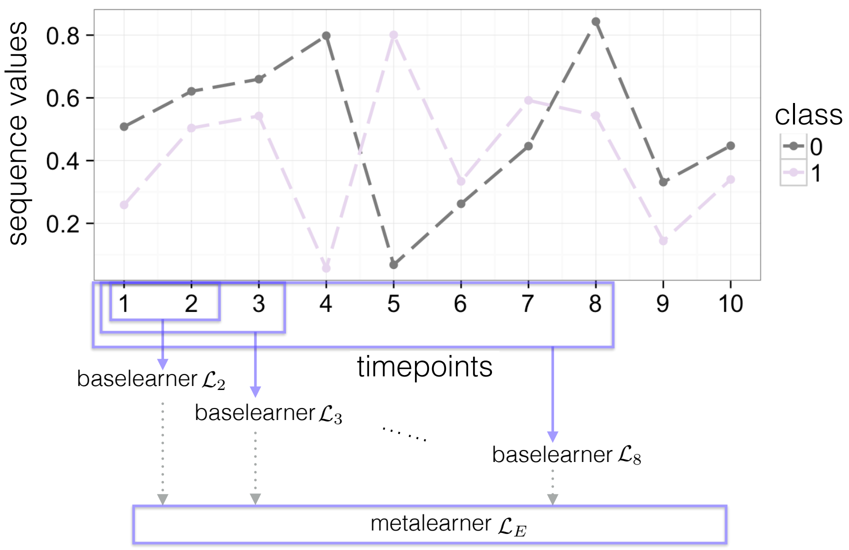

sequence values
0.8
0.6
0.4
0.2
class
0
1
1
2
3
4
5
6
7
8
9
10
timepoints
baselearner $\mathcal{L}_2$
baselearner $\mathcal{L}_3$
baselearner $\mathcal{L}_8$
metalearner $\mathcal{L}_E$

Figure 3.5: The illustration of the progressive framework. Sequences for 10 timepoints are depicted. Colors indicate the classes of the two sequences. We construct multiple baselearners, where each subsequent baselarner uses increased number of timepoints. The final metalearner uses the predictions from all the baselearners.

Example. Let us outline a hypothetical example, where the progressive index-based framework demonstrates its capabilities. Consider a dataset of the same nature as discussed in 2.1.4 *ECoG case study*, where we analyze electroencephalography data from 64 electrodes. For each of the trials there exists a label: whether a patient imagines to move either a tongue or a finger. Moreover, imagine that we have at our disposal the gender and the age of a patient. Now let us initialize a framework:

- (Initialization) We choose a baselearner to be a hybrid model, which handles sequential and static features as discussed in Section 3.2. In a hybrid model we capture the sequential features with the LSTM recurrent neural network as in Section 2.3.2.2 and calculate LSTM odds-ratios (see Section 2.3.2.2 and Equation 2.4 in particular). We train Random Forest on the set of static features concatenated with these odds-ratios (Section 2.3.1.2). Once the baselearner is initialized, we choose the metalearner to be logistic regression. That is $\mathcal{L}_t = \mathsf{hybrid}(\mathsf{LSTM},\ \mathsf{RF})$, $\mathcal{L}_E = \mathsf{logit}(\mathcal{L}_1, \ldots, \mathcal{L}_t)$.

- (Learning phase, step 1: static feature extraction) Let us describe the procedure of extracting static and sequential feature sets more precisely. Firstly, we partition the recording of 3 seconds into a 300 ms overlapping intervals with the sliding window. Thus, the first interval would be a signal from 0 to 300 ms, while the second one from 100 to 400 ms and so on. We end up with 10584 such samples. For each sample we perform a Fourier transformation as shown in Figure 3.1. We pick first 100 frequency components from each piece. Let us denote them $\boldsymbol{x_s} = \{f_1, ... f_{100}\}$. Therefore, the $\boldsymbol{x'_s} = \{f_1, \ldots f_{100}, \text{age}, \text{gender}\}$ is a matrix of 10584×102.

- (Learning phase, step 2: feature matrix) The sequential feature set is the initial raw time series divided into 300 ms overlapping pieces for each of the 64 channels, which results in a 3D matrix $\boldsymbol{x_d}$ of size $10584 \times 300 \times 64$ that can be passed to a generative LSTM directly.

- (Learning phase, step 3: hybrid model) For each of the timepoints t we need to train a baselearner — the hybrid model. In our case the earliest prediction we can make is at $t = 300$ and then we make predictions in 100 ms steps as we use overlapping intervals. For each such t we fit LSTM on a subset of $\boldsymbol{x_d}$ for positive cases and for negative cases separately. 64 new features denoted as $\text{hyb}_1, \ldots \text{hyb}_{64}$ are obtained using $\text{hyb}_i = \log(\text{MSE}_{neg}) - \log(\text{MSE}_{pos})$ for each of the channels. After that, Random Forest is applied on the resulting feature matrix of size 10584×166.

- (Application phase) For each baselearner prediction $\hat{y}_{\mathcal{L}_t}$ at timepoint t we obtain a final prediction using logistic regression. Consider a timepoint at 500 ms. We learn a new classifier, logistic regression, with the baselearner predictions from $\mathcal{L}_{[0,300]}$, $\mathcal{L}_{[100,400]}$ and $\mathcal{L}_{[200,500]}$, where logistic regression estimates and assigns some weights for each of the classifiers (see Figure 3.5).

Note that we do not discuss the proper splitting strategy in this example. See section 6.4.1 for one possible way of cross-validation for the progressive index-based framework.

Chapter 4

Case study I: PBPM

4.1 Introduction

In this section we describe the case study of the predictive business process monitoring problem (PBPM). Recall the description of the classification task from the description 2.1.4. The general idea can be summarized as the ability to predict the deviance probability of a process as early as possible. Existing approaches to the problem of PBPM [Magg 14, Conf 15] essentially map the problem to that of early sequence classification [Xing 10], where a classifier is trained over the set of prefixes of historical traces. This classifier is used at runtime in order to predict the outcome of an ongoing case based on its current (incomplete) trace. A key step is to extract features from prefixes of historical traces. In this respect, existing approaches treat traces as simple symbolic sequences, sequences of symbols, each representing an event but without its payload. Sometime only the payload of data attributes attached to the latest event is included in the feature vector of the classifier, but the evolution of data attributes as the case unfolds is ignored.

In this chapter we investigate an alternative approach where the traces are treated as complex symbolic sequences, that is, sequences of events each carrying a data payload consisting of attribute-value pairs. As we deal with a non-trivial data type, it is crucial to choose how to encode a complex symbolic sequence in terms of feature vectors.

We propose the progressive index-based framework discussed in Chapter 3 and apply it in the BPM domain, as well as define different *encodings* of sequences — feature vector representations of business processes. Namely, we suggest and compare two different baselearners $\mathcal{L}$ and compare them with several baselines. The first baselearner corresponds to index representation of a sequence (see Section 3.1). Using this encoding, we specify for each position in the case the event occurring in that position and the value of each data attribute in that position. The second encoding corresponds to a hybrid baselearner that is obtained by combining the index-based representation with discriminative Hidden Markov Models as described in details in Section 3.2.

The proposed methods are evaluated in terms of their accuracy at different points in a trace based on two real life logs: (i) a patient treatment log provided for the BPI challenge 2011 [Dong 11] and (ii) an insurance claim process log from an insurance company [Suri 13b]. The descriptions and types of these datasets are discussed in Section 2.2.3.1.

4.2 Related work

Next, we provide an overview of existing predictive business process monitoring approaches and where the suggested approach stands. We can broadly classify existing PBPM techniques based on i) sources of data that are taken into account ii) type of the predictions. In a first group of works [Aals 11, Aals 10], the authors present a set of approaches in which annotated transition systems, containing time information extracted from event logs, are used to: (i) check time conformance while cases are being executed, (ii) predict the remaining processing time of incomplete cases, and (iii) recommend appropriate activities to users working on these cases. In [Foli 12], an *ad-hoc* predictive clustering approach is presented, in which context-related execution scenarios are discovered and modeled through state-aware performance predictors. In [Rogg 13], the authors use stochastic Petri nets for predicting the remaining execution time of a process. In [Pola 14] the authors use both information from the control-flow and the payload, using such methods as Support Vector Regression (SVR) and Naive Bayes in order to predict the remaining time of the process. This group of works concentrates on a control-flow of the processes or the predictions are time-related.

A second group of works focuses on approaches that also use control-flow, but instead of predictions they provide recommendations in order to reduce task-specific risks. For example, in [Conf 15], the authors present a technique that helps the participants in making risk-informed decisions, with the aim of reducing the process risks. Risks are predicted by traversing decision trees generated from the logs of past process executions. In [Pika 13], the authors make predictions about time-related process risks, by identifying and exploiting indicators observable in event logs that highlight the possibility of transgressing deadlines. In [Suri 13a], an approach for Root Cause Analysis through classification algorithms is presented. Decision trees are used to retrieve the causes of overtime faults on a log enriched with information about delays, resources and workload.

An approach for prediction of abnormal termination of business processes is presented in [Kang 12]. Here, a fault detection algorithm (local outlier factor) is used to estimate the probability of a fault to occur. Alarms are provided for early notification of probable abnormal terminations. In [Cast 05], Castellanos et al. present a business operation management platform equipped with time series forecasting functionalities. This platform allows for predictions of metrics on running process instances as well as for predictions of aggregated metrics of future instances (e.g., the number of orders that will be placed next Monday).

PBPM focused on specific types of failures has also been applied to real case studies. For example, in [Metz 12, Feld 13], the authors present a technique for predicting “late show” events in transportation processes. In particular, they apply standard statistical techniques to find correlations between “late show” events and external variables related to weather conditions or road traffic.

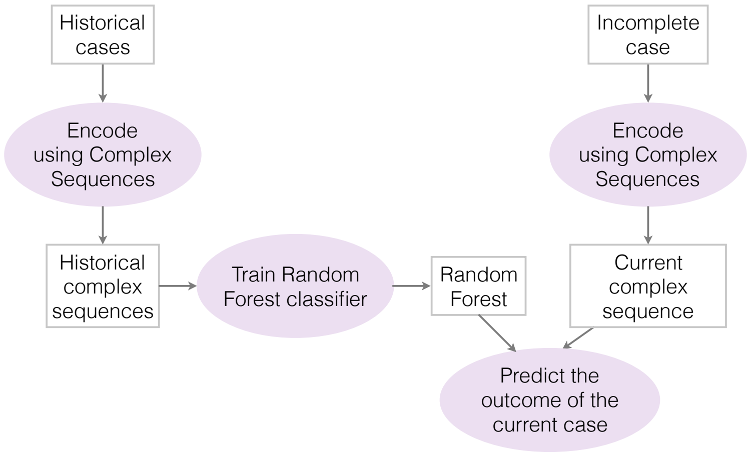
A key difference between these approaches and our technique is that they rely either on the control-flow or on the related data for making predictions, whereas we take both source of information into consideration. The similar approach has been considered in [Magg 14], where a framework for the PBPM of constraint fulfillment and violation has been proposed. In this approach, however, only the payload of the last executed event is taken into account, while neglecting the evolution of data values throughout the execution traces. The present case study aims at addressing the latter limitation of previous studies by treating the input traces as complex symbolic sequences.

It is worth noticing that a few further directions are already proposed and evaluated based on the results of the present work. Verenich at al. [Vere 15] apply clustering as an additional phase for the suggested progressive index-based framework in attempt to speed-up the algorithm results. The work of Teinemaa at al [Tein 16] focuses on the PBPM problem with the additional source of data — textual information. They analyze and propose solutions for handling such unstructured data type by extracting features using techniques from natural language processing. In [Tree 16] the *milestone approach* is compared to a progressive index-based framework. The idea is to discard the notion of time alltogether and instead to define milestones — significant events, where the aggregated information about the processes reflects the state of the process that is important for the prediction. This is especially useful when more granular data for each index t is not available or largely missing, or there is no precise knowledge in what order the events have happened.

4.3 Complex symbolic sequence encodings in the PBPM domain

In this section we define the progressive index-framework (see Chapter 3) for the BPM problem and discuss different choices of baselearners for that task.

Let us first provide a high-level overview of the predictive process in a a schematic way (Figure 4.1). To predict the outcome of an ongoing case, its current (incomplete) trace of length t is encoded using complex symbolic sequences (see Section 2.2.1). A complex symbolic sequence in the PBPM context carries information about the control flow and the data flow of the trace. The approach uses an assumption that a log of historical (completed) cases is available for the training purposes. From the available cases, all the information from the start till the index t is extracted and, in turn, encoded in the form of complex symbolic sequences. In addition, these sequences are labeled using a binary or categorical value according to their outcome — for example, if a case is deviant, it is labeled as 1, and 0 otherwise. The "historical complex symbolic sequences" are used to train a discriminative classifier. After the training phase is completed, the classifier is applied on a current ongoing trace and the most probable outcome is returned in terms of the probability or a score, which indicates the confidence of a classifier towards the real class of that case. We use Random Forest as a discriminative classifier. However, in order to justify our choice we made a comparison of the performances of a few discriminative algorithms when applied to one of the BPM datasets. The results of the comparison are shown in Figure 4.10.



Historical cases
Encode using Complex Sequences
Historical complex sequences
Train Random Forest classifier
Random Forest
Incomplete case
Encode using Complex Sequences
Current complex sequence
Predict the outcome of the current case

Figure 4.1: Overview of the proposed approach.

Recall that the case is represented in the form as depicted in Figure 2.5 (see subsection 2.2.3.1). As a running example, we consider the log in Figure 4.2 pertaining to a medical treatment process. Each case relates to a different patient and the corresponding sequence of events indicates the activities executed for a medical treatment of that patient. Let us define a sequence related to a business process as σ. In the example, *consultation* is the first event of sequence σ_1. Its data payload "$\{33, \text{radiotherapy}\}$" corresponds to the data associated to attributes *age* and *department*. Note that the value of *age* is static: it is the same for all the events in a case, while the value of *department* is different for every event. In the payload of an event, always the entire set of attributes available in the log is considered.

σ_1 (consultation{33, radiotherapy},...,ultrasound{33, nursing ward}):false

...

σ_k (order rate{56, general lab},..., payment{56, clinic}):true

Figure 4.2: Running example of the complex symbolic sequences.

In case for some event the value for a specific attribute is not available, the value *unknown* is specified for it.

The goal of predictive business process monitoring is outlined in Section 2.1.4. We are interested in automatically deriving a function f that, given an ongoing sequence σ_i provides a label for it, i.e., $f : (\mathbf{BPL}, \sigma_i) \rightarrow \{C\}$. It is important to note that we achieve that by training a discriminative classifier on *sequences of the same length* starting from $t = 0$ up to a pre-defined index t that are derived from historical cases in **BPL**. As we discussed earlier, the most crucial moment is to derive proper features. Therefore, in the current case study we represent each sequence σ_i, $i = 1...N$ as a feature vector $x_i = (x_{i1}, x_{i2}, ...x_{it})$, where $t \leq T$.

In the context of the PBPM task we investigate different encoding schemes and compare them with several baselines. The baselines are described in Section 3.1 and the two suggested approaches are the *index-based* and the *HMM-based* encodings. The latter is essentially the encoding that adopts the hybrid model for the classification and is described in more details in Section 3.2. When the traces are treated as simple symbolic sequences, the additional information related to data and data flow is neglected, while in the index-based and HMM-based encodings we express all the information through the multidimensional sequences and apply either a discriminative classifier or use the generative models as feature extractors to compliment the classifier.

Let us further consider the encodings described in Section 3.1 and apply it on a toy example 4.2 of a hospital log.

	consultation	ultrasound	...	payment	label
σ_1	1	1	...	0	false
...					
σ_k	0	0	...	1	true

(a) *boolean* encoding.

	consultation	ultrasound	...	payment	label
σ_1	2	1	...	0	false
...					
σ_k	0	0	...	4	true

(b) *frequency-based* encoding.

	event_1	...	event_t	label
σ_1	consultation		ultrasound	false
...				
σ_k	order rate		payment	true

(c) *simple index* encoding.

	age	event_1	...	event_t	...	department_last	label
σ_1	33	consultation		ultrasound	...	nursing ward	false
...							
σ_k	56	order rate		payment	...	clinic	true

(d) *index latest payload* encoding.

Figure 4.3: Baseline encodings for the example in Figure 4.2.

The first two baseline approaches in our experiments describe sequences of events as feature vectors, where each feature corresponds to an event class (an activity) from the log. Essentially, they take into account only control flow in a very aggregated form. These encodings correspond to such feature extraction methods as boolean and frequency representations (see Section 3.1 for the details). Tables 4.3a, 4.3b show the example from Figure 4.2 encoded using these representations.

Another way of encoding a sequence is by using index representation, where we take into account also information about the order in which events occur in the trace, as in the *simple index-based* encoding. Here, each feature corresponds to a position in the trace and the possible values for each feature are the event classes. By using this type of encoding the example in Figure 4.2 would be encoded as reported in Table 4.3c.

The fourth baseline encoding adds to the simple index baseline the data of the latest payload. Here, data attributes are treated as static features without taking into consideration their evolution over time. Table 4.3d shows this encoding for the example in Figure 4.2.

	age	event_1	...	event_t	...	dep_1	...	dep_t	label
σ_1	33	consultation		ultrasound		radiotherapy		nursing ward	false
...									
σ_j	56	order rate		payment		general lab		clinic	true

(a) *index-based* encoding.

	age	event_1	event_t	...	dep_1	...	dep_t	LLR_event	...	LLR_dep	label
σ_1	33	consultation	ultrasound		radiotherapy		nursing ward	0.12	...	0.56	false
...											
σ_j	56	order rate	payment		general lab		clinic	4.3	...	1.7	true

(b) *HMM-based* encoding.

Figure 4.4: Encodings for the example in Figure 4.2.

In the more complex form of index representation — *index-based* encoding, the data associated with the events in a sequence is divided into static and sequential information. The resulting feature vector x_i, for a sequence σ_i is:

$$x_i = (x_1^{static}, \ldots, x_{m_s}^{static}, e_{11}, \ldots e_{m_d t}, x_{11}^{seq} \ldots x_{m_d t}^{seq}),$$

where each $x_{m_s}^{static}$ is a m-th static feature, each $e_{m_d t}$ is the m_d-th event class at position (index) t and each $x_{m_d t}^{seq}$ is a m_d-th sequential feature associated to a corresponding event. The example in Figure 4.2 transformed into this encoding is shown in Table 4.4a.

In order to deal with complex symbolic sequences in the HMM encoding we adopt the notion of hybrid model (see Section 3.2). It is the index-based approach with the additional information that comes from a generative model in the form of features. For each sequence we compute the log-likelihood ratio (LLR) according to Equation 2.3. The resulting feature vector, therefore, consists of LLR values extracted from each of the sequential features, which is added to the feature vector obtained with the *index-based* encoding. In particular, the input vector for the classifier is, in this case:

$$x_i = (x_1^{static}, \dots, x_{m_s}^{static}, e_{11}, \dots e_{m_d t}, x_{11}^{seq} \dots x_{m_d t}^{seq}, LLR_1, ..LLR_{m_d}),$$

where each LLR_{m_d} is the log-likelihood ratio computed based on the simple symbolic sequence corresponding to either an event class or a sequential feature of the original case. Table 4.4b shows an encoding for the example in Figure 4.2 obtained by using log-likelihood ratio values.

4.4 Evaluation

In this section, we provide the description of the experiments that we carried out. Our evaluation focuses on the following research questions:

- **RQ1**: Do the proposed encodings provide *reliable* results in terms of predictions?
- **RQ2**: Do the proposed encodings provide reliable predictions at *early* stages of the running case?
- **RQ3**: Are the proposed encodings *stable* with respect to the quality of the results provided at different stages of the running case?

The three questions focus on three intertwined aspects. The first one relates to the quality of the results (in terms of prediction correctness) provided by the proposed encodings. The second one investigates how early the encodings are able to provide reliable results. The third one focuses on the stability of the quality of the results when computed at different stages of an ongoing case. In the following, we describe the experiments carried out to answer these research questions.

4.4.1 Datasets

The datasets for the particular case study are briefly described in the Section 2.2.3.1. The event log for the dataset $\mathbf{BPL_1}$ contains domain specific

LTL	# Positive cases	# Negative cases
φ_1	459	684
φ_2	894	249
φ_3	260	883
φ_4	320	823
γ_1	788	277

Table 4.1: Distribution of labels in the datasets.

attributes that are both case attributes and event attributes in addition to the standard XES attributes.[1] For example, *Age*, *Diagnosis*, and *Treatment code* are the case attributes (that we consider as static features) and *Activity code*, *Number of executions*, *Specialism code*, and *Group* are the event attributes (that we consider as sequential features). The second log $\mathbf{BPL_2}$ includes only event attributes like *Claim type*, *Claim reason*, and *Amount.*

We have defined 4 temporal constraints corresponding to the following linear temporal logic rules [Pnue 77] over event classes in dataset $\mathbf{BPL_1}$:

- $\varphi_1 = \mathbf{F}(\text{“}\textit{tumor marker CA} - 19.9\text{”}) \vee \mathbf{F}(\text{“}\textit{ca} - 125 \textit{ using meia}\text{”})$,
- $\varphi_2 = \mathbf{G}(\text{“}\textit{CEA} - \textit{tumor marker using meia}\text{”} \rightarrow \mathbf{F}(\text{“}\textit{squamous cell carcinoma using eia}\text{”}))$,
- $\varphi_3 = (\neg\text{“}\textit{histological examination}-\textit{biopsies nno}\text{”})\mathbf{U}(\text{“}\textit{squamous cell carcinoma using eia}\text{”})$,
- $\varphi_4 = \mathbf{F}(\text{“}\textit{histological examination} - \textit{big resectiep}\text{”})$.

and we have used them to label the cases in the training set from the dataset $\mathbf{BPL_1}$ as compliant or non-compliant (one labeling for each rule). Cases in the training set of dataset $\mathbf{BPL_2}$ have been labeled with respect to a constraint corresponding to a rule γ_1 formalizing a regulation internal to the insurance company. This rule requires a claimant to be informed with a certain frequency about the status of his or her claim. The distribution of labels in the datasets is shown in Table 4.1.

[1] XES (eXtensible Event Stream) is an XML-based standard for event logs proposed by the IEEE Task Force on Process Mining (www.xes-standard.org).

4.4.2 Evaluation procedure

In this case study we use as the main measure AUC (see Section 2.1.2). The measure we use to evaluate *the earliness of a prediction* is based on the number of events that are needed to achieve a minimum value for AUC. Finally, we use standard deviation to evaluate the stability of the results computed at different stages of an ongoing case.

In our experiments, we order the cases in the logs based on the time at which the first event of each case has occurred. Then, we split the logs in two parts. We have used the first part (80% of the cases) as a training set, i.e., we have used these cases as historical data. As we compare different versions of baselearners which require different splitting strategy, the training set is used differently in the experiments based on the different encodings. For most of them, the entire training set was used to train a Random Forest classifier. The only exception is the HMM-based encoding — it uses 75% of the training set for training the HMMs and 25% for training the Random Forest (later on we refer to it as *validation set*). For the application phase we use the remaining 20% of the cases.

A few hyperparameters has to be chosen for our task. For Random Forest classifier, number of trees is fixed to 500 and the optimal number of features to use for each tree (mtry) was estimated separately using 5-fold cross-validation on the training set (see Section 2.1.2). The optimal number of hidden states for HMMs is estimated in a similar way. In particular, the original training set is split, in turn, into training and validation cases and, using these cases, different parameter configurations are tested. The optimal ones – with highest AUC, were chosen for the experiments.

In order to measure the ability of the models to make accurate predictions at an early stage, we computed the AUC values using indexes ranging from 2 to 20. This choice is justified by the observation that for the defined labelings for φ_1-φ_4 and γ_1, encodings based on the sole control flow are able to provide correct predictions after about 20 events.

4.4.3 Results

Figures 4.5-4.8 show the trend of the AUC values when predicting the outcome of the cases in the test set from dataset $\mathbf{BPL_1}$, with respect to labeling rules φ_1-φ_4. In particular, each plot shows the evolution of the AUC values for the encodings under examination when using the first 20 indexes of each case in the test set. In Figure 4.5, we plot the AUC trend for predictions for the compliant rule φ_1.

For very early predictions the baseline based on the latest data payload gives an AUC that is comparable to the one obtained with complex symbolic sequences. However, for longer prefixes, the encodings based on complex symbolic sequences exploit the sequential information of the payload. Note that starting from prefixes of length 7 the AUC for both the encodings based on complex symbolic sequences is above 0.9.

Similar trends can be observed in Figures 4.6-4.7 where the case labeling is based on the compliant rules φ_2 and φ_3. In Figure 4.8, the case labeling based on the compliance with respect to the rule φ_4, the divergence of the AUC values of complex symbolic sequence encodings is more evident. Here, the HMM-based encoding slightly outperforms the one that considers only indexes.

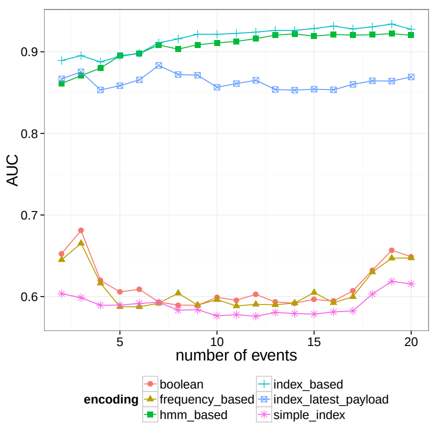


Figure 4.5: AUC values using prefixes of different lengths. Labeling based on compliance with respect to φ_1.

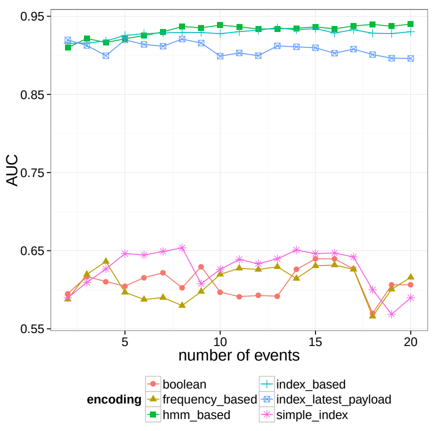


Figure 4.6: AUC values using prefixes of different lengths. Labeling based on compliance with respect to φ_2.

Figure 4.9 shows the AUC trend obtained for the case labeling based on the compliance according to γ_1 of cases in the $\mathbf{BPL_2}$. We can observe that also for this dataset, for early predictions the baseline encoding based on

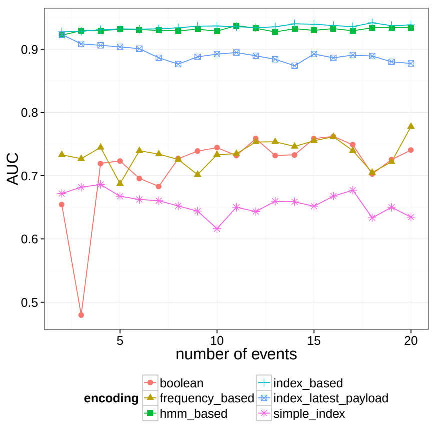


Figure 4.7: AUC values using prefixes of different lengths. Labeling based on compliance with respect to φ_3.

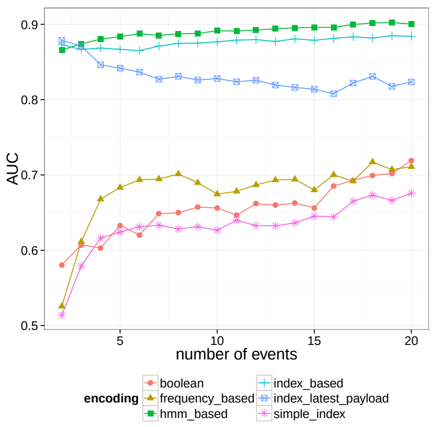


Figure 4.8: AUC values using prefixes of different lengths. Labeling based on compliance with respect to φ_4.

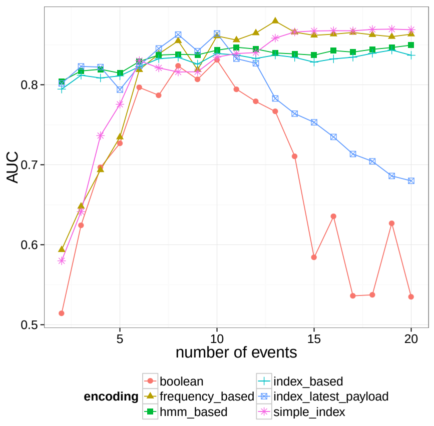


Figure 4.9: AUC values using prefixes of different lengths. Labeling based on compliance with respect to γ_1.

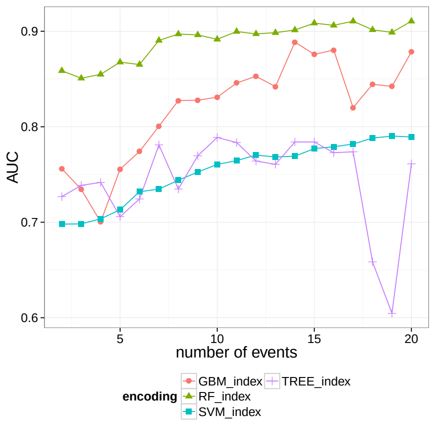


Figure 4.10: AUC values using different classification algorithms. Labeling based on compliance with respect to φ_1.

the latest data payload gives a good AUC, while the other baselines have a lower AUC. For slightly longer prefixes (between 6 and 13), the AUC values of all the baseline encodings is comparable with the encodings based on complex symbolic sequences. From prefixes of length 11 the AUC values

	mean across prefixes					st. deviation across prefixes				
encoding	φ_1	φ_2	φ_3	φ_4	γ_1	φ_1	φ_2	φ_3	φ_4	γ_1
boolean	0.614	0.610	0.714	0.655	0.690	0.027	**0.018**	0.063	0.036	0.111
frequency-based	0.609	0.610	0.735	0.679	**0.816**	0.025	0.021	0.022	0.043	0.084
simple index	0.590	0.627	0.656	0.631	**0.814**	**0.013**	0.025	**0.018**	0.036	0.080
index latest payload	**0.863**	**0.908**	**0.892**	**0.831**	0.787	**0.009**	**0.008**	**0.012**	**0.018**	0.060
index-based	**0.917**	**0.928**	**0.935**	**0.876**	**0.828**	**0.016**	**0.006**	**0.004**	**0.006**	**0.013**
HMM-based	**0.907**	**0.932**	**0.931**	**0.890**	**0.835**	**0.018**	**0.009**	**0.003**	**0.010**	**0.013**

Table 4.2: AUC trends. Bold values show the highest average AUC values (higher than 0.8) and the lowest AUC standard deviation values (lower than 0.02).

for the boolean encoding and for the one based on the latest data payload decrease again. This case study shows that, although baseline encodings can perform very well for certain prefix lengths, their performance is not stable. On the other hand, encodings based on complex symbolic sequences are able to provide a reasonable AUC (around 0.8 in this case) even for short prefixes and to keep it constant or slightly improve it for longer prefixes.

Summing up, the case studies show that the baseline based on the latest data payload and the encodings based on complex symbolic sequences provide, in general, reliable predictions. Table 4.2, reporting the average AUC values for all the encodings under the examination, confirms these results. However, while the baseline encoding is not always able to reach an average AUC value of 0.8, the two encodings based on complex symbolic sequences have an average AUC that is always higher than 0.82. Based on these results, we can, hence, positively answer **RQ1** that the proposed encodings provide *reliable* results in terms of predictions.

Our experimentation also highlights that some of the presented encodings are able to provide reliable predictions at a very early stage of an ongoing case. As shown in Table 4.3 (left), the baseline based on the latest data payload and the encodings based on complex symbolic sequences are able to provide an AUC higher than 0.8 in all the cases under examination at a very early stage of an ongoing case (starting from prefixes of length 2 in most of the cases). This is not the case for the other baseline encodings. The encodings based on complex symbolic sequences are also able in most

	min(prefix) for $AUC = 0.8$					**min(prefix)** for $AUC = 0.9$				
encoding	φ_1	φ_2	φ_3	φ_4	γ_1	φ_1	φ_2	φ_3	φ_4	γ_1
boolean	▨	▨	▨	▨	8	▨	▨	▨	▨	▨
frequency-based	▨	▨	▨	▨	6	▨	▨	▨	▨	▨
simple index	▨	▨	▨	▨	6	▨	▨	▨	▨	▨
index latest payload	2	2	2	2	2	▨	2	2	▨	▨
index-based	2	2	2	2	3	7	2	2	▨	▨
HMM-based	2	2	2	2	2	7	2	2	18	▨

Table 4.3: Min. number of events needed for an AUC > 0.8 (left) and > 0.9 (right).

of the cases to reach an AUC higher than 0.9, though not always at an early stage of an ongoing case. Both these encodings require 7 events for predicting the fulfillment of φ_1 at this level of accuracy. The HMM-based encoding is the only one able to predict the fulfillment of φ_4 with an AUC of 0.9 (after 18 events). Starting from these observations, we can positively answer **RQ2** that the proposed encodings provide reliable predictions at *early* stages of the running case.

Finally, the experiments highlight that some of the encodings have a trend that is more stable than others when making predictions at different stages of the ongoing cases. Table 4.2 shows that the encodings based on complex symbolic sequences have the most stable AUC trends (the standard deviation for AUC is lower than 0.02 in all the cases). This is not always true for the baseline encodings. We can then provide a positive answer to **RQ3** that the proposed encodings are *stable*.

4.5 Discussion

In Chapter 3 we introduce the concept of progressive index-based framework. The presented case study adopts this framework and puts it in a relevant to the field of BPM context. However, the general goal of the framework is to solve a problem of early predictive business process monitoring. Previously we compared different encodings that can be used to create feature representation for baselearners for the progressive index-based framework. In this section we discuss how the progressive indexed-based approach can be applied in order to provide more flexibility in terms of capturing sequential data. In this section we demonstrate that there are various options for the suggested encodings that can be used in the framework.

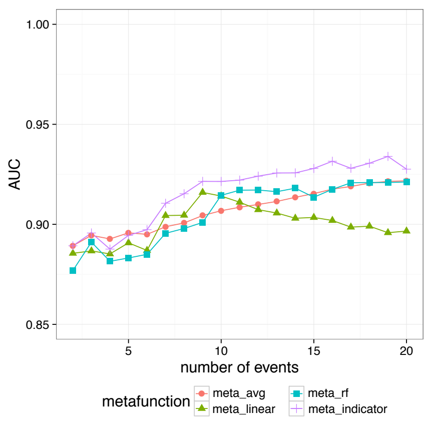
Consider an algorithm of progressive index-based framework outlined in Section 3.4. In the initialization stage we define the baselearners and the metalearners. In our experiments on the BPM data logs we compare different baselearners. Boolean, frequency and simple index-based baselines are various baselearners $\mathcal{L}_t$. More advanced baselearners are the index with latest payload, index-based and HMM-based encodings, where the latter corresponds to a hybrid model described in Section 3.2. For the purpose of baselearner aggregation $\mathcal{L}_E = h(\mathcal{L}_t)$ we use a simple indicator aggregation function. Namely, for each defined index t we train the baselearner $\mathcal{L}_t$ and use the gating function that chooses at $t = T$ baselearner $\mathcal{L}_{t=T}$ with weight 1 and assigns weight 0 to other baselarners $\mathcal{L}_{t \neq T}$. For example, if we want to predict whether the case is deviant only after 3 events, we pick the baselearner trained on all historical cases for 3 events only.

There is the flexibility not only to pick any baselearner (from the simple baselines mentioned above or advanced hybrids or ensembles), but also a metalearner (or an aggregation function) as well. It helps to redistribute the importance of the information contained in the baselearners trained on the previous timestamps. The schematic version of the metalearner for the progressive index-based framework is depicted in Figure 3.5.

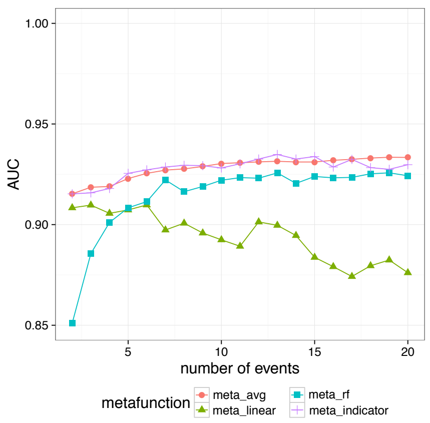
For example, in Figure 4.11 we show the results for dataset $\mathbf{BPL_1}$ with different LTL rules, where we explore different aggregation functions and metalearners and how they affect the classification accuracies (in AUC). Indicator and average aggregations are described in Section 2.4, while "rf" and "linear" correspond to two metalearners – Random Rorest and linear regression respectively, that are trained separately, on the next iteration.

The results of Figure 4.11 suggest that despite its simplicity the indicator aggregation performs either as well or slightly better than other aggregation functions almost for each index. There are a few possible explanations of this phenomenon. The first one is that in the progressive index-based framework we pick the model for index t that already accounts for the information about the previous events. It handles this information as features (not the separate models) and, therefore, the information from other models is not novel nor useful. Note that the difference between the models is not significant — AUC varies between the models less than 4 percent points. The second possible explanation of the simple metalearner being better than more complex ones may lie in the fact that more complex metalearners heavily overfit, and the results on a test set turn out to be worse.

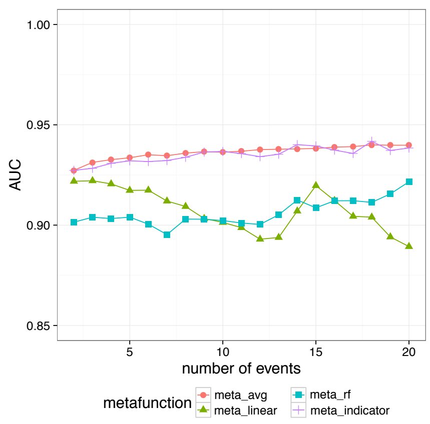
However, it is difficult to claim that the indicator aggregation is preferable over other aggregation functions or metalearners — it might be the case only for a particular case study. In some scenarios the information from the previous events that is captured via models rather than features can be useful. One example of metalearners being beneficial is when the control-flow information is poorly aligned. Consider an example, when the hypothetical sequence 1: `"A->B->A->C->D"` is essentially similar to a sequence 2: `"A->A->C->D"` and is same in terms of the outcome. The second event "B" of the first sequence is just a rare deviation from the more prevalent process of `"A->A->C->D"`. Starting from the 2nd index the index-based encoding takes into account a "shifted" information. If we use identity as an aggregation function, the model $\mathcal{L}_t$ might misclassify such a rare case. However, during the process of metalearner training, it might discover the pattern that the first case is properly classified by the earlier models, but misclassifies it after the "shifting". Such way, the metalearner applies a higher weight for the earlier models (models trained on earlier indexes) for the cases similar to the first sequence. Therefore, using indicator function as a metalearner we may misclassify if such a rare case appears in the test set (or in a real-time prediction), while the more complex metalearner would classify it correctly with the help of the classifiers $\mathcal{L}_{\leq t}$. These types of problems become less relevant with the increasing dataset size, but are expected to be more frequent in smaller historical sets of traces.



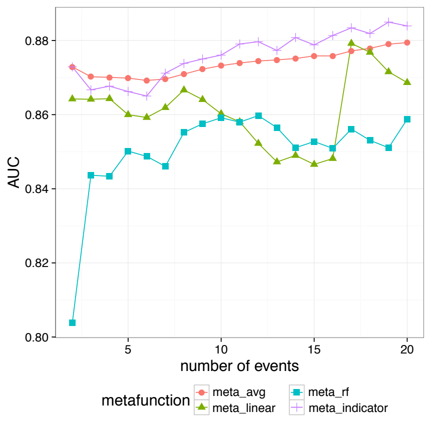
Dataset φ_1



Dataset φ_2



Dataset φ_3



Dataset φ_4

Figure 4.11: AUC values using prefixes of different lengths for different metalearners. Note that vertical axis does not start with zero.

Execution Times All experiments were conducted using R version 3.0.3 on a laptop with processor 2,6 GHz Intel Core i5 and 8 GB of RAM. Tables 4.4 and 4.5 show the average execution time (in seconds) and the standard deviation for the index-based and the HMM-based methods for different prefix lengths.

The execution times for constructing the classifiers (offline) is between 1.08 seconds and 186.41 seconds across all the experiments for the index-based encoding and between 0.99 and 186.41 seconds for the HMM-based encoding. Note that, in addition, the HMM-based encoding also requires

	HMM Training					RF Training				
	2	5	10	15	20	2	5	10	15	20
index-based avg						1.08	5.05	26.29	79.20	176.65
index-based s.d.						0.09	0.22	2.46	5.54	12.28
HMM-based avg	23.14	34.11	49.03	65.95	83.51	0.99	4.88	26.55	81.74	186.41
HMM-based s.d.	1.24	2.53	4.02	4.75	8.23	0.20	0.55	1.18	6.25	11.22

Table 4.4: Execution times per prefix length in seconds for training.

	Predictions				
	2	5	10	15	20
index-based avg	0.23	1.43	6.46	13.37	24.21
index-based s.d.	0.05	0.13	0.57	0.78	1.72
HMM-based avg	0.24	1.45	6.34	13.69	26.40
HMM-based s.d.	0.05	0.14	0.56	0.92	2.96

Table 4.5: Execution times per prefix length in seconds for predictions.

time for training the HMMs, ranging from 23.14 to 83.51 seconds. At runtime, time for making a prediction on a given prefix of a case is in the order of milliseconds for the runtime prediction on short cases (in the order of seconds for longer cases).

4.6 Conclusions

The work has put forward some potential benefits of approaching the problem of predictive business process monitoring using complex symbolic sequence encodings. The empirical evaluation has shown that the index-based encoding achieves higher reliability when making early predictions, compared to pure control-flow encodings or control-flow encodings with only the last snapshot of attribute values. The evaluation has also shown that encodings based on HMMs may add in some cases an additional margin of accuracy and reliability to the predictions, but not in a significant nor systematic manner. The further case studies indicated (see Chapters 5,6) that the benefit of using HMM-based extractor is more prominent in datasets with the higher amount of sequential data, longer sequences and stronger temporal dependency.

A threat to validity is that the evaluation is based on two BPM logs only. Although the logs are representative of real-life scenarios, the results may not generalize to other logs. In particular, the accuracy may be affected by the definition of *positive outcome.* For logs different from the ones used here and other notions of outcome, it is conceivable that the predictive power may be lower. A direction for future work is to evaluate the methods on a wider set of data logs so as to better understand the limitations of the suggested approaches in the domain of PBPM. Further in this thesis we investigate a similar approach applied in other domains that gives a broader overview of the models' performance.

It is worth mentioning that further research was done by other researchers in this field, and the results showed that in many cases index-based encoding performs well [Tree 16, Tein 16, Vere 15].

The methods considered in this work are focused on the problem of *intra-case PBPM*, where the aim is to predict the outcome of one individual ongoing case seen in isolation from others. A macro-level version of this problem is the *inter-case PBPM*, where the goal is to make predictions on the entire set of ongoing cases of a process, like, for example, predicting what percentage of ongoing cases will be delayed or end up in a negative outcome. Initial work on inter-case PBPM [Conf 15] has approached the problem using control-flow encodings plus the last snapshot of attribute values. An avenue for future work is to investigate the use of complex symbolic sequence encodings in this context.

Chapter 5

Case study II: Fraud detection

5.1 Introduction

In this chapter we demonstrate the application of the suggested approach (see Chapter 3) to the problem of fraud detection that was introduced as a second case study 2.1.4. Fraud detection has attracted considerable attention in both academia and industry [Pate 13, Cole 15, McDo 15]. Techniques for fraud detection rely on a wide variety of data, often tied to specific applications. Each application may in fact give rise to several different kinds of data, even more so as fraud schemes evolve over time. In addition, some of the data may be voluminous, incomplete, and not fully reliable. Therefore, one strategic element in fraud detection is the development of approaches for fusing disparate information sources, and for making sense of the aggregate information, robustly and at scale.

We propose and evaluate an approach to this problem using the suggested feature enrichment described in Section 3.2. It is based on supervised machine learning and combines information from diverse sources such as static user profiles, time series that represent user activities, and the results of algorithms that analyze user social connections. Separately, these sources can be insufficient for fraud detection. The data are often sparse, contain missing values, and the abnormal patterns associated with attacks may manifest themselves in different parts of the data. Our work explores a new way to fuse the data sources, synergistically, for the purpose of fraud detection.

We develop and study our approach in the context of Skype. See the definition of the problem in Section 2.1.4 and the description of the dataset along with the data types associated with it in Section 2.2.3.2.

The specificity of the previous case study lies in the encodings of a business log, where the goal is to find the deviant case as early as possible. The aim of the fraud detection task is to go beyond the present, sophisticated defenses, and to detect "stealthy" fraudulent users, namely, those that manage to fool those defenses for a relatively long period of time. Our concrete objective is to catch these stealthy fraudulent users within the first 4 months of activity. Therefore, in opposite to BPM case study discussed in Chapter 4, we do not investigate the earliness parameter, but rather fix time at some threshold. It is requested by the needs of the company and the existing framework at the moment.

Our results indicate that, with our methods, we are able to detect 68% of these users with a 5% false positive rate; and we are able to reduce by 2.3 times the number of these users active for over 10 months.

To this data, we apply techniques and algorithms for supervised learning, but only after non-trivial pre-processing. As will be seen, one of the challenges is relatively short time series to work with. In order to extract as much information as possible from them, we use the approach suggested in Section 2.3.2.1, where we use set of Hidden Markov Models that produce odds of fraudulent vs. normal activities (both in terms of service utilization and social activities). Similarly, we adapt techniques from work on social graphs [Huan 13] in order to estimate the reputation of users according to their social connections.

In summary, the contributions of this work are:

- an architecture and methods to process and fuse information from a variety of sources in order to identify fraudulent users;
- an evaluation of the efficacy of the methods on real data;
- the quantification of the impact of each one of the different sources of information for the task of detecting fraudulent users.

5.2 Related work

In this section we briefly review approaches and techniques related to our study. We concentrate on research in machine learning and in social network analysis, with the focus on work on fraud detection.

Skype allows users to communicate with each other via text messaging, audio, and video calls. It supports both free services as well as paid-for products and subscriptions. Skype is not a telecommunication provider but, in some ways, fraud in Skype resembles fraud in telecommunication services. Previous work (e.g., [Hila 08, Hila 09, Farv 11]) has extensively studied fraud in telecommunications, and in some cases has explored data mining and machine learning techniques [Fawc 97a, Ku 07, Wei 13]. However, these previous studies have mostly leveraged the static user profiles and usage features for detection. In our work, we consider features of product usage based on time series, which provide richer and more fine-grained information than static features of product usage represented by simple statistics (e.g., mean and standard deviations). In addition, we study a broader set of features, including local and global social features.

In Skype, users add each other in friend lists and employ multiple channels to communicate. Thus, the Skype communication graph can be viewed as a social network graph, to some extent, where nodes are Skype users and the edges between then correspond to the contact lists. Social network features were studied in the literature [Mcgl 09, Wang 08] as a tool for fraud detection. Their value motivates us to explore them within a general machine learning framework.

In order to fuse sequential features for usage and other static profile features (for both training and detection), we combined the use of Hidden Markov Models and the (log-odds) comparison to normal users and a classification framework based on Random Forest (see corresponding Sections 2.3.2.1, 2.3.1.2). A number of previous studies discuss the combination of different machine learning methods [Ho 94, Cao 95]. In particular, the general subject of classifier combinations has been considered and justified theoretically by Kittler and Hatef [Kitt 98]. Furthermore, our approach can be regarded as a simple way of cascading classifiers advocated by Viola and Jones in the context of vision applications [Viol 01], except that we are using the cascade to transform different inputs rather than to select regions of the feature space.

In this work, we do not address how fraudsters might adapt and attempt to evade our detection techniques (cf. [Nels 10, Huan 11]). We hope that our use of a large number of features would make evasive actions rather costly. Further investigation of such questions may be worthwhile.

Despite the existence and the deployment of various approaches to fraud detection, many financial institutions and companies still rely on manual review in addition to automatic screening, spending more than half of their fraud management budget on review-staff costs. Recent reports indicate that percentage of fraudulent transactions constantly increasing [Cole 15], suggesting that fraud detection is still an important problem that requires improved solutions.

5.3 Fraud detection classifier and its inputs

In the context of this case study, we define a fraudulent user as a registered user who intentionally deceives another user or a service provider, causing them to suffer a loss. There is a wide variety of fraud schemes [Albr 11]. The kinds of fraud relevant to Skype include, in particular, credit card fraud and other online payment fraud, as well as account abuse such as spam instant messages.

As we mentioned earlier, our aim is to catch those fraudsters that elude the first line of defenses at Skype. We define our target as those fraudsters that engage in activity for over K months (after creating their accounts), where K is a parameter which in this study we will set to 4 months. Our strategy is to combine information from different kinds of activities, both social (e.g., requests to be accepted as contact) and on the use of Skype products. To this end, we cast the problem of fraud detection as that of automated pattern classification.

More specifically the classification task of automatically deriving a function is described earlier in Section 2.1.4. Figure 5.1 depicts the entire workflow of the implemented process. The first two levels — the data and the description of the feature sets are outlined in Section 2.2.3.2.

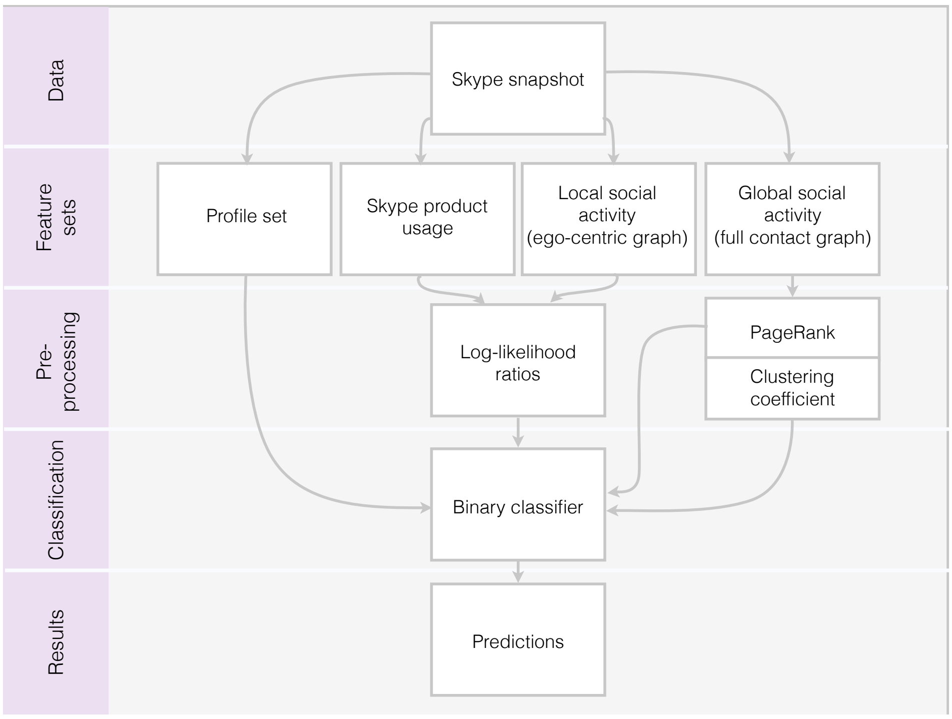


Figure 5.1: Entire workflow for the classification process.

An important note is that Skype takes the privacy of its users very seriously, and we implemented rigorous and carefully considered safeguards throughout this study in order to protect the privacy of Skype users. For example, all Skype IDs were anonymized using a one-way cryptographic salted hash function. None of the Skype usage data contained information about individual Skype communications, such as the parties involved in a communication, the content of communications, or the time and date of communications. Rather, it merely contained the number of days in each month that a Skype user used a particular communications feature, such as Skype chat, Skype video calls, and Skype In and Skype Out calls. Furthermore, the study's data was maintained on separately administered computer system, and access to it was strictly limited to the study's authors. Finally, the data was erased when it was no longer needed for research.

Relying on the user-to-user contact requests, we built a directed graph that consists of 677.8 million nodes and 4.2 billion directed edges. We also have timestamps of edge creations and deletions. Edge creation means that a user (the sender) sends a friendship request to another user (the receiver). Once the request is sent, we count it as an edge creation from the sender to the receiver. After the request, there are two scenarios: either the receiver accepts it or not. In case of acceptance, we consider it as another edge creation from the receiver to the sender. We handle edge deletions by the same principle.

Additionally, we have 10.8 million labels for fraudulent users, which we use both in training, to induce the classifier, and for testing its accuracy (see Section 2.1.2).

The labels further identify four different types of fraud schemes. Skype's definitions and procedures for these four different types of schemes is internal confidential information, which therefore we do not discuss further. Moreover, in our work, we choose not to rely on Skype's informal intent in those definitions, nor on Skype's software for each of the different types, in order to develop robust, self-contained methods.

From dataset **SKY** we extract information to construct a feature vector $(x_{i1}, x_{i2}, \ldots, x_{im})$ for each user. We divide the features into different types according to the kind of information they provide as well as how much processing is needed to transform raw data into the inputs to the classifier. Table 2.2 summarizes the various types that we use throughout our study. As we will see in Section 5.3.1, it is the combination of these different types of features that will enable a better detection of fraudulent activity.

The various pieces of information described in Section 2.2.3.2 constitute the inputs to our classifier. The user-profile features are in the form of categorical data that can be directly fed into the classifier. The remaining features (the activity logs and the local and the global social features) require further processing before they can be fed into the classifier. Part of the novelty in our approach stems from this pre-processing, which provides ways to enrich the discriminative classifier with the features from generative in nature HMM. For that, we use a hybrid model described in Section 3.2.

Log-likelihood ratios of activity logs The LLR are calculated according to the Equation 2.3. For the domain of fraud detection, the user activity logs include information about Skype product usage as well as the addition and deletion of contacts. We represent this information as a set of time series. In order to combine the time series with other static, categorical features, we further transform the data into a set of features represented as log-likelihood ratios. We perform this data transformation for each type of user activity (e.g., audio calls, video calls, contact additions) separately. The process is adopted for the domain by performing the following steps:

1. Given a specific type of user activity (e.g., audio calls), build two different models of activities, one for the normal users and the other for the known fraudsters, based on training data.

2. For each user (to be classified), produce a score using the above two models, representing how close this particular user is to the activities of a fraudster vs. the activities of a normal user.

3. Feed this score into the classifier, where the score will be combined with other features (and the scores from other activities) in order to perform the classification.

This approach leverages the information from an entire time series in order to produce a score that will be input to the classifier. It is fundamentally different from previous approaches that summarize a time series using simple statistics such as its mean or its standard deviation, and it can yield better results [Gold 12].

In the case of this task, after some initial experimentation, we settled for the following parameters and distributions:

- The user can be in one of two hidden states (which represent intensity levels), and the initial distribution π is a binomial distribution.

- Correspondingly, the matrix of transition probabilities A uses also binomial distributions.

- We discretize (see the discretization of time series in Section 3.1) the space of observables (level of activity) into three possible ones: O_1 (no activity in this month), O_2 (between one and five days of activity), and O_3 more than 5 days of activity in the month. Correspondingly, the matrix of emissions comes from a multinomial distribution.

We use the standard Baum-Welch algorithm in order to fit the maximum likelihood parameters listed above, with the same training data as for the rest of the classification training. Once the parameters are fitted, a standard dynamic programming algorithm is deployed. All the details are described further in Section 2.3.2.1.

There is one interesting aspect that the fraud detection approach used in this work can be seen as essentially employing cascading models that we describe in Section 2.2.3.2. The log-likelihood computations for each activity log provide a pre-classification from which the scores can be used and compared at the next level by the final classifier.

5.3.1 Evaluation procedure

For the experimental evaluation of our techniques, we consider a sample of Skype users that consists of $100,000$ randomly chosen users labeled as fraudulent by Skype, and the same number of randomly chosen users without being flagged. From this sample, we include a user u in our study if u is not blocked within 4 months since its account creation. We end up with $34,000$ such users. In this set, the ratio of users labeled as fraudulent to other users is 1:6.

We use 4 months as an observational period to collect activity logs $\ell(u_i, a_j)$. We selected the period of 4 months as a compromise: longer periods may result in more information, but our data pertains to a limited time window, and in addition we expect that relatively few fraudulent users escape detection for many months.

We train the models using 5-fold cross-validation with 5 repeated splits of 50% training and 50% test set. We use repeated splits due to a higher imbalance of the dataset that leads to a higher variation of the measures. We declare TPR, FPR and visualize the tradeoff between them using the ROC. The corresponding AUC helps to observe the overall performance of the models (see Section 2.1.2).

In our initial experiments, we benchmark several classifiers, including Random Forest, SVM, and logistic regression (using both lasso and ridge regularization). We estimated the accuracy via 5-fold cross-validation. As Random Forest yielded superior results (by $\approx 10\%$), we also adopted it for this case study as the main classifier.

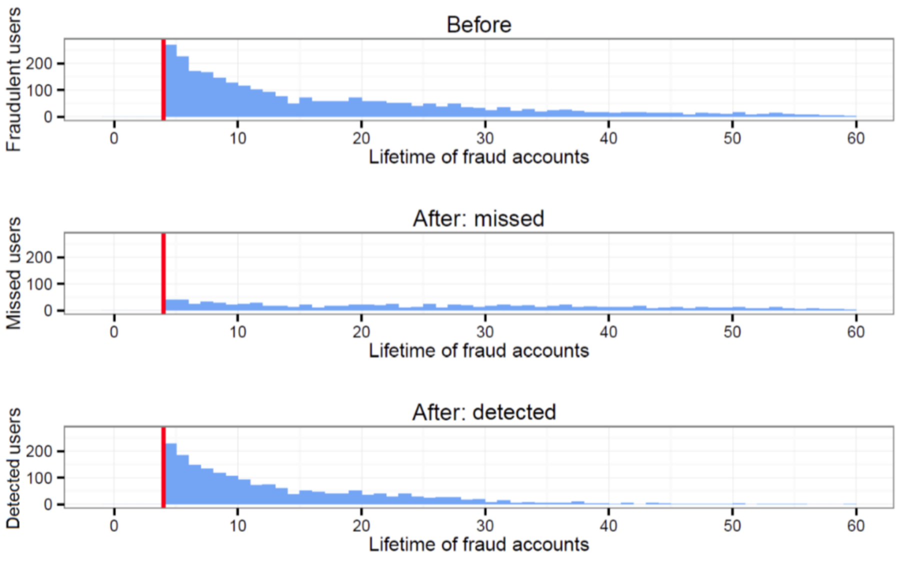


Figure 5.2: Distribution of fraudulent users by their lifetime in months (of undetected activity) before using our approach and after eliminating those fraudsters caught by our approach

5.3.2 Results

Using our approach, we achieve a TPR of 68% with a FPR of 5%. This TPR is especially significant as the fraudulent users that we are detecting were able to overcome the first line of defenses in the existing, effective detection system. Similarly, the FPR of 5%, which may appear as high for a stand-alone system, may be reasonable in the context of other defenses. In addition, with our approach, the number of the fraudulent users that elude detection for more than 10 months since their account creation decreases by a factor of 2.3.

Figure 5.2 plots the distributions of the account lifetimes for fraudulent users before and after the application of our methods. Each account lifetime is calculated as the interval from the account creation to the final detection (closure) of the account by Skype. The horizontal axis is in months. The figure labeled “Before” depicts the lifetime of fraudsters that escape Skype’s current defenses. The figure labeled “After: missed” shows the lifetime of fraudsters that would have escaped detection with our approach, and the one labeled “After:detected” shows those fraudsters that would have been

Table 5.1: Table of TPR under an FPR of 5% with corresponding confidence intervals, and AUC for different models when the features are used in isolation (*isolation*) and added one at a time (*nested*)

Features	**TPR (FPR = 0.05)**	**95% C.I.**	**AUC isolation**	**AUC nested**
Profile set	0.50	(0.48;0.52)	0.79	0.79
Skype product usage	0.25	(0.23;0.27)	0.65	0.84
Local social activity	0.54	(0.52;0.56)	0.74	0.86
Global social activity	0.37	(0.35;0.39)	0.68	0.87
All	0.68	(0.66;0.70)	0.87	

detected after the first 4 months of activity by our methods. The reduction in volume is apparent.

Note that our method is most effective in detecting fraudsters that would have 10 or less months of activity (after the initial 4 months), while missing most of those that stay active for over 30 months. Preliminary investigations indicate that an initial period of observation longer than 4 months would improve the detection of such long-term fraud. Perhaps a cascading set of classifiers, each with a different initial period of observation, would be helpful. We also conjecture that a large number of these fraudsters took over the accounts of normal users. Other techniques, such as change-point detection, may be fruitful in this context.

Figure 5.3 shows the overall performance of classifiers built using only one feature at a time (left-hand side, labeled “separate models”) and classifiers built by adding one feature at a time (right-hand side, labeled “nested models”). For the nested models we begin from the simplest classifier that uses only the profile features. The next model combines profile features and product usage features. We continue to increase the complexity of the classifier by adding the feature sets one-by-one. In this study we use only a particular order for adding the feature sets. The order corresponds to the complexity and computational effort in pre-processing the features as discussed previously (see Section 3.1).

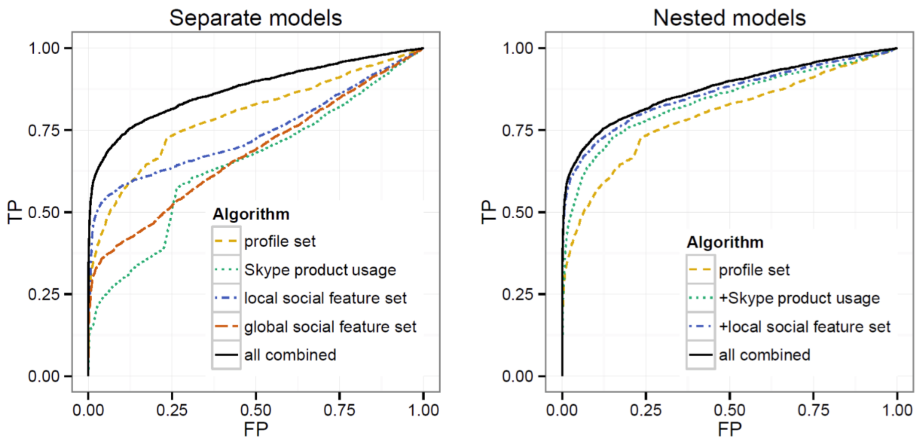


Figure 5.3: ROC curves for models built on one feature (left-hand side and labeled "separate models") and models built by adding one feature at a time (right-hand side and labeled "nested models")

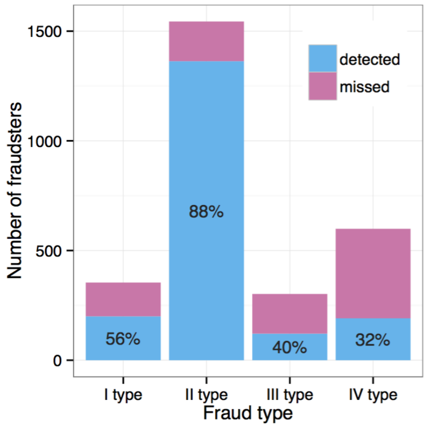


Figure 5.4: Distribution of fraudulent users by types and proportion of detected among them

As depicted in the graph for nested models, the improvement in performance as we add features is monotonic, confirming that all the features contribute to detection.

Table 5.1 represents these results quantitatively. As can be observed, a model based on only the local social activity information has the best TPR, and the user profiles yield the best overall AUC score. Also, as mentioned above, the best model is one where all the features are used.

Finally, we report on the statistics per type of fraudulent user. As can be seen from Figure 5.4, we are able to detect most of the type II fraudsters, but we are less successful in detecting type III and type IV fraudsters. In further work, it may be attractive to investigate various feature sets for different types of fraud, tailoring classifiers to the patterns of behavior that correspond to each type of fraud.

5.4 Conclusions

In this case study, we present and evaluate an approach to detecting fraudulent users based on supervised machine learning. The approach combines information from diverse sources such as user profiles, user activities, and social connections. As our results demonstrate, fraud classification improves as we add each one of these sources (see Figure 5.3).

The concrete goal of our work was to detect stealthy fraudulent users. Specifically, we identified 68% of these users within the first 4 months of activity with a 5% false positive rate, and reduced the number of undetected fraudulent users active for over 10 months by a factor of 2.3. We consider that these quantitative results are encouraging and positive.

A central contribution of this work is a set of methods for transforming raw data into features suitable for consumption by classifiers. In particular, we use HMMs in order to enhance discriminative models from the time series data. The applications of this approach go well beyond fraud detection (as suggested by work on failure prediction [Gold 12]).

Our experiments also suggest several directions for further investigation. The different detection rates for various types of fraud indicate that each source of information may correlate differently with those types of fraud; hence, more elaborate ways of combining and cascading classifiers may lead to enhanced fraud detection for particular types of fraud. It should also be interesting to perform experiments with longer time series, attempting in particular to detect points in time at which users change behavior. Those changes in behavior sometimes result from account hijacking, a difficult, important problem that machine learning may help address.

Chapter 6

Case study III: ECoG signal discrimination

6.1 Introduction

In the previous chapters we outlined two case studies that investigate combination of multivariate sequences and static data, and briefly mention the useful features from the graph data. In BPM domain we compare different baselearners and find that it is beneficial to use a payload data along with the control-flow, while in the case study of fraud detection we investigate how to fuse static features with the graph data and short time series. In this case study we use the ECoG data (see Section 2.1.4 for details). Modular structure of the hybrid approach allows to use any generative model that deals with sequential data in the feature enrichment process. In the two previous case studies this task is performed by HMM. However, there are other possibilities. Recently, RNN with LSTM units demonstrated superior results [Grav 13] and we decided to create an LSTM based hybrid and compare its performance to HMM-based hybrid model.

In previous case studies we showed that the combination of static and sequential features is very common on practice. In addition, for almost any sequential dataset it is possible to extract static features out of sequential data, which leads to a possibility to extract more information from the same dataset. As we showed earlier (see Chapters 4, 5), it is possible to tackle sequential data by transforming sequences into feature vectors that can be fed into a discriminative method. Ensemble [Diet 00] methods provide another way to address the issue: predictions made by a *sequential* model on sequential data are combined with the predictions of a discriminative classifier on static data (see Section 3.2). We devise a hybrid model (see description of such an approach in Section 3.2) and show that the hybrid way of stacking models [Wolp 92] is in general more beneficial than ensemble methods. We summarize our main contributions as follows:

- We postulate an idea that combining sequential and static information can boost classification performance.
- We empirically demonstrate that a hybrid method outperforms ensemble and other baseline methods on ECoG discrimination problem and on several other public datasets.
- We compare hybrid approach with the LSTM-based generative model to the hybrid approach with the HMM-based generative model.
- We perform a controlled experiment on a synthetic dataset to investigate how dataset characteristics affect the baseline, ensemble and hybrid methods' performance.

6.2 Prerequisites

In this section, we describe the details of model architectures and set-ups for the experimental phase.

We employ Random Forest (see Section 2.3.1.2) as a discriminative model in two different ways: as a stand-alone discriminative classifier and as a metalearner in ensemble and hybrid architectures. The `Scikit-learn` [Pedr 11] implementation of Random Forest was used in our experiments. For a HMM model (see Section 2.3.2.1) We use the `hmmlearn` [Lebe 15] implementation of Gaussian HMM. Another generative model that we employ is an LSTM network (see Section 2.3.2.2).

In all LSTM models we use mean squared error (MSE) between the true test sequence and the generated sequence as our error function, RMSProp [Daup 15] acts as the optimization method. `Keras Deep Learning` library [Fran 16] provides the implementation of LSTM.

6.3 Methods

In this section, we first describe the approaches that can be thought of as competitors to the hybrid models, then we explain our main contribution — the concept of a hybrid model and its versions.

Section 6.3.1 describes the methods that use only static or only sequential features (*unicomponent*) with a single discriminative algorithm. Section 6.3.2 makes a step forward by using a feature space that includes both data types (*bicomponent*) transformed to make the dataset suitable for the algorithm at hand. For example, if we apply Random Forest to sequential data we employ sequential-to-static transformation (see Section 3.1); on the opposite side, if we need to apply a sequential model to static data we perform static-to-sequential transformation by changing feature values into "fake" sequences (see Section 3.1.1). Let the value of a static feature f_i of a sample be x, f_i will be represented by a sequence of length l_d with value x at each time step. Section 6.3.3 discusses what are the options to construct feature spaces that combine static and sequential features effectively and describes ensemble and hybrid methods applied to these feature spaces.

6.3.1 Stand-alone models for unicomponent data

In this section we describe the simplest baseline approaches. These are the models that are fit for only one data type: either static or sequential.

Random Forest on Static or Sequential Features. The most straightforward way to handle a multicomponent dataset is to build a model on static features only. In this work such an approach is referred to as RF_s (1)[1]. We denote the number of static features in a dataset as n_s. If the sequential data has the sequences of a fixed length, then it is straightforward to apply Random Forest on this data — we use time spatialization to represent time series as one long feature vector. One obvious drawback of such an approach is low performance due to the curse of dimensionality when sequences are extremely long [Keog 11]. We use this baseline to estimate if the use of a sequential model is justified. Our intuition is that if Random Forest is able to achieve the same performance on the sequential data as sequential

[1]The number in brackets stands for the model ID we use throughout the text.

models do, then the particular sequential dataset does not have a strong temporal component. We denote this method as RF_d (4) and use it for the comparison with sequential models such as HMM and LSTM.

Hidden Markov Models on Sequential Features. The method denoted as HMM_d (2) is a direct application of HMM to sequential data. We use HMM as a classifier as described in the Section 2.3.2.1.

Long Short-Term Memory on Sequential Features. The method denoted as $LSTM_d$ (3) uses the same technique as HMM_d (2) to act as a classifier. In order to capture temporal dynamics in the data we train a LSTM network to predict each value in the sequence $(x_2, x_3, \ldots, x_{l_d})$ given all the previous ones. In the case of multivariate sequences the input size of the network is equal to the number of sequential features n_d; the number of nodes n_{LSTM} in the network is estimated separately for each dataset. We train a single layer of LSTM followed by a fully connected layer of size $n_{LSTM} \times n_d$ (see Figure 6.1 for the visual explanation). There are many other possible LSTM architectures, but it is out of the scope of this work to compare them all. We chose the architecture described above mainly for two reasons: 1) in the current formulation it is more comparable with HMM-based generative models and, 2) among other architectures we tried, this one provided best or comparable results. The architecture of both G_{POS} and G_{NEG} networks are shown in Figure 6.1.

6.3.2 Stand-alone models for bicomponent data

The second class of baseline approaches utilizes both static and sequential data by concatenating them in such a way that a single classification method is applicable. This is the most naïve way of using both data types simultaneously, and, as it will be discussed later, has obvious limitations.

Random Forest on static and sequential features. The method under the name $RF_{s,d}$ (5) transforms sequential data to static, concatenates it with the original static features and employs Random Forest on the resulting feature set.

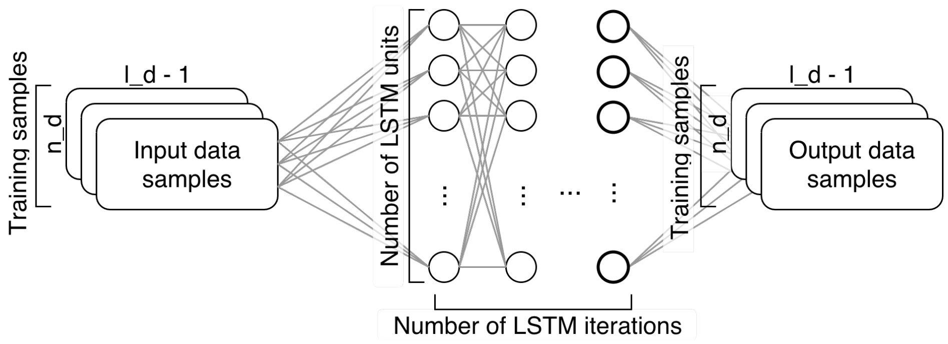


Figure 6.1: LSTM network architecture for sequence generation. We predict value at time step t from the time steps $1 \dots t-1$. Therefore, the output sequence is the lagged version of the input sequence. The activations of the LSTM units at the last iteration (shown in bold) are used as the features in the HYB_{LSTMA} (12) model (see Table 6.1).

Hidden Markov Model on static and sequential features. The method implemented in $HMM_{s,d}$ (6) employs static-to-sequential transformation: for the dataset with n_d sequential features of length l_d it produces n_s additional sequential features of length l_d. All of the values along these sequences are constant and equal to the original value of the static feature (see Section 3.1.1). Using this transformation we extend sequential feature set from n_d features to $n_d + n_s$ features and apply HMM on it similarly to HMM_d (2).

Long short-term memory on static and sequential features. Using the trick from $HMM_{s,d}$ (6) we obtain the sequential features from the static features, concatenate them with the original sequential features and train an LSTM classifier on the combined feature set. The learning algorithm itself is analogous to $LSTM_d$ (3). We refer to this approach as $LSTM_{s,d}$ (7).

6.3.3 Multiple models on bicomponent data

Dealing with bicomponent data is the main focus of this work. In this section we look into more efficient ways of combining static and sequential features, than the straightforward methods presented in the previous section.

6.3.3.1 Ensemble models

With an ensemble approach one can train different models for different data modalities (for example, Random Forest for static features and LSTM for sequential features) and combine their predictions using a linear model or another layer of Random Forest (or any other discriminative method).

In this work we have two methods based on the ensemble approach: ENS_{HMM} (8), which takes the predictions made by Random Forest and the predictions made by HMM classifier, and the ENS_{LSTM} (9) that combines Random Forest predictions with the predictions of LSTM classifier. In both these models the final prediction is obtained by training an additional Random Forest model using predictions as features. For more details about ensemble methods refer to Section 2.4.

Ensemble of HMM and RF. The ensemble method ENS_{HMM} (8) has two stages. First stages works with the first half of the training set — a Random Forest is trained on the static features and HMM classifier on the sequential ones. In the second stage we use the samples from the second half of the training set and estimate class probabilities for each sample using the models trained in the first stage. In case of a binary classification problem each sample is represented by 4 features. We obtain a new dataset where each sample s from the second half of the original training set is represented by: log probabilities that $s \in \texttt{POS}$ and $s \in \texttt{NEG}$ provided by Random Forest, and $L(s|G_{\texttt{POS}})$, $L(s|G_{\texttt{NEG}})$ log-likelihoods provided by HMM. In the similar way we feed samples from the original test set into these models and obtain a test set with the same 4-dimensional feature space. In more details we describe the procedure of such feature extraction for HMM and LSTM in Sections 2.3.2.1 and 2.3.2.2 with the corresponding formulas 2.3 and 2.4. Finally, we train Random Forest on 4-dimensional training set and evaluate it on the corresponding test set. For the detailed explanation of the experimental pipeline see Section 6.4.1.

Ensemble of LSTM and RF. LSTM ensemble ENS_{LSTM} (9) builds a new feature set in a similar to ENS_{HMM} (8) way. The first two features are the same as in the case of ENS_{HMM} (8). The second two features are obtained with `POS` and `NEG` LSTM networks. Namely, an input sequence of a data sample s is fed into each of the networks, and the corresponding output sequences are generated. Per class mean squared errors MSE_{POS} and MSE_{NEG} between the true output sequence and the generated sequences are used as the new features for the sample s. The number of such features is equal to the total number of classes.

6.3.3.2 Hybrid models

The general idea of the hybrid approach is to employ generative models such as HMM or LSTM to act as feature extractors from sequential data. As generative models are able to generate sequences from the training data distribution, it is reasonable to assume that these models can capture temporal dynamics in the data. Therefore, the features extracted using these models can act as an approximation for temporal information contained in the data. These features are concatenated with the static features and a discriminative classifier (Random Forest) is used to build the final predictor.

Since naïve ways of combining sequential data with static features give poor performance (see Figure 6.6) we use the data representation provided by sequential models to obtain a fixed-size feature set that contains knowledge extracted from the temporal component of sequential data.

There are different features that can be extracted from generative models, with one such example being the Fisher kernels [Jaak 99]. In our experiments we use log-likelihood ratios, MSE ratios or LSTM activations as features for the enrichment of the static feature set. Our experiments have shown that combining static features with ratios yields better performance than combining static features with raw log-probabilities or MSE scores. In the following subsections we go through the hybrid architectures we have explored.

Hybrid of static features and HMM ratios. In the HYB_{HMM} (10) method two generative HMM models are built on the first half of the training set: one for the samples with the positive class labels and one for the samples with the negative class labels. These models are used to enrich both static features of the second half of the training set with r_{HMM} ratios as well as the static features of the test set. Finally, Random Forest is trained on the enriched feature set and evaluated on the enriched test set.

Hybrid of static features and LSTM ratios. In a very similar fashion we can use LSTM to build generative models. HYB_{LSTM} (11) extracts MSE ratios r_{LSTM} and enriches the set of static features on the second half of the training set. Random Forest is trained on the enriched training set and evaluated on the enriched test set.

Hybrid of static features and LSTM activations. Depending on how detailed information we want to give to the final classifier we can choose which features to extract from a generative model. The log-likelihood ratios are almost the most compressed form of the information about the data samples. Less compressed features depend on the inner workings of a particular generative model. For a LSTM network we can take activations of the last LSTM layer at the last iteration and use those activations to enrich the set of static features. The model that does that is denoted as HYB_{LSTMA} (12). The final step is similar to all other hybrid architectures: Random Forest is trained on the enriched training set (in this case it consists of static features concatenated with LSTM activations) and evaluates the performance on the test set. In our experiments, however, such an approach is less accurate than the one based on the log-likelihoods.

In Table 6.1 we summarize all the models and mark the feature sets used by each model. Note that in case of univariate datasets some of the models are virtually the same: model (5) is the same as the model (1), model (7) as the same as the model (3) and the model (6) is the same as the model (2). The reason is that for univariate datasets we use spatialized dynamic data as static features.

Table 6.1: List of models and the feature sets they operate upon.

			Raw features		Predictions			Ratios		Activations
	Nr.	Model name	Static	Sequential	RF	HMM	LSTM	HMM	LSTM	LSTM
Stand-alone	1	RF_s	✓							
	2	HMM_d		✓						
	3	$LSTM_d$		✓						
	4	RF_d		✓						
	5	$RF_{s,d}$	✓	✓						
	6	$HMM_{s,d}$	✓	✓						
	7	$LSTM_{s,d}$	✓	✓						
Ensemble	8	ENS_{HMM}			✓	✓				
	9	ENS_{LSTM}			✓		✓			
Hybrid	10	HYB_{HMM}	✓					✓		
	11	HYB_{LSTM}	✓						✓	
	12	HYB_{LSTMA}	✓							✓

6.4 Evaluation

6.4.1 Evaluation procedure

All the experiments for this case study use accuracy as the basic evaluation measure as all the provided datasets are balanced.

In the case of hybrid methods (see Section 3.2) the splitting strategy and feature handling are not straightforward. In this subsection we explain the full pipeline both in the case of a training-test split and in the case of cross-validation.

Training / test split. Splitting the data entails non-trivial steps in order to perform feature enrichment without introducing bias into the models. Methods that do not make use of feature enrichment employ the standard machine learning pipeline where a training set is divided into training and validation subsets. Model hyperparameters are estimated on the validation set and the final model is trained on the whole training data and tested on the test data. In case of ensembles and hybrids, however, the pipeline differs. First, the training set is divided into two equal halves, let us call them training$_A$ and training$_B$. Training$_A$ is used to train the first tier of models, which we call *feature extractors* — given a data sample their purpose is to output additional features that describe that sample. In case

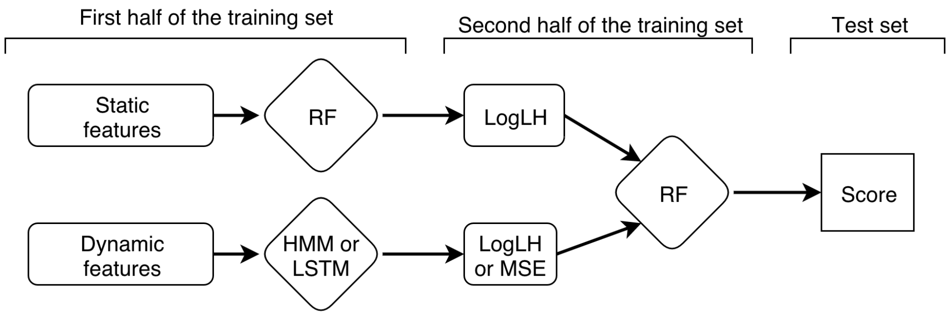


Figure 6.2: Architecture of an ensemble. The first half of the training set is used to create models according to the data modality: Random Forest for static data and a generative model for the sequential data. These models are applied to the second half of the training set and to the test set to extract predictions and form a new feature space. Random Forest is trained on the enriched second half of the training set and evaluated on the enriched test set.

of ensembles these are baselearner output features as log-likelihoods of class labels as discussed in Section 2.3.2, for hybrids the extracted features can take any of the forms discussed in Section 3.2. Hyperparameters of the feature extractor models are estimated on a validation subset of training$_A$.

Once feature extractors are fully trained we use them to enrich training$_B$ and test sets: sequential data of each sample from those sets is fed into a feature extractor. The extracted features are concatenated with the static features of that data sample. For example in case of the hybrid model with LSTM activations, if we had a dataset with M_d sequential features and M_s static features, and we use LSTM network with 128 LSTM units as the feature extractor, then the new feature space would be of size $M_s + 128$.

The final step of the pipeline is to train a second tier — a metalearner — on the new feature space. Hyperparameters are estimated on the validation subset of enriched training$_B$, after that a final model is trained on the whole enriched training$_B$ set and tested on the enriched test set.

The complete process is depicted in Figures 6.2 and 6.3.

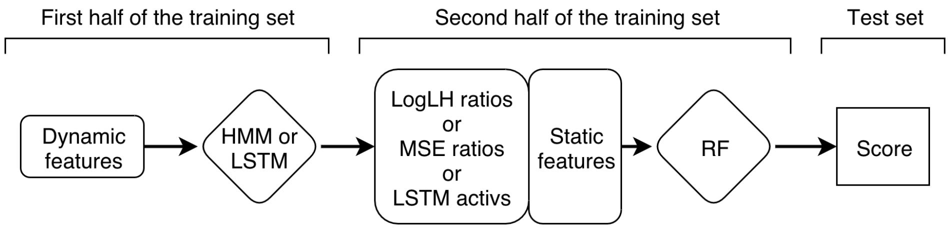

First half of the training set
Second half of the training set
Test set
Dynamic features
HMM or LSTM
LogLH ratios or MSE ratios or LSTM activs
Static features
RF
Score

Figure 6.3: Architecture of the hybrid model. First half of the training set is used to create a model which will act as a *feature extractor*. Feature extractor is applied to *enrich* the feature set of the second half of the training set and the test set with additional features. Random Forest is trained on the second half of the training set to create a final classifier, which is evaluated on the test set.

Cross-validation. In the case of proposed hybrid architecture, cross-validation provides a more efficient use of data. We take advantage of the cross-validation technique in the following manner.

Instead of dividing the data set into two parts as it was done for the training/test split case, we split data set into $\{C_1, \ldots, C_k\}$ chunks, where k is the number of cross-validation folds. We train a feature extractor model using the data samples from chunks $\{C_1, \ldots, C_{i-1}, C_{i+1}, C_k\}$ and apply that model to enrich the samples from the C_i chunk. This process is repeated for every chunk and as a result the whole dataset is enriched without introducing overfitting bias.

The next iteration is to train a second tier model on the enriched data. This is once again done using cross-validation and exactly on the same chunks of data we used before. The process is no different from the classical application of cross-validation as outlined in section 2.1.3: a model is trained on chunks $\{C_1, \ldots, C_{i-1}, C_{i+1}, C_k\}$ and evaluated on the C_i chunk. The reported accuracy is the average accuracy over k folds.

Hyperparameter Optimization. In order to find the best hyperparameter combination for a model, we apply Spearmint [Snoe 12] to search through the parameter space. The method behind the tool is Bayesian optimization and it has been shown to be able to find hyperparameters that yield performance equal or superior to that achieved using other hyperparameter optimization techniques. In our experiments every dataset has its own set of parameters, see Table 6.4 for the details.

6.4.2 Datasets

We compare the described approaches on several datasets from different domains as well as on a simulated data. In this section we describe the datasets and their properties.

Synthetic ARMA dataset. In order to be able to compare results with the ground truth and form the intuition how much information can be extracted from different types of features, we have generated a synthetic dataset with specific properties. Namely, one of the posed questions of this work is whether combining static and dynamic features can boost the overall performance on a given dataset. We model the required conditions by splitting the data into four blocks in the way explained in Table 6.2. Block 1 has samples with positive labels and values for the static features are generated from G_{POS} model, while the dynamic features are generated from the G_{NEG} model. In block 2 the situation is reversed. The two last blocks have correct labels for both parts. Therefore, models that do not use information from both sources should be in a worse position. Indeed, models (1)–(4) cannot achieve accuracy of more than 0.75 as can be seen in Figure 6.7.

The dynamic features are simulated with ARMA(p, q) process [Hami 94], where the orders of autoregressive (AR) and moving average (MA) parts are drawn from a uniform distribution, $p, q \in \{1, \dots, 5\}$, while coefficients of AR and MA processes are $\alpha_i, \beta_i \sim \mathcal{U}(-0.1, 0.1)$, respectively. All values of the static features in the synthetic dataset are drawn from the Gaussian distribution, $\mathcal{N}(\mu, \sigma^2)$, where $\mu \sim \mathcal{U}(0, 2)$ and $\sigma \sim \mathcal{U}(0, 2)$.

For illustration purposes Figure 6.4 depicts eight randomly chosen time series from the created synthetic dataset. The difference between classes is not obvious, and, therefore, is not overly simple as a classification task.

Real-life datasets. We use datasets with different aspects: few univariate time series widely used in the literature — FordA and FordB [Chen 15]

Table 6.2: Synthetic dataset is designed in a specific way. Each block contains static and dynamic data, however the dynamic data in block 1 and static data in block 2 are useless, making it impossible for a model that operates only on one data modality to classify the whole dataset correctly.

Block	Classifiability	Label	Static	Dynamic
1	Classifiable by discriminative model as `T`, but generative model will confuse it for `F`	`T`	$\sim \mathcal{N}_{\mathtt{T}}$	$\sim \mathrm{ARMA}_{\mathtt{F}}$
2	Classifiable by generative model as `T`, but discriminative model will confuse it for `F`	`T`	$\sim \mathcal{N}_{\mathtt{F}}$	$\sim \mathrm{ARMA}_{\mathtt{T}}$
3, 4	Classifiable by both as `F`	`F`	$\sim \mathcal{N}_{\mathtt{F}}$	$\sim \mathrm{ARMA}_{\mathtt{F}}$

and a multivariate dataset from a particular domain — classification of electrocorticography (ECoG) recordings from BCI competition III [Lal 04]. Also, we show the results on the Phalanges and Yoga datasets [Chen 15], where the baseline methods perform as well as ensemble and hybrid approaches, and we discuss why it is the case. For the characteristics of the chosen datasets the reader is referred to Table 6.3.

Table 6.3: Descriptions of the real-life datasets.

Datasets	Samples	Train set	Test set	Static features	Sequential features	Sequence length	Source of benchmark
ECoG	10584	5-fold CV		320	64	300	[Schr 05]
FordA	4291	1320	3601	500	1	500	[Bagn 12]
FordB	4446	810	3636	500	1	500	[Bagn 12]
Phalanges	2658	1800	858	80	1	80	[Chen 15]
Yoga	3300	300	3000	426	1	426	[Bayd 15]

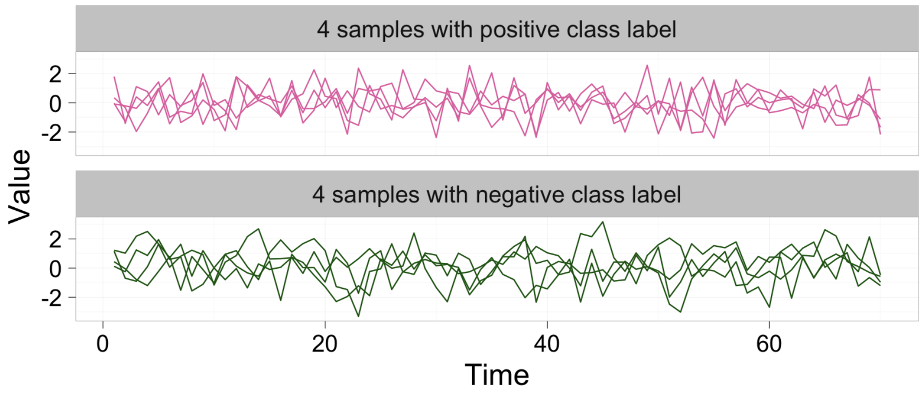


Figure 6.4: Example of generated synthetic time series.

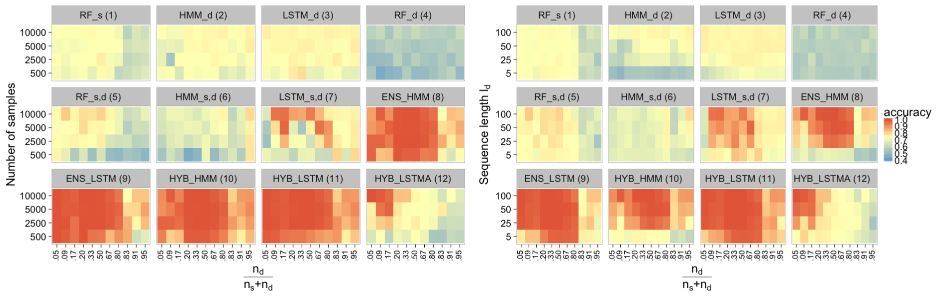


Figure 6.5: Performance of 12 models on the synthetic dataset with varying parameters. The ratio between the number of dynamic features and the total number of features plotted on the horizontal axis. The vertical axis corresponds to the size of the dataset on the left plot and to the length of the sequence on the right plot.

Benchmarks from the literature exist for all of the datasets except for the ECoG dataset. To the best of our knowledge we compare our scores with the highest reported results and follow the same data splitting strategies. Namely, we demonstrate the results on the train and test sets of the same size as used in the literature. The sources of the benchmarks are provided in Table 6.3. It is worth mentioning that despite the fact that the best result gained for the ECoG dataset during the BCI Competition has accuracy of 0.91 [Schr 05], the authors use elaborate hand-crafted features extraction methods such as combination of bandpower, CSSD/Waveform Mean and Fisher Discriminant Analysis. Since we do not have access to the features

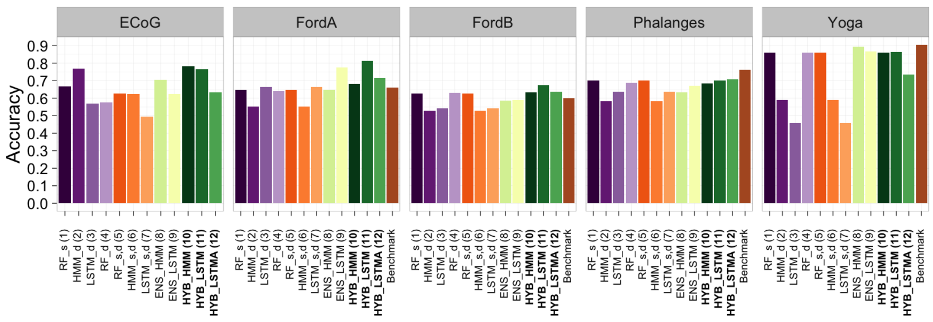


Figure 6.6: Performance on the real-life datasets.

they have used, we limit ourselves to classical Fourier analysis of ECoG signal. Due to the differences in preprocessing the fair comparison cannot be easily drawn. For ECoG dataset we apply 5-fold cross-validation and compute the mean of accuracies over the folds. In Table 6.4 we list the hyperparameters of all the trained models.

Table 6.4: Estimated hyperparameters. Column names stand for: LSTM Size, Dropout, Optimization method, Batch size, number of Epochs; number of HMM States, number of Iterations; number of RF Trees.

	LSTM					HMM		RF
Dataset	S	D	O	B	E	S	I	T
ECoG	2000	0.5	rmsprop	32	50	6	50	500
FordA	512	0.0	rmsprop	1	20	2	50	500
FordB	512	0.0	rmsprop	1	20	2	50	500
Phalanges	128	0.0	rmsprop	1	10	2	50	500
Yoga	256	0.0	rmsprop	1	10	2	50	500

6.4.3 Results

In this section we report accuracies achieved by all of the approaches on synthetic and real-life datasets and discuss our findings.

Insights from synthetic ARMA Dataset. As can be seen from the results on the synthetic dataset (Figure 6.7) RF is far from good when dealing with dynamic data, thus it confirms the intuition that the models designed to work with dynamic data have merit. Stand-alone models on bicomponent data (models (5)–(7), see Section 6.3.2) fail to capture the information from the both sources. This observation is also generally true for the real-life datasets (see Figure 6.6). Both the ensemble and hybrid methods achieve almost perfect accuracy, and thus are good at extracting information from sequential and static sources simultaneously.

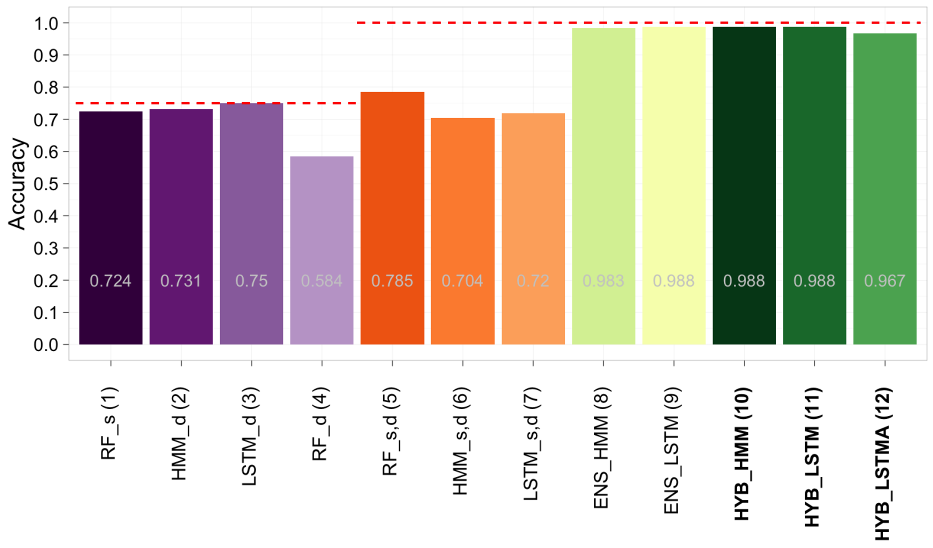


Figure 6.7: Performance on the synthetic dataset. The vertical dashed lines show the maximal achievable level of performance.

Next, we investigate various dataset characteristics with respect to the methods' performance by generating datasets with varying size, sequence length (l_d), number of static (n_s) and dynamic features (n_d). The resulting heatmap is shown in Figure 6.5. Note that the ratio on the horizontal axis is represented in discrete form and the intervals are not equal due to a limited variation of the parameters.

There are a few observations that may be of interest for practitioners:

- Both ensemble and hybrid HMM (models ENS_{HMM} (8), HYB_{HMM} (10)) show superior performance on longer sequences and when the number of dynamic and static features is balanced.
- Hybrid and ensemble models based on LSTM (ENS_{LSTM} (9), HYB_{LSTM} (11)) are less affected by the length of the sequence and the ratio between static and dynamic features. They perform very well across the whole range of those parameters, except for very short sequences.
- All methods show lower performance on the datasets with very short sequences and on the datasets with high imbalance towards dynamic features. The decrease is visible when the proportion of dynamic features reaches 80%.
- Noteworthy is also the overall similarity between the patterns of ensembles and hybrids, though latter have a bit higher accuracy.
- Hybrid model HYB_{LSTMA} (12) with LSTM activations seems to perform well only for a bigger dataset size, longer sequences and with only a few dynamic features.

Results on real-life datasets. The results across all real-life datasets are shown in Figure 6.6. In general, every dataset has its own specifics and thus, no single method performs best on all of them. For example, on the ECoG dataset the model HMM_d (2) is almost as good as hybrid methods in spite of showing poor performance on all the other datasets. On the Yoga and Phalanges datasets Random Forest on the static features performs as well as the hybrid and ensemble methods. One possible reason why on some datasets combination of dynamic and static features does not give higher performance could be the absence of temporal dynamics in the data. In such case the use of sequential models is irrelevant. The degree of temporal connection can be estimated by comparing stand-alone models HMM_d (2) and $LSTM_d$ (3) with RF_d (4). If Random Forest performs on dynamic data better or similar to HMM and LSTM, then the temporal aspect is not present (or sequential models fail to grasp it).

Hybrid model HYB_{LSTM} (11) improved on the best result from the literature on the datasets FordA and FordB. On the ECoG dataset both hybrids HYB_{HMM} (10) and HYB_{LSTM} (11) outperformed the other methods. Moreover, despite the fact that the hybrid methods on Phalanges and Yoga datasets do not beat the accuracy from the literature, in all of the datasets they are either the best or very close to the best results.

It is also interesting to notice that the observations obtained from the analysis of synthetic data are in line with the performance on the real datasets. Namely, if a dataset has fewer samples or the sequences are rather short (as in the case of Yoga and Phalanges datasets), then the hybrid approach does not provide a performance boost. However, if a dataset is large and has long sequences as in the case of ECoG, FordA and FordB (see Table 6.3 for the dataset characteristics), then hybrids outperform other methods.

6.5 Conclusions

We propose and explore a novel way of combining dynamic and static features in order to make it possible for a classification model to capture temporal dynamics and static information simultaneously. Previous approaches to this problem relied on ensemble methods where different models operate on different data modalities and their predictions are combined. The hybrid approach we propose goes to a lower level and explores the possibility of combining models not at the level of predictions, but earlier — it concatenates static features with either predictions of generative models, ratios of class probabilities, or, when possible, with inner representation of the data provided by a generative model. We demonstrate that this approach outperforms other approaches and report results on several public datasets. Additionally we explore the behavior of 12 different models on synthetic data and describe how performance depends on such properties of a dataset as the number of samples, number of static and dynamic features and the length of the sequence, providing the guidelines for practitioners.

Chapter 7

Conclusion and future directions

7.1 Summary of contributions

In spite of the existence of the well-defined machine learning pipeline and a clear evaluation for the classification task the practical use of these techniques in industry and on real-life data is still in its initial state. Manual feature extraction, various data types, poor quality of the datasets contribute to this lag between the academia and industry. In this work we describe several case studies, discuss the use of the classification framework and show the transformation of various sources of data and different data types into the suitable input for a classifier.

This thesis approaches the problem of combining sequential and graph data with static information for a classification task from the perspective of practical applications. We propose a framework that uses information not only from static features, but takes into account relevant information from sequential data, and combines it together.

First, we propose the hybrid model that enables to combine generative and discriminative model types, which, in turn, enriches the model ability to capture data patterns. We use generative models for the sequential data and transform the model outputs to the informative features that are fed to a discriminative classifier. Overall, the results show that hybrid method outperforms others when the data largely consists of sequences, but static features also play an important role and cannot be discarded.

Next, we define a progressive index-based framework that allows one to make early predictions, without the requirement for the case to end. Essentially, it is a system of stacked ensemble type that handles multivariate sequences with the static data and outputs predictions for each timestamp. It can incorporate hybrid models or any other classifiers and learns to use information from sequential and static features as well as to choose which base models to use.

We outline three case-studies, each from the different domain. The first one comes from the Business Process Mining, where we build a model to classify deviant processes at early stages, after the execution of just a few events. Our contribution differs from the other similar works as the proposed classifier takes into account not only the information from the control-flow, but advances by accounting for a data payload for each event. Therefore, the classifier is able to capture deviant behavior not only in the process itself, but also in the accompanied information. We use progressive-index based framework, compare several different encodings of the data and suggest the use of index-based encoding, while the encoding based on the hybrid model does not improve the results in a systematic manner due to the insufficient number of long sequences. The empirical evaluation has demonstrated that with index-based encoding we achieve an AUC of 90% after the execution of 5 events, while baseline encodings without additional information do not achieve higher than AUC of 60%.

The second case-study shows the results of a classifier for a fraudulent behavior in Skype network. We propose a method that uses information from various data sources in order to find stealthy fraudulent users, those that are either able to trick the first-line defense or salient ones. The various sources include profile information, sequential data about product usage, dynamic changes in social network and the information from the snapshot of the global network. We apply the hybrid model to fuse all these sources together, which results in a TPR of 68% with a FPR of 5%.

The third case study explores the possibilities of the hybrid model and compares it with the ensemble technique with the aim of providing guidelines for hybrid model usage. We use ECoG dataset and several other public datasets to overview 11 different classification models in order to choose the best way of combining static and sequential features. Empirical evaluation demonstrated that the use of the state of the art LSTM model as a generative model candidate is more beneficial than use of the Hidden Markov Models. However, the use of LSTM activations as the input to the classifier leads to worse results than in the case of the MSE scores. In addition, with the help of the synthetic data, we alter different dataset parameters like dataset size, number of static and sequential features, and the length of the sequences to empirically evaluate, which properties a dataset should exhibit in order to benefit from the advocated approach.

7.2 Future directions

This work proposes a path towards richer representation of data for a classifier. However, there are several aspects that potentially may improve the suggested body of work. We outline a few interesting directions.

Interpretability. It is often discussed that for the real-life problems black-box methods are much less preferable and people have difficulties trusting the prediction when the reasoning behind the decision of the classification is not clear. This poses a serious challenge to either achieve the same level of precision with the existing transparent classification methods or to come up with the additional techniques that expose the rationale behind the black-box methods. Currently it is an active line of the research and several attempts were made. For example, the work of Ribeiro et al [Ribe 16] proposes the LIME technique, where the predictions of black-box classifiers are approximated locally by a simple, interpretable linear model. Another interesting direction towards interpretability is briefly outlined in the work of Krakovna et al. [Krak 16], where Hidden Markov Models are combined with the LSTM in order to increase the interpretability of these models.

Earliness. One of the benefits using the progressive index-based framework is its ability to make predictions as the case progresses. However, there is a tradeoff between the confidence of the early prediction and the cost of delaying the decision. It is worth investigating how to incorporate the detection of such an optimal timepoint into the suggested framework. The paper of Tavenard et al. [Tave 16] suggests a method, which for each time series provides the time when the prediction should be done in order to minimize the cost of delaying the decision together with the cost of not achieving enough accuracy.

Sequential component. Another contribution to the transparency of the methods would be a pre-processing test that identifies whether the multivariate sequential data contains more information for a classifier or it can be handled in a static manner. It is clear that the simple transformations like sequential-to-static ones are computationally cheaper, and not all the datasets exhibit strong sequential behavior. Although the suggested hybrid approach can achieve the same accuracy as the simpler methods on uni-component data, it is more efficient to use simpler methods when dataset does not exhibit sequential dependency. A simple explorative test before any modeling procedure would help to save time and effort.

Models. The field of machine learning advances with the rapid tempo, and state of the art methods change in no time. It is worth considering to compare the outlined approaches with the more advanced techniques. For example, we use a very simple one layer architecture for the LSTM. However, more complicated architectures can be applied and compared. Also, such concepts as transfer learning and end-to-end learning can potentially help to reduce the multiple step procedure to a single model, where the input is given "as is" and the automatic features are generated during the training. With the deep learning gaining popularity, the whole field of machine learning is shifting from the culture of manual feature extraction to the culture of seeking the architecture for each domain. Then, the model trained on large datasets can be reused for the same task but for the different, smaller, datasets. This is active line of the research with a few examples outlined in [Long 16, Rusu 16]. We believe that such domains as BPM and Fraud detection can benefit from using their "own" pre-trained models that community could use.

It is thrilling to see how machine learning becomes more and more powerful and we believe that the future research can help to improve all the shortcomings of the current one.

Appendix

Source code

The implementation of all the methods described in the paper [Leon 16] and the code for the exploratory analysis is available at the public repository at: www.github/Generative-Models-in-Classification.

Supplementary material for the article "Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes" [Leon 15] is available here: www.github/Sequence-Encodings-for-Predictive-Monitoring.

Acknowledgements

My doctoral studies have been funded by the European Social Fund's Doctoral Studies Internalization Program (DoRa) carried out by Archimedes Foundation and Software Technology and Applications Competence Center (STACC). The preparation of this thesis has been also supported by Horizon 2020 SoBigData project.

I would like to thank my supervisors Marlon Dumas and Jaak Vilo for their valuable advices throughout all these years. They believed in me. Without them I would not start nor finish my PhD studies.

I am very thankful to my reviewers Alessandro Sperduti and Jaakko Hollmén, for their insightful suggestions and positive feedback.

This work would not be in a readable format without my friends who found time to go through it and provide their comments and corrections. Thank you, Irene Teinemaa and Elena Sügis. I am highly grateful to Ilya Kuzovkin, who helped in so many ways that it will not be possible to list it all here.

My doctoral studies have been a steep learning curve, where I was introduced by Konstantin Tretyakov to a new fascinating area of research – Machine Learning. For this life-changing introduction I would like to thank him.

It is said that the environment around shapes person's thoughts, and I was very lucky to be around smart, intelligent people, who inspired me a lot. I would like to express my gratitude to all my co-workers in STACC, BIIT people and my fellows PhD students in Institute of Computer Science at the University of Tartu. Our office kitchen was a place where most of the new ideas were born.

Teamwork is not something that can be easily found and I appreciate the collaboration with my co-authors of the papers.

I am also grateful for my friends. Some of them are far away, like Anastassia Žegulova, some of them are closer, like Teele Tamme ja Liisa Seppel, but all of them have been supportive and helpful on my long way to a PhD degree.

Finally, I am grateful for the support from my family: mom, dad and my sister as well as my grandfather, who checked the progress of my ”dissertation“ every time he called. Unfortunately, he can not see me finishing it, but I think it would make him very happy.

Generatiivsete mudelite kasutamine staatiliste ja jadatunnuste kombineerimiseks klassifitseerimise eesmärgil

Klassifitseerimismudeli täpsus on otseses sõltuvuses treenimise jaoks valitud tunnustest. Sobivate tunnuste otsimine ja läbiproovimine on väljakutset pakkuv ülesanne mudeli treenimisprotsessis. Antud protsess võib osutuda veelgi keerulisemaks kui andmetes esinevad koos nii aegread või jadad kui ka staatilised tunnused, mis ajas ei muutu. Näiteks võib tuua kasutajate profiiliandmed koos nende tarkvara kasutusajalooga mingi perioodi jooksul, äriprotsessid koos nende metakirjeldusega, või liigutuste tagajärjel tekkivate närvisignaalide andmestiku, kus on olulised ka katseisikuid iseloomustavad faktorid. Enamik klassikalistest masinõppe algoritmidest kasutab kas ainult staatilisi tunnuseid või ainult jadatunnuseid, kuid mitte mõlemaid. Praktikas on aga klassifitseerimisprobleemide lahendamisel tihtipeale kasulikum kaasata mõlemad tunnuste liigid.

Käesolev väitekiri uurib, kuidas ühendada nii staatilisi kui ka jadatunnuseid klassifitseerija täpsuse tõstmiseks, ning pakub välja hübriidalgoritmi, mis integreerib jadatunnuste põhjal treenitud generatiivse mudeli väljundi diskriminatiivsesse mudelisse. Põhiidee seisneb selles, et kasutada generatiivseid mudeleid nagu varjatud Markovi mudelid või rekurrentsed närvivõrgud jadatunnuste alusel mudeli treenimiseks ning seejärel kasutada treenitud mudelite väljundeid uute tunnustena koos staatiliste tunnustega lõppklassifitseerijas. Antud töö põhimõte baseerub eeldusel, et generatiivsete ja diskriminatiivsete mudelite kombineerimine aitab saada lisainformatsiooni, mis on oluline klasside eristamise jaoks. Käesolevas töös võrreldakse hübriidmudelit laiemalt levinud ansambelõppega ning mudelitega, mis õpivad ainult staatiliste või ainult jadatunnuste põhjal.

Tihtipeale on tegu ka selliste klassifitseerimisülesandega, kus on vaja olemasolevate andmete põhjal ennustada andmeklassi palju varem kui perioodi lõpus. Näiteks võib tuua tulevaste haiguste enneaegset riski tuvastamist patsiendi haigusloo põhjal või petturi avastamist tema käitumise järgi, enne kui petuskeem on teostatud. Paraku, mida varajasem ennustus, seda vähem andmeid on selle tegemiseks saadaval. Seega on oluline kaasata kõik olemasolevad allikad, kasutades nii staatilisi kui ka jadatunnuseid. Antud töö pakub välja raamistiku, mis võimaldab teha klassi ennustusi varakult, ehk andmejadade alguses.

Pakutud lähenemist on rakendatud kolmes juhtumiuuringus erinevatest valdkondadest. Kahe esimese juhtumiuuringu põhjal on tegu varajase ennustusega, kolmas juhtumiuuring tegeleb klassifitseerimisega täielikel mitmemõõtmelistel jadadel.

Esimeses juhtumiuuringus ehitatakse mudel eristamaks hälvega äriprotsesse regulaarsetest, kasutades selleks mitte ainult äriprotsesse, vaid lisaks protsessidele ka nendega kaasnevat infot. Põhieesmärgiks on klassifitseerida alles algusjärgus olevat protsessi võimalikult varakult ja kõrge täpsusega, kasutades ainult üksikute esimeste sündmuste realiseerimiseandmeid. Selle töö tulemus erineb teistest sarnastest töödest selle poolest, et väljapakutud klassifitseerija arvestab nii informatsiooni protsessi kohta kui ka lisainfot iga realiseeritava sündmuse kohta. Juhtumiuuringus võrreldatakse erinevaid meetodeid selliste andmete käsitlemiseks klassifitseerimise eesmärgil. Empiiriline analüüs näitas, et kasutades pakutud metoodikat saab eristada hälvega ja hälveta protsesse 5 esimese sündmuse realiseerimisel 90% täpsusega (mõõdetud AUC meetrikaga).

Teises juhtumiuuringus tegu on petturite tuvastamisega Skype sotsiaalvõrgustikus, kasutades selleks kasutajate profiiliandmeid, lokaalset käitumist ning sotsiaalvõrgustiku mustreid. Pakutud hübriidmudel võimaldab kõik need allikad hõlmata ühe mudeli alla. Treenitud mudelit rakendatakse, et tuvastada salajaste, pikema perioodi vältel tegutsenud pettureid, keda ei olnud kiiresti blokeeritud. Hübriidmudel saavutab 68% õigepositiivsete tuvastamismäära (ing. k. True Positive Rate). kui on fikseeritud 5% vale-postiiivsete tuvastamismäär (ing.k. False Positive Rate).

Kolmandas juhtumiuuringus klassifitseeritakse katseisikute kujutletavaid liigutusi ajusignaalide põhjal, kus mudelisse kaasatakse nii ajusignaalid kuid ka Fourier' teisenduse abil saadud staatilised tunnused. Antud juhul uuritakse hübriidmudelit lähemalt ning võrreldakse ansambelõppega ja teiste bassmudelitega, eksperimenteeritakse ka generatiivse mudeli valikuga. Kokku võrreldakse empiiriliselt 11 mudelit võrdletud erinevate andmestike põhjal. Lisaks uuritakse sünteetilise andmestiku abil andmete parameetrite mõju mudeli tulemusele. Tulemused näitavad, et hübriidmudel suudab üldjuhul tõsta klassifitseerimistäpsust võrreldes teiste mudeliliikidega ning parima tulemuse saavutab hübriid, kus generatiivseks mudeliks on rekurrentsed närvivõrgud pika lühiajalise mälu sõlmega (Recurrent Neural Networks with Long Short-term Memory unit).

Bibliography

[Aals 10] W.M.P. van der Aalst, M. Pesic, and M. Song. **Beyond Process Mining: From the Past to Present and Future**. In: Barbara Pernici, editor, *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings*, pp. 38–52, Springer, cited on pages: 82

[Aals 11] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. **Time prediction based on process mining**. *Inf. Syst.*, Vol. 36, No. 2, pp. 450–475, cited on pages: 82

[Albr 11] W Steve Albrecht, Chad O Albrecht, Conan C Albrecht, and Mark F Zimbelman. *Fraud examination.* Cengage Learning, cited on pages: 104

[Alla 08] Cyril Allauzen, Mehryar Mohri, and Ameet Talwalkar. **Sequence kernels for predicting protein essentiality**. In: *Proceedings of the 25th international conference on Machine learning*, pp. 9–16, ACM, cited on pages: 74

[Alpa 14] Ethem Alpaydin. *Introduction to machine learning.* MIT press, cited on pages: 20

[Anto 15] Alessandro Antonucci, Mauro Scanagatta, Denis Deratani Mauá, and Cassio Polpo de Campos. **Early Classification of Time Series by Hidden Markov Models with Set-Valued Parameters**. cited on pages: 75, 76

[Bagn 12] Anthony Bagnall, Luke M Davis, Jon Hills, and Jason Lines. **Transformation Based Ensembles for Time Series Classification.** In: *SDM*, pp. 307–318, SIAM, cited on pages: 126

[Bayd 15] Mustafa Gokce Baydogan and George Runger. **Learning a symbolic representation for multivariate time series classifica-**

tion. *Data Mining and Knowledge Discovery*, Vol. 29, No. 2, pp. 400–422, cited on pages: 126

[Ben 03] Asa Ben-Hur and Douglas Brutlag. **Remote homology detection: a motif based approach**. *Bioinformatics*, Vol. 19, No. suppl 1, pp. i26–i33, cited on pages: 74

[Beng 13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. **Representation learning: A review and new perspectives**. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35, No. 8, pp. 1798–1828, cited on pages: 14

[Bish 06] Christopher M Bishop. **Pattern Recognition**. *Machine Learning*, cited on pages: 41

[Bolt 02] Richard J Bolton and David J Hand. **Statistical fraud detection: A review**. *Statistical science*, pp. 235–249, cited on pages: 28

[Brad 97] Andrew P Bradley. **The use of the area under the ROC curve in the evaluation of machine learning algorithms**. *Pattern recognition*, Vol. 30, No. 7, pp. 1145–1159, cited on pages: 26

[Brei 01] Leo Breiman. **Random forests**. *Machine learning*, Vol. 45, No. 1, pp. 5–32, cited on pages: 41, 42, 58

[Brin 12] Sergey Brin and Lawrence Page. **Reprint of: The anatomy of a large-scale hypertextual web search engine**. *Computer networks*, Vol. 56, No. 18, pp. 3825–3833, cited on pages: 66

[Cao 95] Jun Cao, M. Ahmadi, and M. Shridhar. **Recognition of handwritten numerals with multiple feature and multistage classifier**. *Pattern Recognition*, Vol. 28, No. 2, pp. 153–160, cited on pages: 103

[Caru 06] Rich Caruana and Alexandru Niculescu-Mizil. **An empirical comparison of supervised learning algorithms**. In: *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168, ACM, cited on pages: 42

[Cast 05] Malú Castellanos, Norman Salazar, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. **Predictive Business Operations Management**. In: *Proc. of DNIS*, pp. 1–14, Springer, cited on pages: 83

[Chen 15] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. **The UCR Time Series Classification Archive**. July 2015. cited on pages: 125, 126

[Chir 05] Paul-Alexandru Chirita, Jörg Diederich, and Wolfgang Nejdl. **MailRank: using ranking for spam detection**. In: *Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 373–380, ACM, cited on pages: 65

[Chui 92] Charles K Chui. **Wavelets: a tutorial in theory and applications**. *Wavelet Analysis and its Applications, San Diego, CA: Academic Press,| c1992, edited by Chui, Charles K.*, Vol. 1, cited on pages: 64

[Cole 15] C. Coleman Coleman. **Internet Crime report**. 2015. cited on pages: 101, 104

[Conf 15] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. **A recommendation system for predicting risks across multiple business process instances**. *Decision Support Systems*, Vol. 69, pp. 1–19, cited on pages: 81, 83, 100

[Cove 65] Thomas M Cover. **Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition**. *IEEE transactions on electronic computers*, No. 3, pp. 326–334, cited on pages: 73

[Da S 10] Giovanni Da San Martino and Alessandro Sperduti. **Mining structured data**. *IEEE Computational Intelligence Magazine*, Vol. 5, No. 1, pp. 42–49, cited on pages: 73

[Dach 15] Asma Dachraoui, Alexis Bondu, and Antoine Cornuéjols. **Early classification of time series as a non myopic sequential decision making problem**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 433–447, Springer, cited on pages: 76

[Daup 15] Yann N Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. **RMSProp and equilibrated adaptive learning rates for non-convex optimization**. *arXiv preprint arXiv:1502.04390*, cited on pages: 115

[Diet 00] Thomas G Dietterich. **Ensemble methods in machine learning**. In: *Multiple classifier systems*, pp. 1–15, Springer, cited on pages: 67, 70, 115

[Dong 11] B.F. van Dongen. **Real-life event logs - Hospital log**. cited on pages: 33, 82

[Dono 15] David Donoho. **50 years of Data Science**. http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf, 2015. cited on pages: 20

[Duma 16] Ekrem Duman and Yusuf Sahin. **A Comparison of Classification Models on Credit Card Fraud Detection with respect to Cost-Based Performance Metrics**. *Use of Risk Analysis in Computer-Aided Persuasion. NATO Science for Peace and Security Series E: Human and Societal Dynamics*, Vol. 88, pp. 88–99, cited on pages: 28

[Efro 97] Bradley Efron and Robert Tibshirani. **Improvements on cross-validation: the 632+ bootstrap method**. *Journal of the American Statistical Association*, Vol. 92, No. 438, pp. 548–560, cited on pages: 26

[Farv 11] Hamid Farvaresh and Mohammad Mehdi Sepehri. **A data mining framework for detecting subscription fraud in telecommunication**. *Engineering Applications of Artificial Intelligence*, Vol. 24, No. 1, pp. 182–194, cited on pages: 103

[Fawc 06] Tom Fawcett. **An introduction to ROC analysis**. *Pattern recognition letters*, Vol. 27, No. 8, pp. 861–874, cited on pages: 23

[Fawc 97a] T Fawcett and F Provost. **Adaptive fraud detection**. *Data mining and knowledge discovery*, Vol. 316, pp. 291–316, cited on pages: 103

[Fawc 97b] Tom Fawcett and Foster Provost. **Adaptive fraud detection**. *Data mining and knowledge discovery*, Vol. 1, No. 3, pp. 291–316, cited on pages: 28

[Feld 13] Zohar Feldman, Fabiana Fournier, Rod Franklin, and Andreas Metzger. **Proactive event processing in action: a case study on the proactive management of transport processes**. In: *Proc. of DEBS*, pp. 97–106, ACM, cited on pages: 83

[Foli 12] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. **Discovering Context-Aware Models for Predicting Business Process Performances**. In: *Proc. of OTM Confederated Conferences*, pp. 287–304, Springer, cited on pages: 82

[Fran 16] et al. François Chollet. **Keras**. https://github.com/fchollet/keras, cited on pages: 115

[Freu 95] Yoav Freund and Robert E Schapire. **A desicion-theoretic generalization of on-line learning and an application to boosting**. In: *European conference on computational learning theory*, pp. 23–37, Springer, cited on pages: 57

[Frie 01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Vol. 1, Springer series in statistics Springer, Berlin, cited on pages: 24

[Gart 03] Thomas Gärtner. **A survey of kernels for structured data**. *ACM SIGKDD Explorations Newsletter*, Vol. 5, No. 1, pp. 49–58, cited on pages: 73, 74

[Gold 10] Anthony Goldbloom. **Kaggle**. www.kaggle.com, 2010. cited on pages: 56

[Gold 12] Moisés Goldszmidt. **Finding Soon-to-Fail Disks in a Haystack**. In: *Proc. of HotStorage*, USENIX, cited on pages: 107, 112

[Good 16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. **Deep Learning**. 2016. cited on pages: 43

[Grav 13] Alan Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. **Speech recognition with deep recurrent neural networks**. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649, IEEE, cited on pages: 49, 114

[Hami 94] James Douglas Hamilton. *Time series analysis.* Vol. 2, Princeton university press Princeton, cited on pages: 125

[Han 11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques.* Elsevier, cited on pages: 20, 22, 31

[Hata 13] Nima Hatami and Camelia Chira. **Classifiers with a reject option for early time-series classification**. In: *Computational*

Intelligence and Ensemble Learning (CIEL), 2013 IEEE Symposium on, pp. 9–16, IEEE, cited on pages: 75, 76

[He 15] Guoliang He, Yong Duan, Rong Peng, Xiaoyuan Jing, Tieyun Qian, and Lingling Wang. **Early classification on multivariate time series**. *Neurocomputing*, Vol. 149, pp. 777–787, cited on pages: 76

[Heit 09] Geremy Heitz, Stephen Gould, Ashutosh Saxena, and Daphne Koller. **Cascaded classification models: Combining models for holistic scene understanding**. In: *Advances in Neural Information Processing Systems*, pp. 641–648, cited on pages: 68, 72

[Hila 08] Constantinos S. Hilas and Paris As. Mastorocostas. **An application of supervised and unsupervised learning approaches to telecommunications fraud detection**. *Knowledge-Based Systems*, Vol. 21, No. 7, pp. 721–726, cited on pages: 103

[Hila 09] Constantinos S. Hilas. **Designing an expert system for fraud detection in private telecommunications networks**. *Expert Systems with Applications*, Vol. 36, No. 9, pp. 11559–11569, cited on pages: 103

[Hint 06] Geoffrey E Hinton and Ruslan R Salakhutdinov. **Reducing the dimensionality of data with neural networks**. *Science*, Vol. 313, No. 5786, pp. 504–507, cited on pages: 14

[Ho 94] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. **Decision combination in multiple classifier systems**. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 16, No. 1, pp. 66–75, cited on pages: 103

[Hoch 97] Sepp Hochreiter and Jürgen Schmidhuber. **Long short-term memory**. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, cited on pages: 49

[Huan 11] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. **Adversarial machine learning**. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, ACM, cited on pages: 104

[Huan 13] Junxian Huang, Yinglian Xie, Fang Yu, Qifa Ke, Martin Abadi, Eliot Gillum, and Z Morley Mao. **Socialwatch: detection of online service abuse via large-scale social graphs**.

In: *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 143–148, ACM, cited on pages: 65, 102

[Ishi 00] KATSUHIKO ISHIGURO, HIROSHI SAWADA, AND HITOSHI SAKANO. **Multi-class boosting for early classification of sequences**. *Statistics*, Vol. 28, No. 2, pp. 337–407, cited on pages: 76

[Jaak 00] TOMMI JAAKKOLA, MARK DIEKHANS, AND DAVID HAUSSLER. **A discriminative framework for detecting remote protein homologies**. *Journal of computational biology*, Vol. 7, No. 1-2, pp. 95–114, cited on pages: 55, 73, 74

[Jaak 99] TOMMI S JAAKKOLA, DAVID HAUSSLER, ET AL. **Exploiting generative models in discriminative classifiers**. *Advances in neural information processing systems*, pp. 487–493, cited on pages: 73, 74, 120

[Jaco 91] ROBERT A JACOBS, MICHAEL I JORDAN, STEVEN J NOWLAN, AND GEOFFREY E HINTON. **Adaptive mixtures of local experts**. *Neural computation*, Vol. 3, No. 1, pp. 79–87, cited on pages: 71

[Jord 02] A JORDAN. **On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes**. *Advances in neural information processing systems*, Vol. 14, p. 841, cited on pages: 39

[Kang 12] BOKYOUNG KANG, DONGSOO KIM, AND SUK-HO KANG. **Real-time Business Process Monitoring Method for Prediction of Abnormal Termination Using KNNI-based LOF Prediction**. *Expert Syst. Appl.*, cited on pages: 83

[Keog 11] EAMONN KEOGH AND ABDULLAH MUEEN. **Curse of dimensionality**. In: *Encyclopedia of Machine Learning*, pp. 257–258, Springer, cited on pages: 116

[Kitt 98] JOSEF KITTLER, MOHAMAD HATEF, ROBERT PW DUIN, AND JIRI MATAS. **On combining classifiers**. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 20, No. 3, pp. 226–239, cited on pages: 103

[Koha 95] RON KOHAVI ET AL. **A study of cross-validation and bootstrap for accuracy estimation and model selection**. In: *Ijcai*, pp. 1137–1145, cited on pages: 26

[Kots 06] Sotiris Kotsiantis and Dimitris Kanellopoulos. **Discretization techniques: A recent survey**. *GESTS International Transactions on Computer Science and Engineering*, Vol. 32, No. 1, pp. 47–58, cited on pages: 63

[Krak 16] Viktoriya Krakovna and Finale Doshi-Velez. **Increasing the interpretability of recurrent neural networks using hidden markov models**. *arXiv preprint arXiv:1606.05320*, cited on pages: 134

[Kriz 12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **Imagenet classification with deep convolutional neural networks**. In: *Advances in neural information processing systems*, pp. 1097–1105, cited on pages: 14, 49

[Ku 07] Yungchang Ku, Yuchi Chen, and Chaochang Chiu. **A Proposed Data Mining Approach for Internet Auction Fraud Detection**. pp. 238–243, cited on pages: 103

[Kunc 03] Ludmila I Kuncheva and Christopher J Whitaker. **Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy**. *Machine learning*, Vol. 51, No. 2, pp. 181–207, cited on pages: 58

[Kunc 04] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms.* John Wiley & Sons, cited on pages: 70

[Lal 04] Thomas N Lal, Thilo Hinterberger, Guido Widman, Michael Schröder, N Jeremy Hill, Wolfgang Rosenstiel, Christian E Elger, Niels Birbaumer, and Bernhard Schölkopf. **Methods towards invasive human brain computer interfaces**. In: *Advances in neural information processing systems*, pp. 737–744, cited on pages: 30, 126

[Lang 14] Martin Längkvist, Lars Karlsson, and Amy Loutfi. **A review of unsupervised feature learning and deep learning for time-series modeling**. *Pattern Recognition Letters*, Vol. 42, pp. 11–24, cited on pages: 49

[Lebe 15] Sergei Lebedev and et al. **hmmlearn**. https://github.com/hmmlearn/hmmlearn, cited on pages: 115

[Leon 12] Anna Leontjeva, Konstantin Tretyakov, Jaak Vilo, and Taavi Tamkivi. **Fraud detection: Methods of analysis for**

hypergraph data. In: *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pp. 1060–1064, IEEE, cited on pages: 29

[Leon 13] Anna Leontjeva, Moises Goldszmidt, Yinglian Xie, Fang Yu, and Martín Abadi. **Early security classification of skype users via machine learning**. In: *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pp. 35–44, ACM, cited on pages: 19, 29, 35

[Leon 15] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. **Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes**. In: *Business Process Management*, pp. 297–313, Springer, cited on pages: 19, 28, 33, 137

[Leon 16] Anna Leontjeva and Ilja Kuzovkin. **Combining Static and Dynamic Features for Multivariate Sequence Classification**. In: *2016 International Conference on Data Science and Advanced Analytics*, IEEE, cited on pages: 19, 137

[Lest 05] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. **A hybrid discriminative/generative approach for modeling human activities**. cited on pages: 72

[Li 10] Congcong Li, Adarsh Kowdle, Ashutosh Saxena, and Tsuhan Chen. **Towards holistic scene understanding: Feedback enabled cascaded classification models**. In: *Advances in Neural Information Processing Systems*, pp. 1351–1359, cited on pages: 72

[Lin 03] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. **A symbolic representation of time series, with implications for streaming algorithms**. In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, ACM, cited on pages: 15

[Lin 07] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. **Experiencing SAX: a novel symbolic representation of time series**. *Data Mining and knowledge discovery*, Vol. 15, No. 2, pp. 107–144, cited on pages: 63

[Lin 12] JESSICA LIN, SHERI WILLIAMSON, KIRK BORNE, AND DAVID DEBARR. **Pattern recognition in time series**. *Advances in Machine Learning and Data Mining for Astronomy*, Vol. 1, pp. 617–645, cited on pages: 63

[Lipt 15] ZACHARY C LIPTON, JOHN BERKOWITZ, AND CHARLES ELKAN. **A critical review of recurrent neural networks for sequence learning**. *arXiv preprint arXiv:1506.00019*, cited on pages: 49, 51, 52

[Loh 11] WEI-YIN LOH. **Classification and regression trees**. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 1, No. 1, pp. 14–23, cited on pages: 42

[Long 16] MINGSHENG LONG, JIANMIN WANG, AND MICHAEL I JORDAN. **Unsupervised Domain Adaptation with Residual Transfer Networks**. *arXiv preprint arXiv:1602.04433*, cited on pages: 135

[MacK 97] DAVID JC MACKAY. **Ensemble Learning for Hidden Markov Models**. *Unpublished manuscript, http://wol. ra. phy. cam. ac. uk/-mackay*, cited on pages: 46

[Magg 14] FABRIZIO MARIA MAGGI, CHIARA DI FRANCESCOMARINO, MARLON DUMAS, AND CHIARA GHIDINI. **Predictive Monitoring of Business Processes**. In: *Proc. of CAiSE*, pp. 457–472, Springer, cited on pages: 81, 83

[Maso 14] SAEED MASOUDNIA AND REZA EBRAHIMPOUR. **Mixture of experts: a literature survey**. *Artificial Intelligence Review*, Vol. 42, No. 2, pp. 275–293, cited on pages: 71

[McDo 15] NANCY K. ET AL. MCDONNELL. **2015 AFP Payments Fraud and Control Survey**. Tech. Rep., The Association for Financial Professionals, cited on pages: 101

[Mcgl 09] MARY MCGLOHON, STEPHEN BAY, MARKUS G ANDERLE, DAVID M STEIER, AND CHRISTOS FALOUTSOS. **SNARE : A Link Analytic System for Graph Labeling and Risk Detection**. pp. 1265–1273, cited on pages: 103

[Merc 09] JAMES MERCER. **Functions of positive and negative type, and their connection with the theory of integral equations**. *Philosophical transactions of the royal society of London. Series A, contain-*

ing papers of a mathematical or physical character, Vol. 209, pp. 415–446, cited on pages: 73

[Metz 12] Andreas Metzger, Rod Franklin, and Yagil Engel. **Predictive Monitoring of Heterogeneous Service-Oriented Business Networks: The Transport and Logistics Case**. In: *Proc. of SRII Global Conference*, IEEE, cited on pages: 83

[Mnih 13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. **Playing atari with deep reinforcement learning**. *arXiv preprint arXiv:1312.5602*, cited on pages: 21

[Murp 12] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, cited on pages: 20, 22, 39, 41, 46, 71, 73

[Negr 08] Pablo Negri, Xavier Clady, Shehzad Muhammad Hanif, and Lionel Prevost. **A cascade of boosted generative and discriminative classifiers for vehicle detection**. *EURASIP Journal on Advances in Signal Processing*, Vol. 2008, p. 136, cited on pages: 71, 72

[Nels 10] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, and JD Tygar. **Classifier evasion: Models and open problems**. In: *International Workshop on Privacy and Security Issues in Data Mining and Machine Learning*, pp. 92–98, Springer, cited on pages: 104

[Nowl 90] Steven J Nowlan and Geoffrey E Hinton. **Evaluation of Adaptive Mixtures of Competing Experts.** In: *NIPS*, pp. 774–780, cited on pages: 71

[Page 99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. **The PageRank citation ranking: bringing order to the web.** cited on pages: 65

[Parr 13] Nathan Parrish, Hyrum S Anderson, Maya R Gupta, and Dun-Yu Hsiao. **Classifying with confidence from incomplete information.** *Journal of Machine Learning Research*, Vol. 14, No. 1, pp. 3561–3589, cited on pages: 76

[Pate 13] Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari, and Joaquim Celestino JúNior. **An intrusion detection and prevention system in cloud computing: A systematic review**.

Journal of network and computer applications, Vol. 36, No. 1, pp. 25–41, cited on pages: 101

[Pedr 11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. **Scikit-learn: Machine Learning in Python**. *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, cited on pages: 115

[Pika 13] Anastasiia Pika, WilM.P. Aalst, ColinJ. Fidge, ArthurH.M. Hofstede, and MoeT. Wynn. **Predicting Deadline Transgressions Using Event Logs**. In: *Proc. of BPM Workshops*, pp. 211–216, Springer, cited on pages: 83

[Pnue 77] Amir Pnueli. **The Temporal Logic of Programs**. In: *Proc. of FOCS*, pp. 46–57, IEEE, cited on pages: 90

[Pola 14] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. **Data-aware remaining time prediction of business process instances**. In: *Neural Networks (IJCNN), 2014 International Joint Conference on*, pp. 816–823, IEEE, cited on pages: 82

[Rabi 89] Lawrence Rabiner. **A tutorial on hidden Markov models and selected applications in speech recognition**. *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257–286, cited on pages: 43

[Rayk 10] Vikas C Raykar, Balaji Krishnapuram, and Shipeng Yu. **Designing efficient cascaded classifiers: tradeoff between accuracy and cost**. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 853–860, ACM, cited on pages: 71

[Reyn 15] Douglas Reynolds. **Gaussian mixture models**. *Encyclopedia of Biometrics*, pp. 827–832, cited on pages: 44

[Ribe 16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. **" Why Should I Trust You?": Explaining the Predictions of Any Classifier**. *arXiv preprint arXiv:1602.04938*, cited on pages: 134

[Ries 10] Kaspar Riesen and Horst Bunke. *Graph classification and clustering based on vector space embedding.* World Scientific Publishing Co., Inc., cited on pages: 15

[Rogg 13] Andreas Rogge-Solti and Mathias Weske. **Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays**. In: *Proc. of ICSOC*, pp. 389–403, Springer, cited on pages: 82

[Rume 88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. **Learning representations by back-propagating errors**. *Cognitive modeling*, Vol. 5, No. 3, p. 1, cited on pages: 50

[Rusu 16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. **Progressive neural networks**. *arXiv preprint arXiv:1606.04671*, cited on pages: 135

[Schr 05] Michael Schröder, Thilo Hinterberger, Thomas Navin Lal, Guido Widman, and Niels Birbaumer. **BCI III Competition Results**. http://www.bbci.de/competition/iii/results/index.html, 2005. cited on pages: 126, 127

[Sewe 08] Martin Sewell. **Ensemble learning**. *RN*, Vol. 11, No. 02, cited on pages: 56

[Snoe 12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. **Practical Bayesian optimization of machine learning algorithms**. In: *Advances in neural information processing systems*, pp. 2951–2959, cited on pages: 124

[Sonn 05] Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. **Learning interpretable SVMs for biological sequence classification**. In: *Annual International Conference on Research in Computational Molecular Biology*, pp. 389–407, Springer, cited on pages: 73

[Sriv 15] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. **Unsupervised learning of video representations using LSTMs**. *arXiv preprint arXiv:1502.04681*, cited on pages: 49

[Suri 13a] Suriadi Suriadi, Chun Ouyang, WilM.P. Aalst, and ArthurH.M. Hofstede. **Root Cause Analysis with Enriched**

Process Logs. In: *Proc. of BPM Workshops*, pp. 174–186, Springer, cited on pages: 83

[Suri 13b] SURIADI SURIADI, MOE THANDAR WYNN, CHUN OUYANG, ARTHUR H. M. TER HOFSTEDE, AND NIENKE J. VAN DIJK. **Understanding Process Behaviours in a Large Insurance Company in Australia: A Case Study**. In: *Proc. of CAiSE*, pp. 449–464, Springer, cited on pages: 82

[Suts 13] ILYA SUTSKEVER. *Training recurrent neural networks.* PhD thesis, University of Toronto, cited on pages: 48

[Suts 14] ILYA SUTSKEVER, ORIOL VINYALS, AND QUOC V LE. **Sequence to sequence learning with neural networks**. In: *Advances in neural information processing systems*, pp. 3104–3112, cited on pages: 49

[Tave 16] ROMAIN TAVENARD AND SIMON MALINOWSKI. **Cost-Aware Early Classification of Time Series**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 632–647, Springer, cited on pages: 76, 135

[Teh 12] YEE WHYE TEH, MICHAEL I JORDAN, MATTHEW J BEAL, AND DAVID M BLEI. **Hierarchical dirichlet processes**. *Journal of the american statistical association*, cited on pages: 46

[Tein 16] IRENE TEINEMAA, MARLON DUMAS, FABRIZIO MARIA MAGGI, AND CHIARA DI FRANCESCOMARINO. **Predictive Business Process Monitoring with Structured and Unstructured Data**. In: *International Conference on Business Process Management*, pp. 401–417, Springer, cited on pages: 84, 100

[Ting 99] KAI MING TING AND IAN H WITTEN. **Issues in stacked generalization**. *J. Artif. Intell. Res.(JAIR)*, Vol. 10, pp. 271–289, cited on pages: 59

[Tree 16] HENRI TREES, ANNA LEONTJEVA, AND MARLON DUMAS. *Predictive Monitoring of Multi-level Processes.* Master's thesis, University of tartu, Estonia, cited on pages: 84, 100

[Vapn 98] VLADIMIR NAUMOVICH VAPNIK AND VLAMIMIR VAPNIK. *Statistical learning theory.* Vol. 1, Wiley New York, cited on pages: 39

[Vere 15] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Chiara Di Francescomarino. **Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring**. cited on pages: 84, 100

[Viol 01] Paul Viola and Michael Jones. **Rapid object detection using a boosted cascade of simple features**. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, pp. I–511, IEEE, cited on pages: 71, 72, 73, 103

[Vite 67] Andrew Viterbi. **Error bounds for convolutional codes and an asymptotically optimum decoding algorithm**. *IEEE transactions on Information Theory*, Vol. 13, No. 2, pp. 260–269, cited on pages: 48

[Wang 08] Jyun-Cheng Wang and Chui-Chen Chiu. **Recommending trusted online auction sellers using social network analysis**. *Expert Systems with Applications*, Vol. 34, No. 3, pp. 1666–1679, cited on pages: 103

[Watt 98] Duncan J Watts and Steven H Strogatz. **Collective dynamics of 'small-world'networks**. *nature*, Vol. 393, No. 6684, pp. 440–442, cited on pages: 36, 66

[Wei 13] Wei Wei, Jinjiu Li, Longbing Cao, Yuming Ou, and Jiahang Chen. **Effective detection of sophisticated online banking fraud on extremely imbalanced data**. *World Wide Web*, Vol. 16, No. 4, pp. 449–475, cited on pages: 103

[Wind 14] David Kofoed Wind and Ole Winther. **Model Selection in Data Analysis Competitions.** In: *MetaSel@ ECAI*, pp. 55–60, cited on pages: 56

[Wolp 92] David H Wolpert. **Stacked generalization**. *Neural networks*, Vol. 5, No. 2, pp. 241–259, cited on pages: 115

[Xing 10] Zhengzheng Xing, Jian Pei, and Eamonn J. Keogh. **A brief survey on sequence classification**. *SIGKDD Explorations*, Vol. 12, No. 1, pp. 40–48, cited on pages: 15, 32, 73, 81

[Xing 11] Zhengzheng Xing, Jian Pei, S Yu Philip, and Ke Wang. **Extracting Interpretable Features for Early Classification on Time Series.** SIAM, cited on pages: 75

[Xing 12] Zhengzheng Xing, Jian Pei, and S Yu Philip. **Early classification on time series**. *Knowledge and information systems*, Vol. 31, No. 1, pp. 105–127, cited on pages: 75, 76

[Yosi 14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. **How transferable are features in deep neural networks?** In: *Advances in neural information processing systems*, pp. 3320–3328, cited on pages: 54

[Zhou 12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms.* CRC press, cited on pages: 43, 55, 59, 70

Curriculum Vitae

Name:	Anna Leontjeva
Date of Birth:	14.11.1985
Citizenship:	Estonian

Education:

2010 – 2017	University of Tartu, Faculty of Mathematics and Computer Science, doctoral studies, specialty: computer science.
2008 – 2010	University of Tartu, Faculty of Mathematics and Computer Science, master studies, specialty: mathematical statistics.
2004 – 2007	University of Tartu, Faculty of Mathematics and Computer Science, bachelor studies, specialty: mathematical statistics.

Work experience:

2017 – ..	University of Tartu, junior researcher
2009 – 2017	STACC OÜ, researcher
2007 – 2010	Statistics Estonia, leading statistician

Elulookirjeldus

Nimi:	Anna Leontjeva
Sünnikuupäev:	14.11.1985
Kodakonsus:	Eesti

Haridus:

2010 – 2017	Tartu Ülikool, Matemaatika-informaatikateaduskond, doktoriõpe, eriala: informaatika.
2008 – 2010	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, eriala: matemaatiline statistika.
2004 – 2007	Tartu Ülikool, Matemaatika-informaatikateaduskond, bakalaureuseõpe, eriala: matemaatiline statistika.

Teenistuskäik:

2017 – ..	Tartu Ülikool, noorem teadur
2009 – 2017	STACC OÜ, teadur
2007 – 2010	Eesti Statistika, juhtiv statistik

List of original publications

1. Leontjeva, A., Kuzovkin, I. (2016, October). Combining Static and Dynamic Features for Multivariate Sequence Classification. In the Proceedings of the 3rd IEEE International Conference on Data Science and Advanced Analytics (pp. 21-30). (referred to as Publication III).

   Lead author. The author contributed to the idea, partly to the methods, the writing, literature review, and the formalizations.

2. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F. M. (2015, August). Complex symbolic sequence encodings for predictive monitoring of business processes. In International Conference on Business Process Management (pp. 297-313). Springer International Publishing. (referred to as Publication I).

   Lead author. The author contributed to the methods and the analysis, partly to the idea, the writing, and the formalizations.

3. Teinemaa, I., Leontjeva, A., Dumas, M., Kikas, R. (2015, August). Community-Based Prediction of Activity Change in Skype. In Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015 (pp. 73-80). ACM.

4. Leontjeva, A., Goldszmidt, M., Xie, Y., Yu, F., Abadi, M. (2013, November). Early security classification of skype users via machine learning. In Proceedings of the 2013 ACM workshop on Artificial intelligence and security (pp. 35-44). ACM. (referred to as Publication II).

Lead author. The author contributed to the methods and the analysis, literature review, partly to the idea, the writing, and the formalizations.

5. Leijen, D. A., Leontjeva, A. (2012). Linguistic and review features of peer feedback and their effect on the implementation of changes in academic writing: A corpus based investigation. Journal of Writing Research, 4(2).

   The author contributed partly to the methods and the analysis.

6. Leontjeva, A., Tretyakov, K., Vilo, J., Tamkivi, T. (2012, August). Fraud detection: Methods of analysis for hypergraph data. In Proceedings of the First IEEE/ACM International Workshop on Cybersecurity of Online Social Networks (IEEE/ACM CSOSN 2012), Istanbul. IEEE Conference Publications, (pp.1060-1064). IEEE.

   Lead author. The author contributed to the analysis, partly to the methods and writing.

20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. Töö kaitsmata.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.
43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.

44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.