A dynamic naive Bayes classifier (DNBC) is an extension of the popular probabilistic graphical model called hidden Markov model. Output variables are assumed to be statistically independent, which helps us in case of the curse of dimensionality occurring in high-dimensional space. Moreover, output variables that come from different probability distributions can be learned easier.

This thesis aims to create a parallel implementation of DNBC that can be easily executed on a computational cluster. For that reason, the algorithm will be implemented in the Scala language on top of Apache Spark.

1. Study DNBC and its possibilities of parallelism.
2. Implement DNBC that is applicable for both discrete and continuous output variables.
3. Parallelize the implementation on top of Apache Spark.
4. Evaluate the parallel implementation and compare it with the sequential implementation in terms of parallel scalability.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Paralelní implementace dynamického naivního Bayesovského klasifikátoru

Pavel Lučivňák

Katedra teoretické informatiky
Supervisor: Ing. Tomáš Šabata

April 12, 2018

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In V Praze on April 12, 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Lučivňák, Pavel. *Paralelní implementace dynamického naivního Bayesovského klasifikátoru*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Contents

List of Figures

List of Tables

Introduction

Chapter 1

Cíl práce

Chapter 2

Mathematical background

2.1 Probability density function

Probability density function (PDF) of a continuous random variable determines a likelihood that a given value occurs. Furthermore, 2.1 gives a probability of value being in interval $[a, b]$.

$$P(a \leq x \leq b) = \int_a^b \mathrm{PDF}(x)dx \tag{2.1}$$

To satisfy the property that $P(-\inf < x < \inf) = 1$, the following has to hold:

$$\int_{-\inf}^{\inf} \mathrm{PDF}(x)dx = 1. \tag{2.2}$$

2.2 Gaussian distribution

Gaussian (normal) distribution is defined by a probability density function

$$\mathrm{PDF}_{\mathcal{N}(\mu,\sigma^2)}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{2.3}$$

The distribution is defined by two parameters - mean μ and variance σ^2. Figure 2.1 provides a visualization of the probability density function.

Normal distribution is one of the most widespread distribution found in nature [4]. Imagine n independent identically distributed (i.i.d.) random variables. For large $n > 30$, both sum and average of the variables converge to normal distribution. This phenomenon occurs because of the Central Limit Theorem. See book *Introduction to Probability* [5] for a proof.

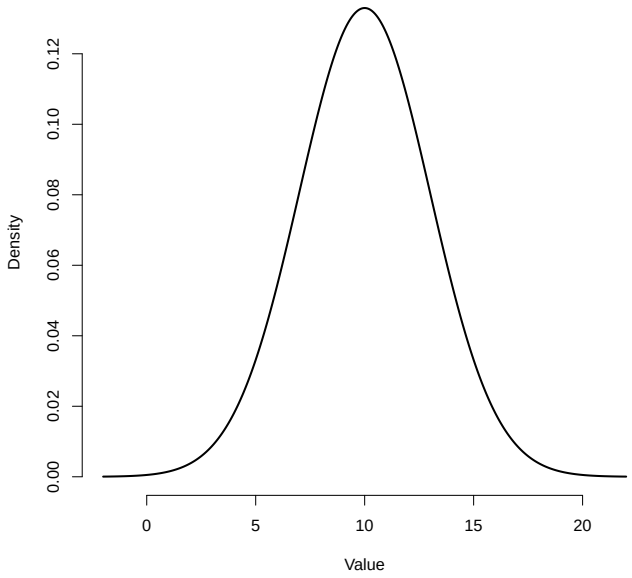

Figure 2.1: Visualization of Gaussian PDF with parameters $\mu = 10$ and $\sigma^2 = 3^2$.

2.3 Gaussian mixture distribution

Combining multiple Gaussian distributions together results in Gaussian mixture distribution. The PDF is given by 2.4, where n is number of components.

$$\mathrm{PDF}_{\mathcal{N}^*(\theta)}(x) = \sum_{i=1}^{n} w_i \mathrm{PDF}_{\mathcal{N}(\mu_i, \sigma_i^2)}(x), \tag{2.4}$$

where w_i is a weight of particular component. The following conditions have to hold:

$$w_i \geq 0, 1 \leq i \leq n, \tag{2.5}$$

$$\sum_{i=1}^{n} w_i = 1. \tag{2.6}$$

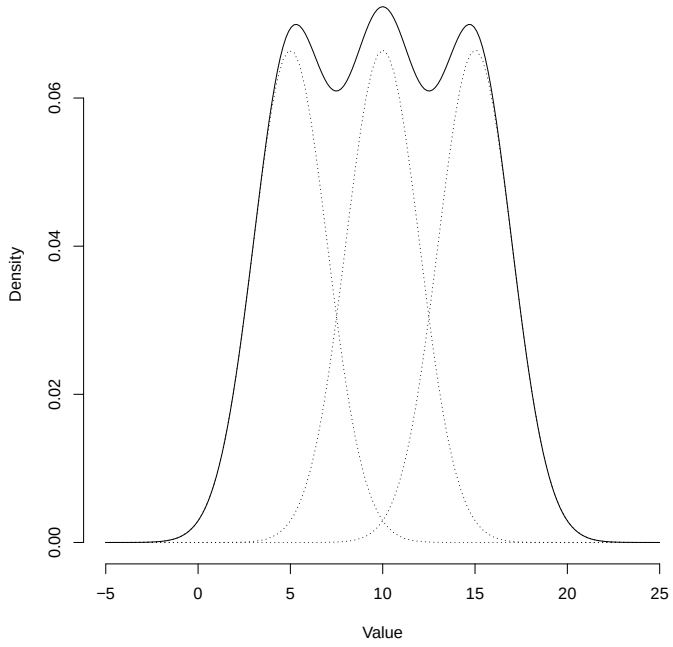

Figure 2.2: Visualization of a mixture with three components with parameters $\mu_1 = 5, \mu_2 = 10, \mu_3 = 15$ and $\sigma^2_{1,2,3} = 2^2$. Weights are $w_{1,2,3} = \frac{1}{3}$. Author: Smason79 [1].

Chapter 3

Parameter estimation

In context of this chapter, the goal of parameter estimation is to estimate parameters of a probability distribution in a way that describes given data the most.

3.1 Maximum likelihood estimation

Maximum likelihood estimation (MLE) is a technique for finding parameters of a probabilistic model that best describe behavior of a random variable. The method aims to maximize the likelihood of all the learning data.

Formally $\text{argmax}_{\theta \in \Theta} \prod_{j=1}^{M} L(x_j, \theta)$, where:

- M is number of data points,
- $X = \{x_1, x_2, \ldots, x_M\}$ is a learning set of data points,
- L is a likelihood function (defined further),
- θ describes model parameters,
- Θ is a set of all model parameters.

3.2 Discrete distribution

Here I present a straightforward way of defining the L likelihood function in case of discrete random variables. $L(x, \theta) = x_{cnt}/M$, where x_{cnt} is the number of times $x \in X$ occurs in learning data. The number of data points is M. Since the likelihood function does not depend on parameter θ, there is no expression to optimize.

There is a reason I did not choose any standard discrete probability distribution to define L. The random variable does not have to be $\mathbf{Z}$, nor $\mathbf{N}$. In fact it can be any abstract object, such as an animal. A type, where comparison

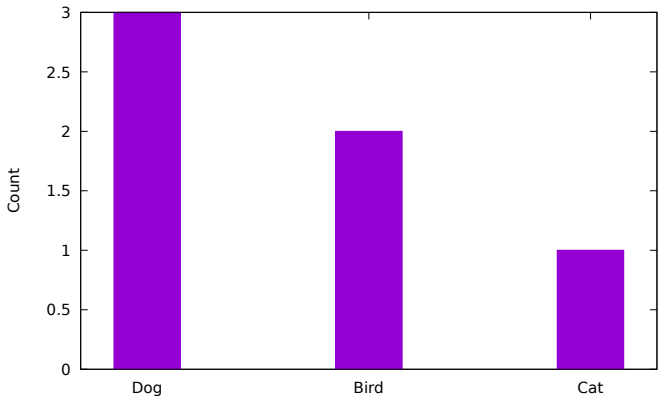

Figure 3.1: Frequency of data in discrete data set.

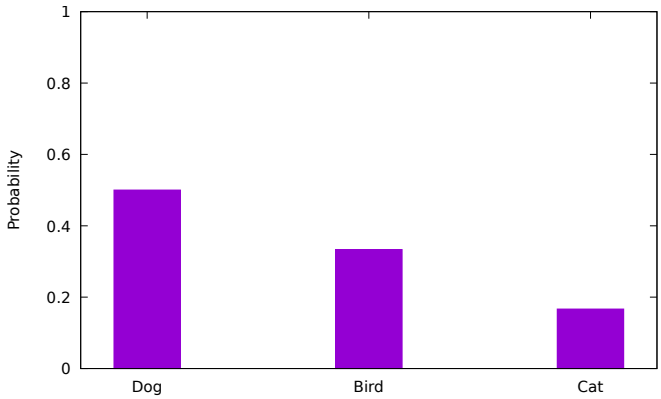

Figure 3.2: Likelihood function L of data in discrete data set. Probability is zero at undefined states.

between two objects doesn't make sense. Therefore, it wouldn't make sense to assign non zero probability to values that are not specified in learning phase.

As an example, consider the following data: {dog, bird, dog, cat, bird, dog}. Figure 3.2 shows a likelihood function L associated with the data.

3.3 Gaussian distribution

Normal distribution is defined by $\theta = \{\mu, \sigma^2\}$. The goal is to find parameter $\theta \in \Theta$, such that the product in 3.1 is maximized.

$$\prod_{j=1}^{M} L(x_j, \theta) \tag{3.1}$$

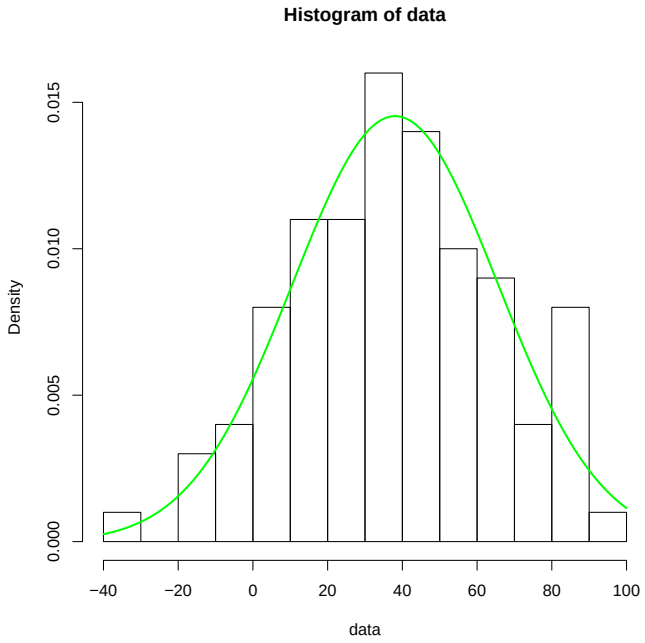

Figure 3.3: Histogram of randomly generated data from normal distribution $\mathcal{N}(40, 32^2)$. The green curve is a plot of normal distribution with the maximum likelihood estimate of θ parameters.

In case of Gaussian distribution, the likelihood function L is defined by

$$L(x, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.2}$$

where $x \in X$. Taking derivative of 3.1 with respect to μ equal to 0 yields

$$\sum_{j=1}^{M} x_j - M\mu = 0. \tag{3.3}$$

Maximum likelihood estimate of μ is therefore

$$\mu_{\text{est}} = \overline{X}_M. \tag{3.4}$$

Derivative of 3.1 with respect to σ^2 equal to zero results in MLE of σ^2 to be

$$\sigma^2_{\text{est}} = \frac{1}{M} \sum_{j=1}^{M} (x_j - \overline{X}_M)^2. \tag{3.5}$$

Figure 3.3 shows an example of MLE on normally distributed random variable.

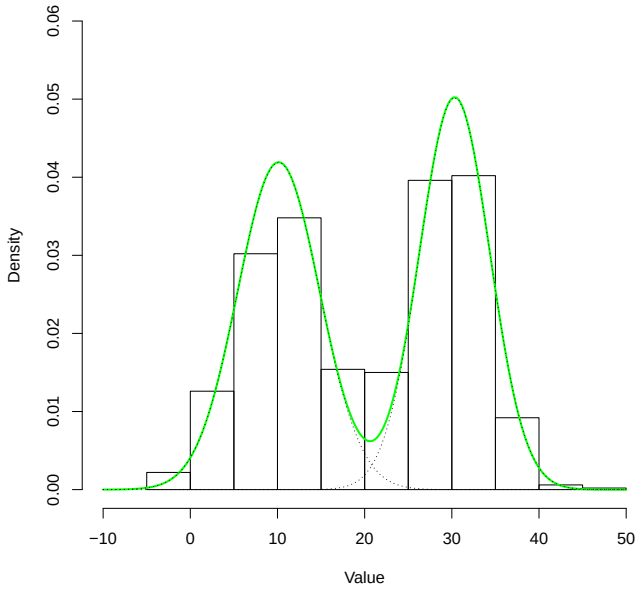

Figure 3.4: Histogram of randomly generated data from two normal distributions $\mathcal{N}(10, 5^2)$ and $\mathcal{N}(30, 4^2)$. The green curve is the best estimate, found using EM algorithm, of underlying normal mixture distribution. Dotted curves are estimates of individual Gaussian mixture components.

3.4 Gaussian mixture distribution

Given K components, there are K component parameters $\theta_1, \theta_2, \ldots, \theta_K$ and K weights $w_1, w_2, \ldots, w_K$ to estimate. Here I provide description of Expectation-Maximization algorithm based on lecture notes by Padhraic Smyth [6]. This algorithm provides an estimate of the aforementioned parameters.

Let's define a membership weight w_{jk}, of data point x_j in component k as

$$w_{jk} = \frac{\alpha_k \mathrm{PDF}_{\mathcal{N}(\theta_k)}(x_j)}{\sum_{i=1}^{K} \alpha_i \mathrm{PDF}_{\mathcal{N}(\theta_i)}(x_j)}, \tag{3.6}$$

where α_k is a probability that a randomly selected x_j was generated by component k. w_{jk} can be thought of as a level of certainty that x_j was generated by component k. $\sum_{k=1}^{K} w_{jk} = 1$ holds.

Likelihood of all data points given gaussian mixture parameters is defined as

$$l = \sum_{j=1}^{M} \sum_{k=1}^{K} \alpha_k \mathrm{PDF}_{\mathcal{N}(\theta_k)}(x_j). \tag{3.7}$$

EM is an iterative algorithm, typically iterating until convergence is detected. Each iteration consists of two parts. An E-step and an M-step.

In the E-step, weight w_{jk} is computed for each data point x_j and component k.

Define $W_k = \sum_{j=1}^{M} w_{jk}$, the weight of all data points in component k. In the M-step, new model parameters are calculated as follows:

$$\alpha_k^{\text{new}} = \frac{W_k}{M}, \tag{3.8}$$

$$\mu_k^{\text{new}} = \frac{\sum_{j=1}^{M} w_{jk} x_j}{W_k}, \tag{3.9}$$

$$\sigma_k^{2\text{new}} = \frac{\sum_{j=1}^{M} w_{jk} (x_j - \mu_k^{\text{new}})^2}{W_k}. \tag{3.10}$$

Where μ_k^{new} resembles standard equation for estimating mean, and $\sigma_k^{2\text{new}}$ is similar to an equation for estimating variance. The differences are in weighting.

Before the iterations begin, an initial guess of model parameters is necessary. This can be chosen randomly or through a heuristic. One issue with the EM algorithm is that it does not guarantee to find a global optimum.

Termination of the algorithm is determined by checking that the likelihood 3.7 of all data points hasn't improved enough in between iterations. In another words

$$l_{\text{cur}} - l_{\text{prev}} < \text{tol}. \tag{3.11}$$

Chapter 4

Analysis

4.1 Markov chain

4.1.1 Introduction

Markov chain describes possible sequences of events in which a probability of being at a state at given time depends only on value of previous state.

4.1.2 Description

A Markov chain is defined by

- set of states $S = \{S_1, S_2, \ldots, S_N\}$,
- transition matrix $A \in \mathbf{R}^{N \times N}$,
- initial probability vector $\pi \in \mathbf{R}^N$.

Where $a_{ij}, 1 \leq i, j \leq N$ is a probability of transitioning from state S_i into state S_j. π_i is a probability that the initial state is S_i.

Suppose a sequence of states $y_1, y_2, \ldots, y_T$, where $T \geq 2$. At time t, where $1 \leq t \leq T-1$, the transition probability a_{ij} is defined as $P(y_{t+1} = S_j | y_t = S_i)$.

4.1.3 Markov property

Markov chain satisfies a Markov property. This property says that $P(y_{t+1} = S_i | y_1 = s_1, y_2 = s_2, \ldots, y_t = s_t) = P(y_{t+1} = S_i | y_t = s_t)$. In another words, being at state S_i at time $t+1$ only conditionally depends on being at state s_t at time t. Further history doesn't influence the probability.

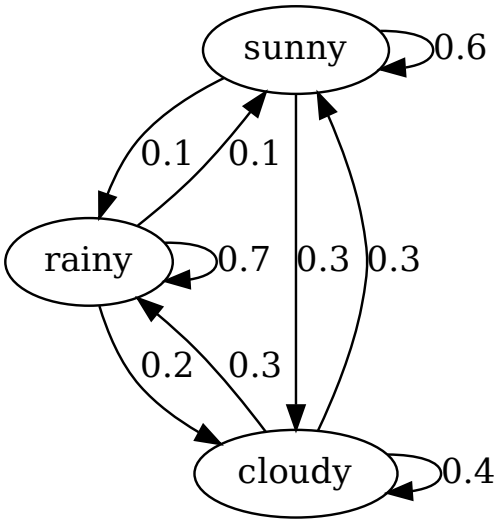

Figure 4.1: Visualization of a Markov chain with three states.

4.1.4 Example

As an example, consider a Markov chain (MC) with three states: $S_1 =$ sunny, $S_2 =$ rainy and $S_3 =$ cloudy. This MC models how weather changes on every day. Suppose the following transition matrix:

$$A = \begin{bmatrix} 0.6 & 0.1 & 0.3 \\ 0.1 & 0.7 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}. \tag{4.1}$$

If the weather is sunny, tomorrow will be most likely sunny as well. With probability 0.3 it will be cloudy. Rainy weather is very unlikely tomorrow. Consider the following initial probability vector:

$$\pi = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.5 \end{bmatrix}. \tag{4.2}$$

The most likely weather on first day of measurement is cloudy.

4.1.5 Limitations

The previous example only takes into account weather at time t to predict weather at time $t + 1$. In reality, there are multiple factors that influence what the weather will be like. According to the example, if a weather is sunny now, it is unlikely it will start raining. If we, however, take into account humidity, we can improve the prediction. If we know that humidity is high, even though it was sunny, we can predict it started raining. Hidden Markov Model addresses this issue.

4.2 Hidden Markov Model

4.2.1 Introduction

Hidden Markov Model (HMM) is an extension of Markov chain. The states are now called hidden states and every hidden state has an associated observed state.

4.2.2 Description

HMM is defined by: (based on paper by L. Rabiner [7])

- set of hidden states $S = \{S_1, S_2, \ldots, S_N\}$,
- set of observed symbols X,
- transition matrix $A \in \mathbf{R}^{N \times N}$,
- observation probability functions $B = \{b_1(o), b_2(o), \ldots, b_N(o)\}$,
- initial probability vector $\pi \in \mathbf{R}^N$.

The values of the X can be either discrete or continuous. The values of S are assumed to be discrete. The observation probability function $b_i(o)$ describes a probability $P(x_t = o|y_t = S_i)$. This probability function is also called an emission function.

Imagine a walk through a HMM graph (see figure 4.3 for an example graph). At time point t, there is a hidden state y_t and an observed symbol x_t. Value of the observed symbol conditionally depends on value of the hidden state. The dependency is described by emission function $b_i(o)$.

If $t \neq 1$ and $T > 1$, there is a hidden state y_t that conditionally depends on value of previous hidden state y_{t-1}. The dependency is described by transition a_{ij}. If $t = 1$, there is a hidden state y_1 that conditionally depends on initial transition π_i.

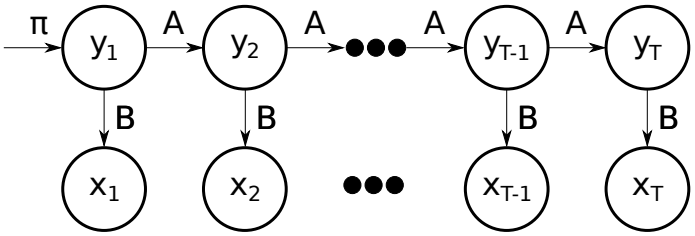

Figure 4.2: Visualization of a walk through a Hidden Markov Model. At each time point, there is a hidden state y_t and an observed symbol x_t.

4.2.3 Operations

There are three common operations that can be performed on HMM. Learning, inference and scoring.

4.2.3.1 Learning

Learning is used to estimate the model parameters $\{\pi, A, B\} = \lambda$. It is desired to estimate the parameters such that $\prod_{j=1}^{M} P(\mathbf{x}_j|\lambda)$ is maximized. Where $M \in \mathbf{N}$ is a number of sequences to be learned. $\mathbf{x}_j$ is a j-th sequence of observed symbols. In another words, it is desired to maximize the probability that given sequences of observed symbols were generated by the model.

4.2.3.2 Inference

Inference returns the most likely sequence of hidden states, given a sequence of observed symbols.

4.2.3.3 Scoring

Given a sequence of observed symbols $x_1, x_2, \ldots, x_T$, what is the probability that the sequence was generated by HMM with parameters λ? Scoring answers this question.

4.2.4 Learning

4.2.4.1 MLE/EM approach

Initial transition function The initial probability vector π can be viewed as a probability function π_i, assigning a probability to every hidden state $S_i \in S$. This is a function of discrete random variable that defines discrete probability distribution. As such, MLE technique (discussed in chapter 3) can be used to estimate the distribution parameters.

Transition function Every row of A defines a probability function a_{ij} with parameter j. These functions are called transition functions. The function describes a probability of transitioning into state S_j form state S_i. The transition functions define discrete probability distributions. Parameters of every such distribution can be estimated using MLE.

Emission function The observation probability functions $b_i(o)$ are functions of either discrete or continuous random variable that takes on values in set of observed symbols X. For purposes of this thesis, it is assumed that continuous variables have Gaussian or Gaussian mixture distribution.

The functions $b_i(o)$ are also called emission functions (emissions). If values of the set of observed symbols X are discrete, then the emissions define discrete distributions and the parameters can be estimated through MLE. If the values of X are continuous, then the emissions define continuous probability distributions. As discussed earlier, parameters of Gaussian distribution can be estimated using MLE. Parameters of Gaussian mixture distribution can be estimated through EM algorithm.

4.2.5 Inference

Given a sequence of observed symbols, what is the most likely associated sequence of hidden states? Viterbi algorithm answers this question. Let's define $\alpha_t(i)$ to be the maximum probability of sequence of hidden states and observed symbols up to time point t:

$$\alpha_t(i) = max_{y_1,y_2,\ldots,y_{t-1}} P(y_1, y_2, \ldots, y_{t-1}, y_t = S_i, x_1, x_2, \ldots, x_t|\lambda), \tag{4.3}$$

where λ are parameters of HMM model.

Initialization At time point $t = 1$, the probability of being at hidden state S_i is simply the initial probability of being at that state and a probability of being at observed state x_1 given hidden state S_i. In another words:

$$\alpha_1(i) = \pi_i b_i(x_1). \tag{4.4}$$

Recursion Taking into account the structure of HMM, the equation 4.3 can be rewritten recursively as

$$\alpha_t(j) = [\max_i \alpha_{t-1}(i)a_{ij}]b_j(x_t). \tag{4.5}$$

Hidden state at particular time point t is determined by

$$s_t = S_{\text{argmax}_i\, \alpha_t(i)}. \tag{4.6}$$

Complexity At each time point t and hidden state index j, the equation 4.5 loops through every hidden state index i. There are N hidden states. In addition, the recursion continues for every time point t. There are T time points. Thus the overall complexity is N^2T.

4.2.6 Scoring

In previous section, I have defined $\alpha_t(i)$ to be a probability of a sequence of hidden states up to time point t, where $y_t = S_i$, and a sequence of observed symbols up to time point t, given model parameters.

In order to compute the probability that sequence of observed symbols was generated by the HMM (score), take a look how $\alpha_T(i)$ is defined:

$$\alpha_T(i) = max_{y_1,y_2,\ldots,y_{T-1}} P(y_1, y_2, \ldots, y_{T-1}, y_T = S_i, x_1, x_2, \ldots, x_T|\lambda). \quad (4.7)$$

The score is then

$$\sum_{i=1}^{N} \alpha_T(i). \quad (4.8)$$

4.2.7 Example

Suppose an extension of an example mentioned in Markov chain chapter. i.e. there are three hidden states of weather: S_1 = sunny, S_2 = rainy and S_3 = cloudy. Lets introduce a continuous observed random variable describing humidity level. It is easy to imagine that on sunny day, the humidity will be typically lower than on a rainy day. This observable variable can help in predicting future weather.

4.2.8 Limitations

There is only one observed variable, which may not be sufficient for successful prediction in real-life applications. One could increase the dimensionality of the obsereved variable. That would, however, lead to a problem called curse of dimensionality. As we would increase the dimensionality of observed variable, the learning time would increase greatly up to the point where it would no longer be tractable. Dynamic naive Bayesian classifier addresses this issue.

4.3 Bayesian network

Bayesian network is a probabilistic graphical model consisting of nodes and edges. The model is a directed acyclic graph (DAG). Every node describes a random variable which can be either discrete or continuous. Edges describe a conditional dependency among the nodes (variables).

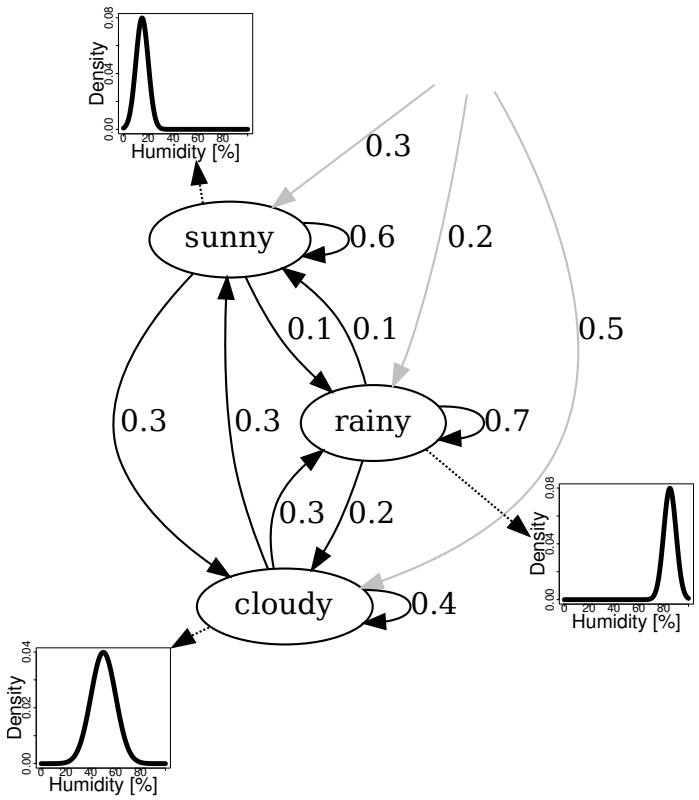

Figure 4.3: Visualization of a Hidden Markov Model with three states and a continuous observed variable.

As an example, consider a Bayesian network in figure 4.4. Every node represents a discrete random variable with two possible states - *true* and $false$. It can be imagined that dehydration can cause red skin, but it can also influence presence of cancer. In addition, red skin may signal a lack of water intake, or the skin may be red because of cancer cells. Every node defines a conditional probability table (CPT) given nodes it depends on.

4.4 Naive Bayesian classifier

Naive Bayesian classifier is a classifier (model that assigns class labels to given inputs) where features (random variables) are assumed to be conditionally independent from each other.

Imagine we would like to classify a class C_k given feature vector $\mathbf{x}$:

$$P(C_k|x_1, x_2, \ldots, x_n). \tag{4.9}$$

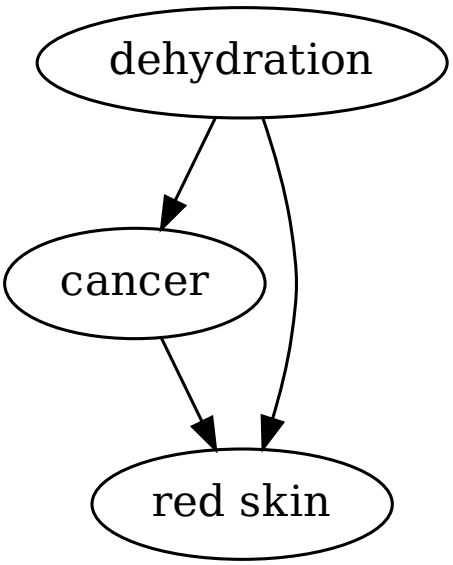

Figure 4.4: Visualization of a Bayesian network with three states.

dehydration	
true	false
0.1	0.9

Table 4.1: CPT for *dehydration*.

input	cancer	
dehydration	true	false
true	0.1	0.9
false	0.01	0.99

Table 4.2: CPT for *cancer*.

input		red skin	
cancer	dehydration	true	false
true	true	0.4	0.6
true	false	0.3	0.7
false	true	0.4	0.6
false	false	0.1	0.9

Table 4.3: CPT for *red skin*.

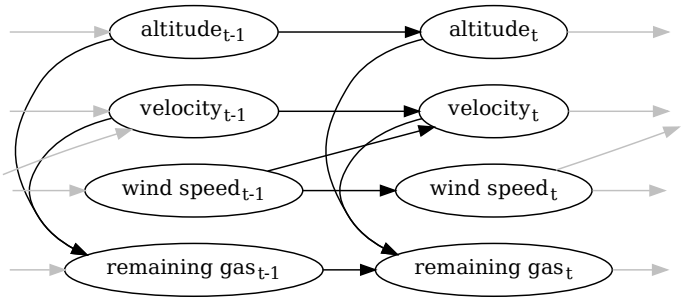

Figure 4.5: Visualization of a Dynamic Bayesian network with four variables at time t.

Using Bayes' theorem, the conditional probability can be rewritten as

$$P(C_k|\mathbf{x}) = \frac{P(C_k, \mathbf{x})}{P(\mathbf{x})} = \frac{P(C_k)P(\mathbf{x}|C_k)}{P(\mathbf{x})}. \tag{4.10}$$

Because the features are assumed to be conditionally independent from each other, the probability in numerator can be rewritten as

$$P(\mathbf{x}|C_k) = P(x_1|C_k)P(x_2|C_k)\cdots P(x_n|C_k). \tag{4.11}$$

Because of the independence assumption, learning of the classifier can be performed effectively, as mentioned in artice by K. M. Leung [8].

4.5 Dynamic Bayesian network

Dynamic Bayesian network is a Bayesian network which accounts for time. At each time point t, the values of random variables can be computed based on values of associated random variables in time $t-1$.

Consider figure 4.5 as an example. It can be imagined that velocity of an airplane at time t conditionally depends on velocity at time $t-1$ as well as on wind speed at time $t-1$. Remaining gas at time t conditionally depends on remaining gas at $t-1$ and it is also influenced by altitude and velocity at current time t.

4.6 Dynamic naive Bayesian classifier

4.6.1 Introduction

Dynamic naive Bayesian classifier (DNBC) is an extension of Hidden Markov Model (HMM). The difference is that there may be a number of observed

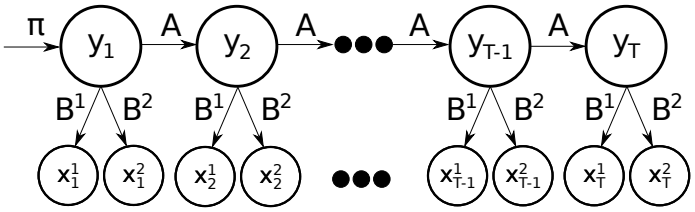

Figure 4.6: Visualization of a Dynamic naive Bayesian classifier with two observed variables. At each time point, there is a hidden state y_t and two observed states: x_t^1 and x_t^2.

variables. In contrast, HMM is defined for only one observed variable. DNBC is dynamic, because it classifies sequences with variables at every time t. It is also naive, because the output variables are assumed to be conditionally independent from each other. Figure 4.6 demonstrates a walk through a model with 2 observed variables.

4.6.2 Description

DNBC is defined by

- number of observed variables M,
- set of hidden states $S = \{S_1, S_2, \ldots, S_N\}$,
- sets of observed symbols $X^j, j \in [1, M]$
- transition matrix $A \in \mathbf{R}^{N \times N}$,
- observation probability functions $B^j = \{b_1^j(o), b_2^j(o), \ldots, b_N^j(o)\}, j \in [1, M]$,
- initial probability vector $\pi \in \mathbf{R}^N$.

The values of particular set of observed symbols X^j can be either discrete or continuous. The values of S are assumed to be discrete. The observation probability function $b_i^j(o)$ describes a probability $P(x_t^j = o | y_t = S_i)$. $b_i^j(o)$ is called an emission function.

Imagine a walk through DNBC graph. At time point t, there is a hidden state y_t and M observed symbols x_t^j where $j \in [1, M]$. Value of every observed symbol conditionally depends on value of the hidden state. The dependency is described by emission function $b_i^j(o)$.

An important property of DNBC is that the observed variables are assumed to be independent from each other. Therefore, DNBC does not define any conditional dependency function between two observed variables.

If $t \neq 1$ and $T > 1$, there is a hidden state y_t that conditionally depends on value of previous hidden state y_{t-1}. The dependency is described by transition

a_{ij}. If $t = 1$, there is a hidden state y_1 that conditionally depends on initial transition π_i.

4.6.3 Operations

There are three common operations that can be performed on DNBC. Learning, inference and scoring.

4.6.3.1 Learning

Learning is used to estimate the model parameters $\{\pi, A, B^1, B^2, \ldots, B^M\} = \lambda$. It is desired to estimate the parameters such that $\prod_{j=1}^{M} P(\mathbf{x}_j|\lambda)$ is maximized. Where $M \in \mathbf{N}$ is a number of sequences to be learned. $\mathbf{x}_j$ is a j-th sequence of observed symbols. In another words, it is desired to maximize the product of probabilities that given sequence of observed symbols was generated by given model.

4.6.3.2 Inference

Inference returns the most likely sequence of hidden states, given sequences of observed symbols. Each sequence of observed symbols corresponds to particular observed variable. Each sequence thus has the same number of elements.

4.6.3.3 Scoring

Given sequences of observed symbols $x_1^j, x_2^j, \ldots, x_T^j, j \in [1, M]$ (one sequence per observed variable), what is the probability that the sequences were generated by DNBC with parameters λ? Scoring answers this question.

4.6.4 Learning

4.6.4.1 Maximum likelihood estimation

This learning approach is similar to one described in case of HMM. The only difference is that there are multiple emission functions $b_i^j(o)$ where $j \in [1, M]$ and M is the number of observed variables. Therefore, there is now $M \times N$ continuous distributions to be learned. Each distribution is assumed to be Gaussian or Gaussian mixture.

4.6.5 Inference

Inference algorithm is again Viterbi. It is simply extended to support multiple observed variables. Let $\mathbf{x}_t^j$ be a sequence of observed symbols of variable j up to time point t.

$$\mathbf{x}_t^j = x_1^j, x_2^j, \ldots, x_t^j. \tag{4.12}$$

Let's define $a_t(i)$ to be the maximum probability of sequence of hidden and sets of observed symbols up to time point t:

$$a_t(i) = max_{y_1,y_2,\ldots,y_{t-1}} P(y_1, y_2, \ldots, y_{t-1}, y_t = S_i, \mathbf{x}_t^1, \mathbf{x}_t^2, \ldots, \mathbf{x}_t^M | \lambda), \quad (4.13)$$

where λ are parameters of DNBC model.

Initialization At time point $t = 1$, the probability of being at hidden state S_i is equal to the initial probability of being at that state and probabilities of being at observed symbol x_1^j given hidden state S_i. In another words:

$$\alpha_1(i) = \pi_i \prod_{j=1}^{M} b_i^j(x_1^j). \quad (4.14)$$

Recursion Taking into account the structure of DNBC, the equation 4.13 can be rewritten recursively as

$$\alpha_t(j) = [\max_i \alpha_{t-1}(i) a_{ij}] \prod_{k=1}^{M} b_j^k(x_t^k). \quad (4.15)$$

Hidden state at particular time point t is determined by

$$s_t = S_{\text{argmax}_i \, \alpha_t(i)}. \quad (4.16)$$

Complexity At each time point t and hidden state index j, the equation 4.15 loops through every hidden state index i and every observed variable index k. There are N hidden states and M observed variables. In addition, the recursion continues for every time point t. There are T time points. Thus the overall time complexity is $N(N + M)T$.

4.6.6 Scoring

Scoring is performed the same way as in case of HMM. That is, the score can be calculated as

$$\sum_{i=1}^{N} \alpha_T(i). \quad (4.17)$$

4.6.7 Example

Consider a problem of weather prediction. Let's define a set of hidden states to be $S = \{S_1 = \text{sunny}, S_2 = \text{rainy}, S_3 = \text{cloudy}\}$. Let's consider the following observed variables: temperature, humidity and windiness. At every time t, the

hidden state conditionally depends on hidden state at time $t - 1$. The three hidden variables at time t conditionally depend on hidden state t. These variables can help in predicting the hidden state.

4.6.8 Limitations

The observed variables are assumed to be conditionally independent from each other. While this assumption enables the use of algorithms with relatively low time complexity, it also limits the ability to describe more complicated relations among random variables. For example, the observed variables may be correlated in reality.

Chapter 5

Implementation

I have implemented a Dynamic naive Bayesian Classifier in Scala programming language that runs on top of Apache Spark. I named this software *dnbc-scala*.

5.1 Scala language

Scala is a programming language that combines functional and object-oriented programming. Scala code compiles into Java Virtual Machine (JVM) bytecode. This enables one to incorporate existing Java libraries into a Scala project.

5.2 Apache Spark

Apache Spark is an engine for cluster-computing. It provides Application Programming Interface (API) that developers can use to execute computations on multiple nodes - processors or machines in a cluster. Spark is also fault-tolerant, meaning that if a computation fails at given node, another node will take over.

5.3 Project structure

The project is divided into 5 modules:

- *core* - main module containing DNBC interface, helper class for loading data set into required format and a class for measuring performance of DNBC on a data set,
- *core-test* - contains tests of *core* module,
- *performance* - provides command line interface for generating a parametric data set and measuring DNBC performance on it,

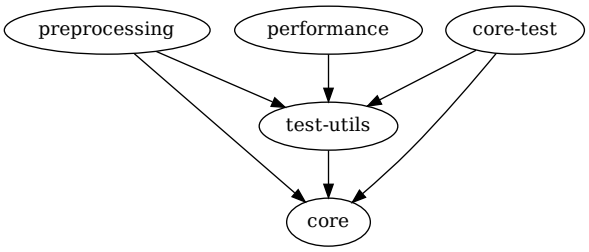

Figure 5.1: Dependencies among modules in *dnbc-scala* project.

- *preprocessing* - transforms a predefined data set into data sets with different parameters,
- *test-utils* - provides commonly used functions in tests or in performance measurement.

Figure 5.1 provides overview of the project structure and module dependencies.

5.4 Usage

DynamicNaiveBayesianClassifier object *DynamicNaiveBayesianClassifier* object in *core* module serves as a factory for already learned class *DynamicNaiveBayesianClassifier*.

```
object DynamicNaiveBayesianClassifier {
  def mle(sc: SparkContext,
          sequences: Iterable[Seq[State]],
          continuousVariableHints: Option[List[Int]] =
              Option.empty)
          : DynamicNaiveBayesianClassifier = {
          // ...
          }
}
```

The function *mle* returns a learned (using MLE) model. *sequences* are sequences of *States* used for learning. Optional argument *continuousVariableHints* can be provided to set the number of components in gaussian mixtures. The implementation does not try to estimate the number. If the parameter is not provided, the continuous variables are expected to have normal distribution.

The model parameters (*Edges*) are learning in every sequence of states. After all the sequences have been processed, learning is finalized by calling *learnFinalize* on every edge. These functions are executed in parallel since they are not data-dependent with each other.

DynamicNaiveBayesianClassifier class This class represents a DNBC model with known parameters.

```
class DynamicNaiveBayesianClassifier(
initialEdge: LearnedDiscreteEdge,
transitions: Map[String, LearnedDiscreteEdge],
discreteEmissions: List[Map[String, LearnedDiscreteEdge
    ]],
continuousEmissions: List[Map[String,
    LearnedContinuousEdge]]) {
def inferMostLikelyHiddenStates(observedStates: Seq[
    ObservedState]): List[String]
def score(observedStates: Seq[ObservedState]): Double
}
```

The *inferMostLikelyHiddenStates* function returns the most likely sequence of hidden states given a sequence of observed states. I am using Viterbi algorithm. The following is a function for Viterbi initialization.

```
  private def viterbiInitialize(observedStates: Seq[
      ObservedState]): Map[String,Double] = {
    var vcur = Map.empty[String,Double]
    for (hiddenState <- transitions.keys) {
      var emissionsSum = 0.0
      emissionsSum += discreteEmissions.zipWithIndex.map
          (z => Math.log(z._1(hiddenState)
        .probability(observedStates.head.
            DiscreteVariables(z._2)))).sum
      emissionsSum += continuousEmissions.zipWithIndex.
          map(z => Math.log(z._1(hiddenState)
        .probability(observedStates.head.
            ContinuousVariables(z._2)))).sum
      vcur += (hiddenState -> (emissionsSum + Math.log(
          initialEdge.probability(hiddenState))))
    }
    vcur
  }
```

The *score* function returns a log-probability that given sequence of observed states was generated by the DNBC. It is internally using Viterbi algorithm to compute $\alpha_T(i)$.

The Viterbi algorithm is not implemented in parallel. The reason is that the algorithm proceeds by time points, and each time point depends on the previous one. At every time point, the parallelization is not worth it, since there is only a bunch of additions taking place.

State An observed state at given time point consists of observations of all the discrete and continuous variables. This is represented by *ObservedState* class.

```
class ObservedState(discreteVariables: List[String],
    continuousVariables: List[Double]) {
  def DiscreteVariables: List[String] =
      discreteVariables
  def ContinuousVariables: List[Double] =
      continuousVariables
}
```

A state at given time point consists of a hidden state and an *ObservedState*. Throughout the project, the hidden states are of *String* type.

```
class State(hiddenState: String, observedState:
    ObservedState) {
  def HiddenState: String = hiddenState
  def ObservedState: ObservedState = observedState
}
```

Edge An edge represents a particular random variable, parameters of which are to be learned.

```
trait Edge[T] {
  def learn(occurrence: T): Unit
  def learnFinalize(): LearnedEdge[T]
}
```

The *learn* function notifies an edge that *occurence* occured. User calls *learnFinalize* function to estimate the variable parameters based on previous occurences.

There are two kinds of edges: *DiscreteEdge* and *ContinuousEdge*. As expected, *DiscreteEdge* represents a discrete random variable, whereas *ContinuousEdge* represents a continuous random variable.

To estimate the parameters of continuous random variables using MLE, I am using jMEF Java library. Previously I used Spark's MLlib library for estimating parameters of a Gaussian Mixture Model (GMM). The problem with this is approach was that it can only estimate the parameters by making a computation across multiple nodes in a cluster. This is not a desired behavior since there are many estimations of small data that should run in

parallel instead. Therefore, I ended up using jMEF library which estimates the parameters in sequential fashion.

LearnedEdge A *LearnedEdge* represents a random variable which parameters have been already estimated. The *probability* function returns a probability that a state occurs.

```
trait LearnedEdge[T] {
  def probability(state: T): Double
}
```

RandomEdge A *RandomEdge* generates random value given random variable parameters. This is useful for performance measurement, where data sets are randomly generated.

```
trait RandomEdge[T] {
  def next(): T
}
```

5.5 Underflow

Since the Viterbi algorithm is computing products of probabilities, the resulting value gets small in few iterations. To avoid underflow, one can compute a log-probability instead, using the following trick:

$$\log(abc) = \log(a) + \log(b) + \log(c). \tag{5.1}$$

Now instead of multiplying tiny numbers together, one can use addition.

5.6 Data set generation

The *test-utils* module contains a function *generatePerformanceDataSet*. This generates a random data set, given parameters, that can be used for measuring learning and testing time. The table 5.1 contains a list of supported parameters.

The initial edge probabilities are generated from a normal distribution and then normalized. The transition destinations are chosen randomly from a uniform distribution. The probabilities of transitions are then set the same way as in case of initial edge. Same approach is used for discrete emissions.

For continuous emissions, the components of GMM are chosen randomly in uniform fashion in range of these parameters:

$$\mu \in [-10, 10], \sigma^2 \in [1, 4]. \tag{5.2}$$

The weights in a *RandomContinuousEdge* are all equal.

parameter
length of sequences
number of training sequences
number of testing sequences
number of hidden states
number of discrete observed variables
number of continuous observed variables
max number of components per mixture
transitions per hidden state

Table 5.1: Data set generation parameters

Chapter 6

Experiments

6.1 Unit tests

I am using a Toy Robot data set [2] as a base data set for unit tests. Imagine a robot wandering through a maze. There are 12 hidden states. Every hidden state is a 2D coordinate. Every hidden state has an associated color. The color is a discrete observed variable consisting of 4 states: red, green, blue and yellow. The initial state is chosen randomly. The data set contains of 200 training sequences and 200 test sequences. Every sequence consists of 200 time points. 10% of the data set observed states at time points are chosen at random. This depicts robot's occasional failure to detect correct color.

Measurement For every data set in unit tests, an average success rate (ASR) is measured. This average is the average percentage of hidden states inferred correctly in the testing phase.

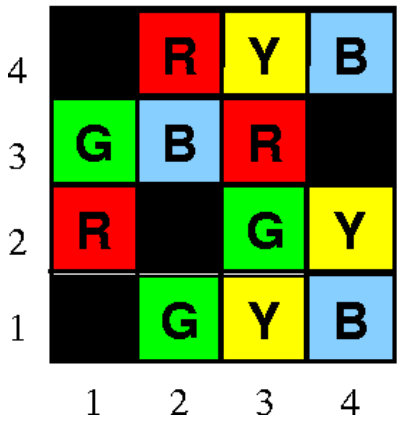

Figure 6.1: Visualization of a Toy Robot data set [2].

CPU	2x 8-core Intel Xeon E5-2650 v2 2.6 GHz
RAM	96 GB
disk	2x500 GB HDD WD Velociraptor 10k SATA

Table 6.1: Machine specification [3]

Discrete variable ASR on the Toy Robot data set is 65%.

Continuous variable The Toy Robot data set is transformed to have a single continuous observed variable instead. Every color (discrete random variable) is replaced by temperature. The idea is that the 4 colors have different temperature distributions if sun is shedding light on the maze. ASR of this data set is 42%.

Discrete and continuous variable A discrete observed variable is added to the continuous variable data set. This variable describes the quadrant of given 2D coordinate. It is assumed that these two variables are independent. ASR of this data set is 76%.

Gaussian mixture A data set with single continuous observed variable is generated. Hidden states can take on two values: true and false. If the hidden state is true, observed state is generated from a Gaussian distribution. If the hidden state is false, observed state is generated from a Gaussian mixture distribution. Point of this test is to check that providing a variable hint describing the number of mixture components improves the ASR. ASR with correctly set variable hint is 3% higher compared to a case where no hint is set.

Scoring A score is computed for a sequence of observed states that comes from a data set a DNBC was trained on. Another score is computed for a sequence of observed states that was generated randomly. Since the score is a log-probability, the values are negative. The following holds:

```
assert ( scoreReal*1.5 > scoreRandom ).
```

6.2 Performance

In this section, I focus on learning and testing time given a generated parametric data set. The measurements were made on a machine specified in table 6.1. Every job was assigned 15 GB RAM and 10 GB disk space.

length of sequences	200
number of training sequences	1000
number of testing sequences	200
number of hidden states	10
number of discrete observed variables	30
number of continuous observed variables	30
max number of components per mixture	3
transitions per hidden state	5

Table 6.2: Data set generation parameters for scalability measurement.

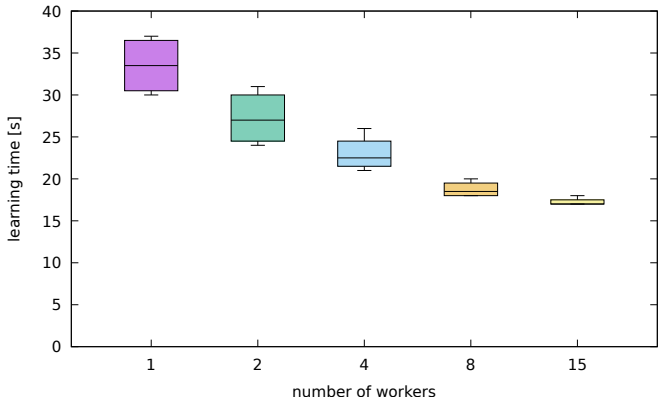

Figure 6.2: Learning time per number of workers.

Scalability The table 6.2 describes parameters used for data sets generation. Goal of this measurement is to see how well the software scales as we increase number of workers. Figure 6.2 shows results of this measurement. Every configuration was measured 4 times to enable better understanding of the data.

As it can be seen, the software does not scale particularly well. In fact, the majority of time is spent reading the data set data. This is not a parallel operation. Reading data is the bottleneck.

To focus on scalability of learning itself, figure 6.3 shows learning time of all continuous emissions, without the data loading. As it can be seen, this segment scales relatively well.

Training sequences Figure 6.4 shows approximately linear complexity as number of training sequences increases.

Hidden states Interestingly, as we increase the number of hidden states, the learning time does not change. Testing time increases super-linearly. This can be see in figure 6.5.

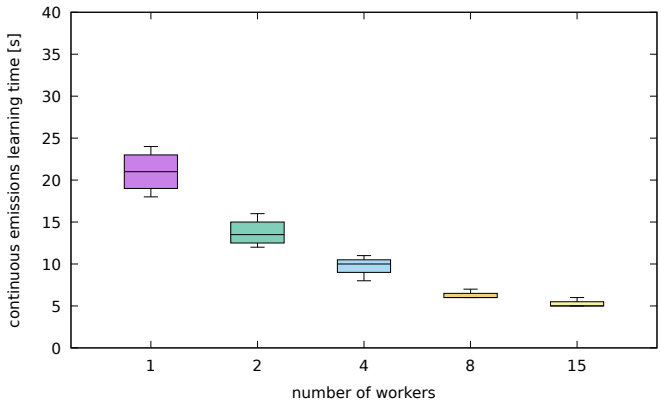

Figure 6.3: Continuous emissions learning time per number of workers.

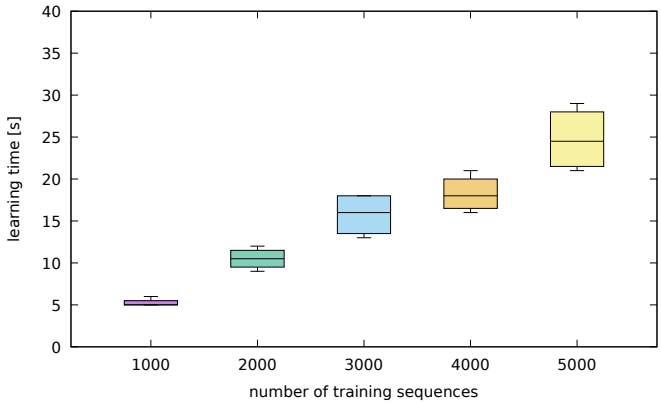

Figure 6.4: Learning time per number of training sequences.

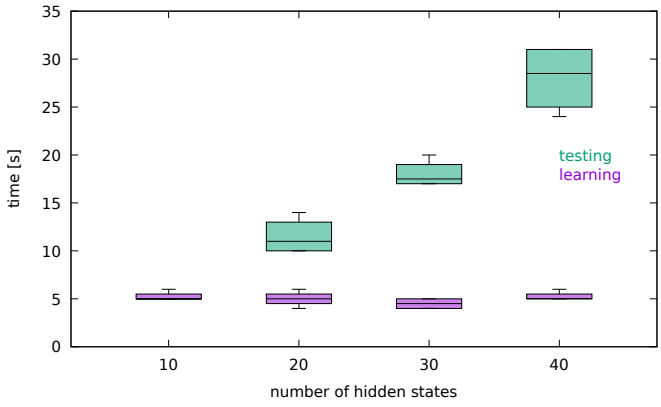

Figure 6.5: Learning and testing times per number of hidden states.

Conclusion

Bibliography

[1] Smason79: [online], 2012, [cit. 2018-04-01]. Dostupné z: https://commons.wikimedia.org/wiki/File:Gaussian-mixture-example.svg

[2] [online], 2006, [cit. 2018-04-12]. Dostupné z: https://www.cs.princeton.edu/courses/archive/fall06/cos402/hw/hw5/hw5.html

[3] [online], [cit. 2018-04-12]. Dostupné z: https://metavo.metacentrum.cz/pbsmon2/resource/luna.fzu.cz

[4] Patel, J. K.; Read, C. B.: *Handbook of the Normal Distribution*. Statistics: A Series of Textbooks and Monographs, Taylor & Francis, druhé vydání, 1996, ISBN 9780824793425.

[5] Grinstead, C. M.; Snell, J. L.: *Introduction to Probability*. American Mathematical Society, 2012, ISBN 9780821894149.

[6] Smyth, P.: Mixture Models and the EM Algorithm. *Department of Computer Science, University of California, Irvine*, 2017, [cit. 2018-04-01]. Dostupné z: http://www.ics.uci.edu/~smyth/courses/cs274/notes/mixture_models_EM.pdf

[7] Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. 1989.

[8] Leung, K. M.: Naive Bayesian Classifier. *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, listopad 2007, [cit. 2018-04-01]. Dostupné z: http://cis.poly.edu/~mleung/FRE7851/f07/naiveBayesianClassifier.pdf

Appendix A

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Appendix B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS