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

March 29, 2018

Declaration

In V Praze on March 29, 2018 . . . .

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

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

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

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

TODO: mention why normal distribution is so important?

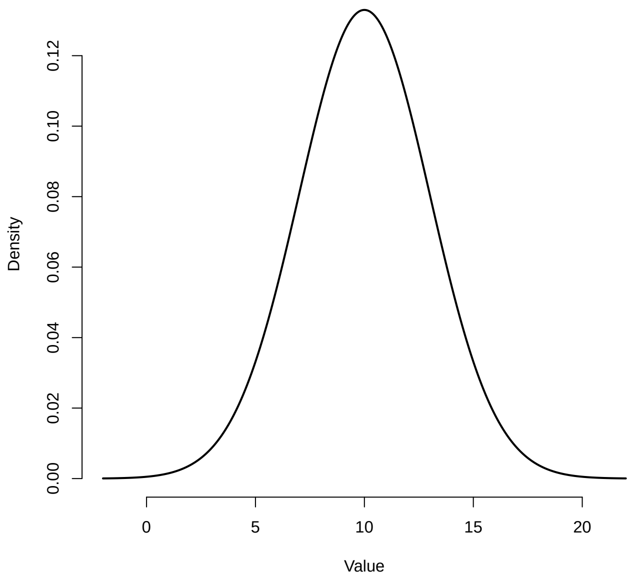

Figure 2.1: Visualization of Gaussian PDF with parameters $\mu = 10$ and $\sigma^2 = 3^2$.

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 mixtures.

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

Where w_i is a weight of particular normal distribution. The following conditions have to hold:

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

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

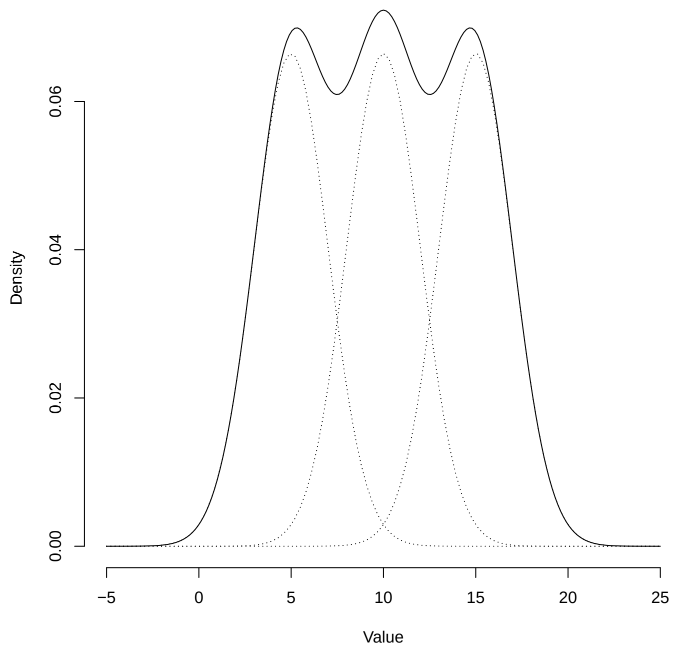

Figure 2.2: Visualization of a mixture of three Gaussian distributions with parameters $\mu_1 = 5, \mu_2 = 10, \mu_3 = 15$ and $\sigma^2_{1,2,3} = 2^2$. Author: Smason79, Wikimedia

Chapter 3

Analysis

3.1 Markov chain

3.1.1 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.

3.1.2 Description

A Markov chain is defined by

- set of states $S = \{S_1, S_2, \dots, S_N\}$,
- 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 $q_1, q_2, \dots, q_T$, where $T \geq 2$. At time t, where $1 \leq t \leq T-1$, there is a probability $P(q_{t+1} = S_j | q_t = S_i)$. This probability is described by a_{ij}.

3.1.3 Markov property

Markov chain satisfies a Markov property. This property says that $P(q_{t+1} = S_i | q_1 = s_1, q_2 = s_2, \dots, q_t = s_t) = P(q_{t+1} = S_i | q_t = s_t)$. 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.

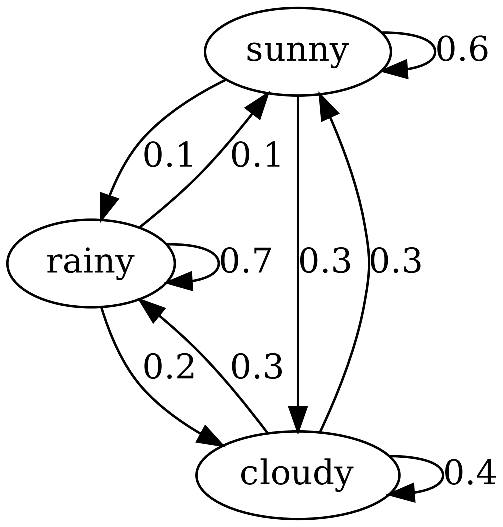

Figure 3.1: Visualization of a Markov chain with three states.

3.1.4 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:

$$A = \begin{bmatrix} 0.6 & 0.1 & 0.3 \\ 0.1 & 0.7 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix} \tag{3.1}$$

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:

$$\pi = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.5 \end{bmatrix} \tag{3.2}$$

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

3.1.5 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.

3.2 Hidden Markov Model

3.2.1 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.

3.2.2 Description

HMM is defined by

- set of hidden states $S = \{S_1, S_2, \ldots, S_N\}$,
- set of observed states 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. There are numerous different definitions of HMM, such as [TODO], [TODO] or [TODO]. The observation probability function $b_i(o)$ describes a probability $P(x_t = o|q_t = S_i)$. $b_i(o)$ is also called an emission function.

Imagine a walk through a HMM graph. At time point t, there is a hidden state q_t and an observed state x_t. Value of the observed state 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 q_i that conditionally depends on value of previous hidden state q_{t-1}. The dependency is described by transition a_{ij}.

If $t = 1$, there is a hidden state q_1 that conditionally depends on initial transition π_i.

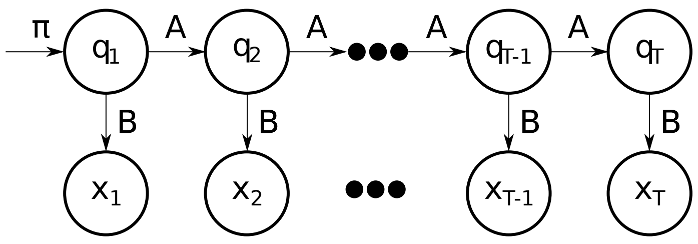

Figure 3.2: Visualization of a walk through a Hidden Markov Model. At each time point, there is a hidden state q_t and an observed state x_t.

3.2.3 Operations

There are two most important operations that can be performed on HMM. Learning and inference.

3.2.3.1 Learning

Learning is used to calculate 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 states. In another words, it is desired to maximize the product of probabilities that given sequence of observed states was generated by given model.

3.2.3.2 Inference

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

3.2.3.3 Scoring?

3.2.4 Learning

3.2.4.1 Maximum likelihood estimation

This method uses maximum likelihood estimation (MLE) to compute model parameters. 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_j is a j-th data point
- L is a likelihood function (defined further)

- θ describes model parameters
- Θ is a set of all model parameters

Let's examine how one can view parameters of HMM as probabilistic functions of random variables.

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.

Transition function Every row of A defines a probability function a_{ij} with parameter j. The function describes a probability of transitioning into state S_j form state S_i. a_{ij} is a function of discrete random variable.

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 states X. For purposes of this thesis, it is assumed that continuous variables have Gaussian or Gaussian mixture distribution.

Discrete random variables 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 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 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 3.4 shows a likelihood function L associated with the data.

Continuous random variables This work considers two kinds of distributions for continuous random variables: Gaussian distribution and Gaussian mixture distribution.

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 3.3 is maximized.

$$\prod_{j=1}^{M} L(x_j, \theta) \tag{3.3}$$

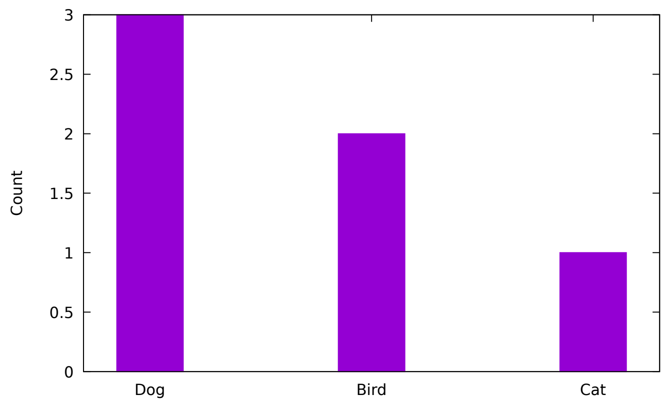

Figure 3.3: Frequency of data in discrete data set.

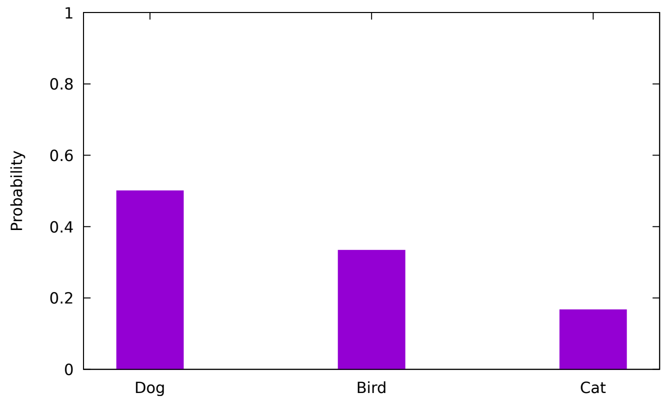

Figure 3.4: Likelihood function L of data in discrete data set. Probability is zero at undefined states.

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

$$L(x, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

Taking derivative of 3.3 with respect to μ equal to 0 yields

$$\sum_{j=1}^{M} x_j - M\mu = 0 \tag{3.5}$$

Maximum likelihood estimate of μ is therefore

$$\mu_{\text{est}} = \overline{X}_M \tag{3.6}$$

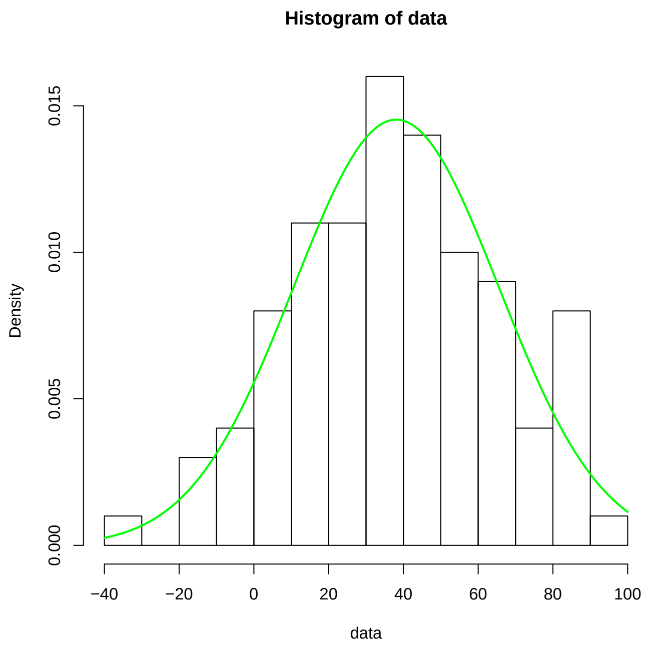

Figure 3.5: Histogram of randomly generated data from normal distribution $\mathcal{N}(40, 32^2)$. The green curve is a plot of normal distribution with the maximum likelihood estimate of θ parameters.

Derivative of 3.3 with respect to σ^2 equal to zero results in MLE of σ^2 to be

$$\sigma^2_{\text{est}} = \frac{1}{M} \sum_{j=1}^{M} (X_j - \overline{X}_M)^2 \tag{3.7}$$

Figure 3.5 shows an example of MLE on normally distributed random variable.

Gaussian mixture distribution TODO: not a MLE!

Given K gaussians, there are K parameters $\theta_1, \theta_2, \ldots, \theta_K$ to estimate. Here I provide description of Expectation-Maximization algorithm based on TODO. This algorithm provides an estimate of the aforementioned parameters.

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

$$w_{jk} = \frac{\alpha_k \text{PDF}_{\mathcal{N}}(x_j, \theta_k)}{\sum_{i=1}^{K} \alpha_i \text{PDF}_{\mathcal{N}}(x_j, \theta_i)} \tag{3.8}$$

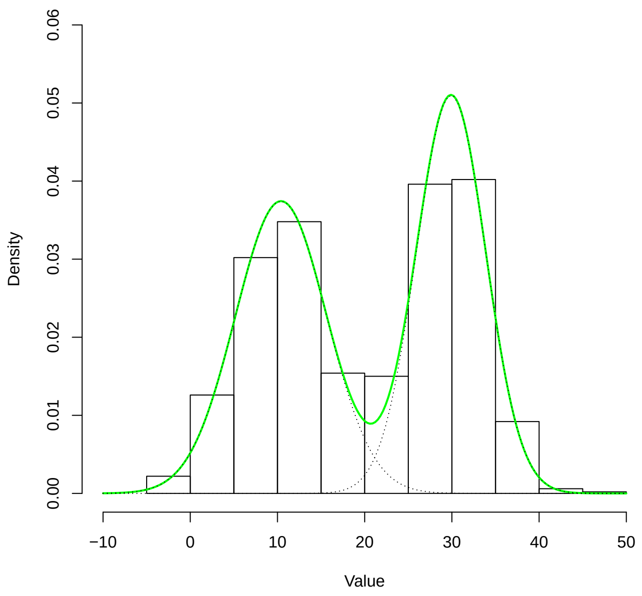

Figure 3.6: Histogram of randomly generated data from two normal distributions $\mathcal{N}(10, 5^2)$ and $\mathcal{N}(30, 4^2)$. The green curve is the best estimate, found using EM algorithm, of underlying normal mixture distribution. Dotted curves are estimates of particular normal distributions that are part of the Gaussian mixture.

Where α_k is a probability that a randomly selected x_j was generated by mixture k.

w_{jk} can be thought of as a level of certainty that x_j was generated by mixture k. $\sum_{k=1}^{K} w_{jk} = 1$ holds.

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

$$l = \sum_{j=1}^{M} \sum_{k=1}^{K} \alpha_k \text{PDF}_{\mathcal{N}}(x_j, \theta_k) \tag{3.9}$$

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 mixture k.

Define $W_k = \sum_{j=1}^{M} w_{jk}$, the weight of all data points in mixture k. In the

M-step, new model parameters are calculated as follows

$$\alpha_k^{\text{new}} = \frac{W_k}{M} \tag{3.10}$$

$$\mu_k^{\text{new}} = \frac{\sum_{j=1}^{M} w_{jk} x_j}{W_k} \tag{3.11}$$

$$\sigma_k^{2\,\text{new}} = \frac{\sum_{j=1}^{M} w_{jk}(x_j - \mu_k^{\text{new}})^2}{W_k} \tag{3.12}$$

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 3.9 of all data points hasn't improved enough in between iterations. In another words

$$l_{\text{cur}} - l_{\text{prev}} < \text{tol} \tag{3.13}$$

3.2.5 Inference

Given a sequence of observed states, 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 and observed states up to time point t:

$$\alpha_t(i) = max_{q_1,q_2,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_{t-1}, q_t = S_i, x_1, x_2, \ldots, x_t|\theta), \tag{3.14}$$

where θ 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:

$$\alpha_1(i) = \pi_i b_i(x_1) \tag{3.15}$$

Recursion Taking into account the structure of HMM, the equation 3.14 can be rewritten recursively as

$$\alpha_t(j) = [\max_i \alpha_{t-1}(i)a_{ij}]b_j(x_t) \tag{3.16}$$

Hidden state at particular time point t is determined by

$$s_t = S_{\mathrm{argmax}_i \, \alpha_t(i)} \tag{3.17}$$

Complexity At each time point t and hidden state index j, the equation 3.16 loops through every hidden state index i. There are $|S|$ hidden states. In addition, the recursion continues for every time point t. There are T time points. Thus the overall complexity is $|S|^2T$.

3.2.6 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.

3.2.7 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.

3.3 Dynamic naive Bayesian classifier

3.3.1 Description

Dynamic naive Bayesian classifier (DNBC) is an extension of Hidden Markov Model (HMM). The difference is that there may be a number of observed variables. In contrast, HMM is defined for only one observed variable. Figure 3.7 demonstrates model structure with 2 observed variables. DNBC is defined by

1. Number of observed variables N
2. Initial transition function $I : S \mapsto \mathbf{R}$

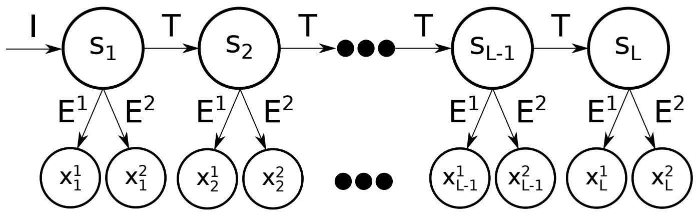

Figure 3.7: Visualization of a Dynamic naive Bayesian classifier with two observed variables. At each time point, there is a hidden state s_i and two observed states: x_i^1 and x_i^2.

3. Transition function $T : S \times S \mapsto \mathbf{R}$
4. Emission functions $E^j : S \times X^j \mapsto \mathbf{R}, j \in [1, N]$
5. Set of hidden states S
6. Sets of observed states $X^j, j \in [1, N]$

Each set of observed variables contains a set of observed states. This set can be either discrete or continuous. The set of hidden states is assumed to be discrete.

Given the following:

1. A number $L \in [1, \inf)$
2. A sequence of hidden states of length L
3. Sequences of observed states of length L
4. An index $i \in [1, L]$: one can think of this as a discrete point in time

At time point i, there is a hidden state s_i and N observed states x_i^j where $j \in [1, N]$. Value of every observed state depends on value of the hidden state. The dependency is described by emission function E^j. An important property of DNBC is that the observed variables are assumed to be independent. Therefore, DNBC does not define any dependency function between two observed variables.

If $i \neq 1$ and $L > 1$, there is a hidden state s_i that depends on value of previous hidden state s_{i-1}. The dependency is described by transition function T.

If $i = 1$, there is a hidden state s_1 that depends on initial transition function I.

3.3.2 Operations

There are two most important operations that can be performed on DNBC. Learning and inference.

3.3.2.1 Learning

Learning is used to calculate the model parameters $\{I, T, E^1, E^2, \dots, E^N\} = \lambda$. 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 states. In another words, it is desired to maximize the product of probabilities that given sequence of observed states was generated by given model.

3.3.2.2 Inference

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

3.3.3 Learning

3.3.3.1 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 E^j where $j \in [1, N]$. Every emission function is defined as $E^j : S \times X^j \mapsto \mathbf{R}$. For every j and hidden state $s \in S$, there is a probability function $X^j \mapsto \mathbf{R}$. This function can be represented by continuous probabilistic model. Therefore, there is now $N \times |S|$ continuous distributions to be learned. Each distribution is assumed to be Gaussian or Gaussian mixture.

3.3.4 Inference

Inference algorithm is again Viterbi. It is simply extended to support multiple observed variables. Let x_t^j be a sequence of observed states of variable j up to time point t.

$$x_t^j = x_1^j, x_2^j, \dots, x_t^j \tag{3.18}$$

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

$$a_t(i) = max_{s_1, s_2, \dots, s_{t-1}} P(s_1, s_2, \dots, s_{t-1}, s_t = S_i, x_t^1, x_t^2, \dots, x_t^N|\theta) \tag{3.19}$$

Where S_i is a function $\mathbf{N} \mapsto S$, assigning a unique index to each hidden state in S. θ 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 state x_1^j given hidden state S_i. In another words:

$$a_1(i) = I(S_i) \prod_{j=1}^{N} E^j(S_i, x_1^j) \tag{3.20}$$

Recursion Taking into account the structure of DNBC, the equation 3.19 can be rewritten recursively as

$$a_t(j) = [\max_i a_{t-1}(i) T(S_i, S_j)] \prod_{k=1}^{N} E^k(S_j, x_t^k) \tag{3.21}$$

Hidden state at particular time point t is determined by

$$s_t = S_{\mathrm{argmax}_i\, a_t(i)} \tag{3.22}$$

Complexity At each time point t and hidden state index j, the equation 3.21 loops through every hidden state index i and every observed variable index k. There are $|S|$ hidden states and N observed variables. In addition, the recursion continues for every time point t. There are L time points. Thus the overall time complexity is $|S|(|S| + N)L$.

Chapter 4

Návrh

Chapter 5

Realizace

Conclusion

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