FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Heterogenní kernel
Student:	Samuel Fabo
Vedoucí:	Ing. Jan Motl
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Mnohé algoritmy ve strojovém učení, například Support Vector Machine, umí pracovat s kernely. Komplikací ale je, že běžně používané kernely vyžadují, aby data byla vždy jen jednoho datového typu, například double. Když data obsahují různé datové typy, například typu string a double, můžeme je převést na jednu reprezentaci. Problém ale je, že výsledná reprezentace bývá paměťově náročná, takže se data už nemusí celá vejít do paměti. Možným řešením celého problému je navrhnout kernel, který pracuje přímo s různými datovými typy.

Navrhněte a implementujte kernel, který má následující vlastnosti:
1) umí pracovat s kombinací numerických a nominálních atributů,
2) umí se vypořádat s nominálními atributy o vysoké kardinalitě,
3) umí se vypořádat s chybějícími hodnotami.

Proveďte:
1) rešerši,
2) popište navržený kernel,
3) experimentálně porovnejte navržený kernel s vybranými postupy z rešerše,
4) vyhodnoťte výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. února 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Heterogeneous Kernel

Samuel Fabo

Department of Applied Mathematics
Supervisor: Ing. Jan Motl

May 16, 2019

Acknowledgements

I want to thank Ing. Jan Motl, who was guiding me throughout the whole process of creating this thesis and from whom I learned a lot. Thanks also go to my parents and family, who are supporting me during my entire studies.

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.

In Prague on May 16, 2019 …………………

Czech Technical University in Prague
Faculty of Information Technology
© 2019 Samuel Fabo. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Fabo, Samuel. *Heterogeneous Kernel.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Metódy strojového učenia, ktoré používajú kernelové funkcie sú dobre preskúmané, avšak väčšina týchto kernelových funkcií vie pracovať len s numerickými vstupnými dátami. Aby tieto kernely vedeli spracovať kategorické dáta, sme schopní priviesť do numerickej formy hlavne pomocou metód one-hot-encoding (OHE) alebo target-encoding. Nevýhodou OHE je, že signifikantne zvyšuje dimenzionalitu dát, ak je počet hodnôt v kategorických príznakoch (kardinalita) vysoká.

Táto práca prináša riešenie pre zmiešané dáta, s potenciálne vyššou kardinalitou kategorických príznakov. Je tu predstavený nový kernel, ktorý vie pracovať so zmiešanými dátami a má veľmi dobré výsledky merania času a pamäte na dátach s vysokou kardinalitou. Predstavujem tu hlavne tzv. *kategorický vektorový súčin*, ktorý imituje klasický vektorový súčin po OHE ako aj *kategorické Euclidovské vzdialenosti* imitujúce klasický prístup po OHE. Tento *heterogénny kernel* vie pracovať ako lineárny, polynomiálny, a RBF kernel.

Výsledky meraní ukázali, že tento kernel vie urýchliť výpočet a zmenšiť prírastok pamäte, ak by dataset obsahoval ako numerické, tak kategorické príznaky o vyššej kardinalite. Tento fakt bol taktiež demonštrovaný na reálnych datasetoch.

Kľúčové slová Nominálne dáta, kategorické dáta, heterogénne dáta, kernel metódy, klasifikácia, podporné vektory, SVM, kernelová hrebeňová regresia, vysoká kardinalita, chýbajúce hodnoty, chýbajúce dáta, predspracovanie dát

Abstract

Machine learning methods using kernel functions are well explored, but most of the kernel functions work only with numerical input. To let these numerical kernels work with categorical features, we need to use preprocessing methods such as one-hot-encoding (OHE) or target-encoding. The disadvantage of OHE is that it significantly increases the dimensionality of the data whenever the number of values in categorical features (cardinality) is high.

This thesis proposes a solution for mixed data with potentially high cardinality categorical features. A new kernel for heterogeneous data is introduced, having good runtime and memory results on data with higher cardinality. Here, I introduce *categorical dot product*, imitating dot product after OHE, same as *categorical Euclidean distances*, imitating classical approach after OHE. This *heterogeneous kernel* can work as linear, polynomial, and RBF kernel.

Results of measurements have shown how this kernel can decrease the runtime and lower the memory consumption if the dataset contains both numerical and categorical features of high cardinality. I also demonstrated this fact on real datasets.

Keywords Nominal data, categorical data, heterogeneous data, kernel methods, classification, support vector machine, kernel ridge regression, high-cardinality, missing data, missing values, data preprocessing

Contents

List of Figures

List of Tables

List of Listings

Introduction

Heterogeneous data, considered in this thesis, are a mixture of numerical data and categorical (nominal) data. Categorical features have a cardinality (defining how many values the feature has), which can be high and hard to deal with. There are well known preprocessing methods which can be applied when dealing with heterogeneous data with high-cardinality categorical features. Similar to no-free-lunch-theorem, all methods need to be applied and tried separately for every single dataset – we do not have a universal solution.

There is no kernel in Scikit-learn library [4] designed for heterogeneous data. This thesis proposes a plug-in replacement for Scikit kernels that requires no preprocessing of heterogeneous data. One can process categorical data with less memory consumption, smaller runtime and less tweaking of the used model.

In this thesis, I will introduce some methods of preprocessing categorical data (as a part of the heterogeneous data), and for the numerical data. Then I will talk about statistical models used for prediction, mostly about kernel models. In the end, I will introduce a kernel designed by my own, usable for dealing with heterogeneous data, hand-by-hand with experiments and runtime/memory results.

Goals

To gain some basic knowledge, I will research how statistical kernel models work, how they process the data, etc. When dealing with categorical features, many preprocessing techniques are used, I will describe and compare them.

I will compare my designed kernel(s) with other methods numerically. Then I will provide some tests and comparisons with other preprocessing methods. I will describe my methods in detail. My kernels will be able to deal with a combination of numerical and categorical features, high cardinality categorical features (in reasonable time), and missing values in categorical features.

CHAPTER 1

Kernel Models

In pattern analysis, we want to find similarities in our data. We can pass them into algorithms, which are effective, robust and can spot (possibly linear) patterns in given data. We can call the data-points in a given space as vectors in a feature space. If needed, we can transform our feature space into another, possibly with more dimensions. Here, the kernel-trick is used, where we compute the similarity measure (e.g., dot product) of two vectors in other feature space, with no need for transformation. This trick is done for capacity and efficiency reasons. [2]

I will discuss some of the models, which use kernel-trick and kernel methods in general below.

1.1 The Maximal Margin Classifier

Before we can move to the definition of the support vector machine, we need to introduce the problem of binary classification.

Let's start with a simple case, with target class variable (labels) $\boldsymbol{Y}$, where $Y_i \in \{\pm 1\}$ at values $\boldsymbol{x}_1, ..., \boldsymbol{x}_p$ of features $X_1, ..., X_p$ are composing the training set

$$\mathcal{S} = \{(\boldsymbol{x}_1, Y_1), ..., (\boldsymbol{x}_N, Y_N)\}. \tag{1.1}$$

Then we want to find a prediction function

$$f(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{w}^T \boldsymbol{x} + b), \tag{1.2}$$

such that $\mathbb{E}(0.5|f(\boldsymbol{x}) - Y|)$. [1]

1.1.1 Linearly Separable Case

The precondition of this approach says that points need to be linearly separable, which means there is no noise between positive and negative points. [2]

In $\mathbb{R}^2$ this would be a line drawn between positive and negative points as seen in Fig. 1.1a.

We want to find a hyperplane

$$\boldsymbol{w}^T \boldsymbol{x}_i + b, \tag{1.3}$$

$\forall i \in \{1, \dots, N\}$, separating the positive and negative points. The weigh vector of the hyperplane in Eq. 1.3 is $\boldsymbol{w}$. The $\frac{|b|}{||\boldsymbol{w}||}$ is a perpendicular distance from the origin to the hyperplane (also called an offset). [5] By $||.||$ we mean Euclidean norm.

Let d_+ (resp. d_-) be the shortest distance from the closest positive (resp. negative) point to the hyperplane. The sum of these distances is called the "margin": $d_+ + d_-$. This algorithm tries to find so-called support vectors, which will define the largest "margin" possible. Best with normalized and standardized points, these hyperplanes satisfy the following [1]:

$$\boldsymbol{w}^T \boldsymbol{x}_i + b \geq +1, \text{for} : Y_i = +1 \tag{1.4}$$

$$\boldsymbol{w}^T \boldsymbol{x}_i + b \leq -1, \text{for} : Y_i = -1, \tag{1.5}$$

defining anything on or above (resp. below) this boundary is of one class, with label 1 (resp. -1). These can be combined into the following set of inequalities:

$$Y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \geq 0, \quad \forall i \in \{1, \dots, N\}. \tag{1.6}$$

These inequalities require to compute the appropriate scale for $\boldsymbol{w}$ and b, considering the points, for which the Eq. 1.4 holds. These points lie on hyperplane:

$$\boldsymbol{w}^T \boldsymbol{x}_i + b = +1. \tag{1.7}$$

Similarly, the points that satisfy the Eq. 1.5 lie on the hyperplane

$$\boldsymbol{w}^T \boldsymbol{x}_i + b = -1. \tag{1.8}$$

Hence $d_+ = d_- = \frac{1}{||\boldsymbol{w}||}$, so the margin is $\frac{2}{||\boldsymbol{w}||}$. To minimize the distance between 1.4 and 1.5 we need to solve the following problem:

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimize}} \quad & \frac{\|\boldsymbol{w}\|^2}{2} \\ \text{subject to} \quad & \text{constraints from Eq. 1.6.} \end{aligned} \tag{1.9}$$

Bigger the margin ($\frac{2}{||\boldsymbol{w}||}$), bigger the generalization power of the final classifier. In the case of linearly separable data, we want to determine the points, which will satisfy the Eq. 1.6. [5]

Expression 1.9 is a quadratic optimization problem, which can be solved using standard quadratic programming optimization methods. [2] The expression 1.3 measures the length of the perpendicular projection of points $\boldsymbol{x}_i$, onto the line determined by $\boldsymbol{w}$. Given unknown points $\boldsymbol{z}_j$, we can decide, on which side of the separating hyperplane each one lays by computing $\operatorname{sgn}\left(\boldsymbol{w}^T\boldsymbol{z}_i + b\right)$. This is also called the hard margin. [2]

That is how we can predict, which point belongs to which class. We can see an example in Fig. 1.1a. [2]

1.1.2 Non-Separable case

Most of the time we are working with data, which are not separable – data are noisy and given classes are overlapping, so we cannot find the separating hyperplane. In this case, we want to find such hyperplane that will be less erroneous when predicting. Therefore we need to relax the constraints 1.4 and 1.5, by adding a nonzero slack variables ξ_i. These constraints then become: [5] [2]

$$\boldsymbol{w}^T\boldsymbol{x}_i + b \geq +1 - \xi_i, \text{for} : Y_i = +1 \tag{1.10}$$

$$\boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1 + \xi_i, \text{for} : Y_i = +1 \tag{1.11}$$

$$\xi_i \geq 0, \forall i \in \{1, \dots, N\}. \tag{1.12}$$

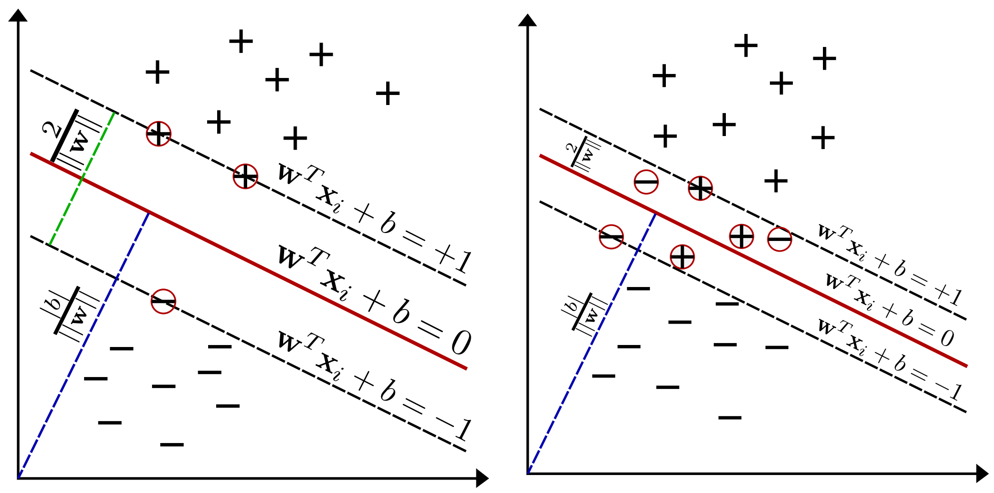
if $\xi = 0$, we will get the previous version, for separable data.

Now we need to deal with the extra cost of errors, which will change the objective function from 1.9 that we want to minimize, to:

$$\begin{aligned} \underset{\boldsymbol{w},\boldsymbol{\xi}}{\text{minimize}} \quad & \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=0}^{n}\xi_i \\ \text{subject to} \quad & Y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i \geq 0, \\ & \xi_i \geq 0, \quad \forall i \in \{1, \dots, N\}, \end{aligned} \tag{1.13}$$

where C is constant, also called a regularization parameter. The constraints are similar to Eq. 1.6, but the slack variables were added. Same as other hyperparameters, C needs to be tuned for each model and dataset separately. The greater the C, the higher error penalization. [1] We can see the error penalization in Fig. 1.1b

Simple example of the margin is shown in Fig. 1.1. Here, pluses are positive points; minuses are negative points. Separating hyperplanes are red, margins are green, offset is blue, support vectors are in red circles. No violation of the margin is shown in Fig. 1.1a (linearly separable data). Slack variables are used in Fig. 1.1b. Those points which violate the margin also become support vectors (circled).


$\frac{2}{\|\mathbf{w}\|}$
$\boldsymbol{w}^T\boldsymbol{x}_i + b = +1$
$\boldsymbol{w}^T\boldsymbol{x}_i + b = 0$
$\boldsymbol{w}^T\boldsymbol{x}_i + b = -1$
$\frac{|b|}{\|\mathbf{w}\|}$

(a) Linearly separable data

(b) Non separable data

Figure 1.1: Sample data with margins and separating hyperplane. [1], [2]

1.2 Ridge Regression

Linear ridge regression is a classical statistical problem that aims to find a linear function that models the dependencies [6] between values $x_1, \ldots, x_p$ of features $X_1, \ldots, X_p$ determining the target variable Y, by

$$Y = \boldsymbol{w}^T\boldsymbol{x} + \varepsilon. \tag{1.14}$$

In Eq. 1.14, $\boldsymbol{w} = (w_0, w_1, \ldots, w_p)^T$ are unknown parameters (the weigh vector), $\boldsymbol{x} = (1, x_1, \ldots, x_p)^T$, and ε is a random variable with the expectation $E(\varepsilon) = 0$. [7]

We aim to find $\boldsymbol{w}$ by its estimate $\hat{\boldsymbol{w}}$ and then estimate $\hat{Y}$ by:

$$\hat{Y} = \hat{\boldsymbol{w}}^T\boldsymbol{x}. \tag{1.15}$$

Therefore, we want to find the parameters similar to $\boldsymbol{w}$, so the error of the model will be the smallest possible. Then we will use this as $\hat{\boldsymbol{w}}$. We can define the error as a loss function $L : \mathbb{R}^2 \to \mathbb{R}$, which we will apply to the true value of Y and the corresponding value $\hat{Y}$. The most common is the quadratic loss function: $L(Y, \hat{Y}) = L(Y - \hat{Y})^2$. [8]

A training set of N sample pairs $\{(\boldsymbol{x}_i, Y_i), \ldots, (\boldsymbol{x}_N, Y_N)\}$, are drawn independently from the same model, written in matrix form as [8]:

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{w} + \boldsymbol{\varepsilon}, \tag{1.16}$$

where

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix}, \quad \boldsymbol{Y} = \begin{pmatrix} Y_1^T \\ \vdots \\ Y_N^T \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{pmatrix}, \quad E(\boldsymbol{\varepsilon}) = 0. \tag{1.17}$$

During the training, we want to minimize the residual sum of squares (RSS):

$$\text{RSS}(w) = \sum_{i=1}^{N} L(Y_i, \boldsymbol{w}^T \boldsymbol{x}_i) = \sum_{i=1}^{N} (Y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 = ||\boldsymbol{Y} - \mathbf{X}\boldsymbol{w}||^2, \tag{1.18}$$

where the minimum is given by setting $\nabla \text{RSS}(\boldsymbol{w}) = \mathbf{0}$, getting the normal equations [8]:

$$\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{X}\boldsymbol{w} = \mathbf{0}. \tag{1.19}$$

If $\mathbf{X}^T\mathbf{X}$ is regular, there is only one solution minimizing $\text{RSS}(\boldsymbol{w})$:

$$\hat{\boldsymbol{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{Y}. \tag{1.20}$$

If not, we may introduce a regularization term to $\text{RSS}(\boldsymbol{w})$, which leads to **ridge regression** [7]:

$$\text{RSS}(\boldsymbol{w}) = ||\mathbf{Y} - \mathbf{X}\boldsymbol{w}||^2 + \lambda \mathbf{I}'\boldsymbol{w}, \tag{1.21}$$

where $\mathbf{I}' \in \mathbb{R}^{p+1,p+1}$ is an identity matrix, $\mathbf{I}'_{1,1} = 0$, and $\lambda > 0$.

Therefore normal equations are given by:

$$\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{X}\boldsymbol{w} - \lambda\mathbf{I}'\boldsymbol{w} = \mathbf{0}, \tag{1.22}$$

and since $\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I}'$ is regular for every $\lambda > 0$, there is exactly one solution [7]

$$\hat{\boldsymbol{w}}_\lambda = (\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I}')^{-1}\mathbf{X}^T\boldsymbol{Y}. \tag{1.23}$$

The prediction of $\hat{\boldsymbol{Y}}$ of the unseen data $\boldsymbol{z}$ is [8]:

$$\hat{\boldsymbol{Y}} = \hat{\boldsymbol{w}}_\lambda^T \boldsymbol{z}. \tag{1.24}$$

This was the introduction to the ridge regression model. To define kernel ridge regression, we first need to introduce the kernel in general.

1.3 Kernels

Kernel functions are characterized by the property that all finite kernel matrices are positive semi-definite. An equivalent formulation is that the kernel satisfies Mercer's conditions. It also needs to reproduce kernel Hilbert space. [2]

Again, we are given some training data

$$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N) \in \mathbf{X} \times \boldsymbol{Y}, \tag{1.25}$$

where $\mathbf{X}$ is an input domain, and $\boldsymbol{Y}$ is the target (or class) of the domain (recalling to Sec. 1.1). In the case of binary pattern recognition, given some new input $\boldsymbol{z} \in \mathbf{X}$, we want to predict the corresponding $Y_z \in \pm 1$. Loosely speaking, we want to choose Y_z such that $(\boldsymbol{z}, Y_z)$ is in some sense similar to the training examples. To this end, we need similarity measures in $\mathbf{X}$ and $\{\pm 1\}$. The latter is easier, as two target values can only be identical or different. For the former, we require a function

$$k : \mathbf{X} \times \mathbf{X} \to \mathbb{R}, \qquad (\boldsymbol{x}_i, \boldsymbol{x}_j) \mapsto k(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{1.26}$$

satisfying $\forall i, j \in \{1, \ldots, N\}, \quad \forall \boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbf{X}$, and

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j), \tag{1.27}$$

where $\phi(\cdot)$ maps into some dot product space $\mathcal{H}$, called the feature space. The similarity measure $k(\cdot, \cdot)$ is usually called a **kernel**, and ϕ is called its *feature map.* [9]

There are examples of positive definite kernels which can be be efficiently evaluated even though they correspond to dot products in infinite dimensional dot product spaces. In such cases, substituting $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ is crucial and is called the *kernel trick.* [9]

Given a kernel k and inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbf{X}, \forall i, j \in \{1, \ldots, N\}$ we can define $N \times N$ sized matrix $\mathbf{G}$. This is called the **Gram** matrix (or kernel matrix) of k with respect to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. The i, j-th element of $\mathbf{G}$ is defined as: [9]

$$\mathbf{G}_{i,j} = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{1.28}$$

1.4 Suppor Vector Machine (SVM)

The key features of SVMs are the use of kernels, the absence of local minima, the sparseness of the solution and the capacity control obtained by optimizing the margin. [2]

In SVM, transforming the training set space with ϕ, we want to find the hyperplane similar to Eq. 1.3:

$$\boldsymbol{w}^T \phi(\boldsymbol{x}) + b, \tag{1.29}$$

and define the set of inequalities (boundaries of the separation), similar to Eq. 1.6 again, already with the slack variables, because most of the time we are working with noisy data:

$$Y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 + \xi_i \geq 0, \quad \forall i \in \{1, \ldots, N\}, \tag{1.30}$$

which brings us to redefining the optimization problem, similar to expression 1.13: [10]

$$\begin{aligned} \underset{\boldsymbol{w}, b, \boldsymbol{\xi}}{\text{minimize}} \quad & \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=0}^{n} \xi_i, \\ \text{subject to} \quad & Y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 + \xi_i \geq 0, \\ & \xi_i \geq 0, \quad \forall i \in \{1, \ldots, N\}. \end{aligned} \tag{1.31}$$

Using the method of Lagrange multipliers and the duality principle it can be shown that the equivalent Lagrangian dual problem [2] [10] is to

$$\begin{aligned}
&\text{minimize} && \tilde{L}(\boldsymbol{a}) = \sum_{i=1}^{N} a_n - \frac{1}{2}\sum_{i,j=1}^{N} a_i a_j Y_i Y_j, k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&\text{with respect to} && a_1, \dots, a_N, \\
&\text{subject to} && 0 \leq a_i \leq C \quad \text{and} \quad \sum_{i=1}^{N} a_i Y_i = 0.
\end{aligned} \tag{1.32}$$

The solution is called support vector machine (SVM), and the hyperplane we are looking for is taking form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i \boldsymbol{Y}_i k(\boldsymbol{x}_i, \boldsymbol{x}_j) + b, \tag{1.33}$$

and a prediction of given $\boldsymbol{z}$ is given by $\hat{\boldsymbol{Y}}(\boldsymbol{z}) = \operatorname{sgn}(f(\boldsymbol{z}))$. [10]

1.5 Kernel Ridge Regression

Given some input training set $\mathbf{X}$, $\boldsymbol{Y}$ and $\boldsymbol{\varepsilon}$ are defined as in Eq. 1.17, where $\boldsymbol{\Phi} = \left(\boldsymbol{\phi}(\boldsymbol{x}_1)^T, \dots, \boldsymbol{\phi}(\boldsymbol{x})^T\right)^T$ we can determine the target variable in a matrix form: [8]

$$\boldsymbol{Y} = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\varepsilon}, \tag{1.34}$$

Similarly, as in ridge regression, Eq. 1.21, we minimize RSS, but in its dual representation [8]:

$$\mathrm{RSS}_\lambda(\boldsymbol{w}) = ||\mathbf{Y} - \boldsymbol{\Phi}\boldsymbol{w}||^2 + \lambda \boldsymbol{w}^T \boldsymbol{w}. \tag{1.35}$$

The parameters $\boldsymbol{w}$ are taking form $\boldsymbol{\Phi}^T\boldsymbol{\alpha}$, where $\in \mathbb{R}^N$. Inserting this into $\mathrm{RSS}_\lambda(\boldsymbol{w})$ we get:

$$\begin{aligned}
\mathrm{RSS}_\lambda(\boldsymbol{\alpha}) &= ||\mathbf{Y} - \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\alpha}||^2 + \lambda \boldsymbol{\alpha}^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \boldsymbol{\alpha} \\
&= ||\mathbf{Y} - \mathbf{G}\boldsymbol{\alpha}||^2 + \lambda \boldsymbol{\alpha}^T \mathbf{G}\boldsymbol{\alpha}
\end{aligned} \tag{1.36}$$

with Gram matrix $\mathbf{G}$, where normal equations are similar to Eq. 1.22:

$$\begin{aligned}
\boldsymbol{\Phi}^T\boldsymbol{Y} - \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} - \lambda\mathbf{I}'\boldsymbol{w} &= \mathbf{0} \\
\mathbf{G}(\boldsymbol{Y} - \mathbf{G}\boldsymbol{\alpha} - \lambda\boldsymbol{\alpha}) &= \mathbf{0}.
\end{aligned} \tag{1.37}$$

Hence the matrix $(\mathbf{G} + \boldsymbol{\lambda}\mathbf{I}')$ is positive definite [8], and $\lambda > 0$, there is always exactly one solution, and we get $\hat{\boldsymbol{\alpha}}$ as an estimate of $\boldsymbol{\alpha}$:

$$\hat{\boldsymbol{\alpha}} = (\mathbf{G} + \boldsymbol{\lambda}\mathbf{I}')^{-1}\boldsymbol{Y}. \tag{1.38}$$

The prediction of unknown vector $\boldsymbol{z}$ is then

$$\boldsymbol{Y}(\boldsymbol{z}) = \sum_{i=1}^{N} \hat{\alpha}_i k(\boldsymbol{x}_i, \boldsymbol{z}) = \hat{\boldsymbol{\alpha}}_i k(\boldsymbol{z}), \tag{1.39}$$

where $k(\boldsymbol{z}) = (k(\boldsymbol{x}_1, \boldsymbol{z}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{z}))^T$. [6] [8]

1.6 Existing Kernels

Two key properties are required for an application of a kernel function. Firstly, it should capture the measure of similarity appropriate to the particular task and domain, and secondly, its evaluation should require significantly less computation than would be needed in an explicit evaluation of the corresponding feature mapping ϕ. [5]

1.6.1 Dot (Linear)

Let $\mathcal{V}$ be a vector space. We define an operation $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ a scalar multiplication (denoted as $\boldsymbol{x}^T\boldsymbol{y}$ in a vector form), if it satisfies properties: commutative, distributive, scalar multiplication, and $\boldsymbol{x}^T\boldsymbol{x} \geq 0, \boldsymbol{x}^T\boldsymbol{x} = 0 \Leftrightarrow \boldsymbol{x} = 0$. [11]

Dot kernel is a simple dot product (scalar multiplication) of two points in Hilbert's space, defined as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T\boldsymbol{y}. \tag{1.40}$$

It is also called a *linear kernel.* [2] For visualization see Fig. 1.1.

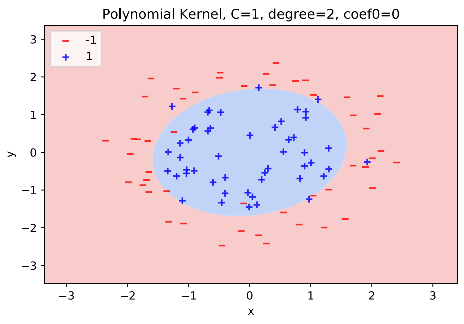
1.6.2 Polynomial

Here the *kernel trick* is done, and the kernel operates in higher dimensional space than the original feature space. Definition:

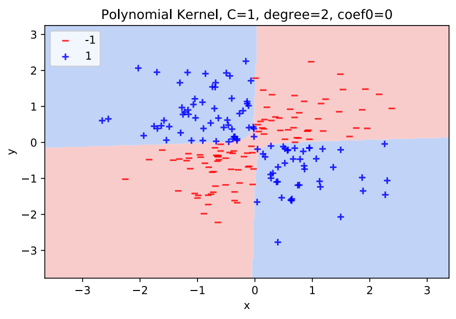
$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + a)^d \tag{1.41}$$

Where $a \in \mathbb{N}$ and $d \in \mathbb{N}\backslash\{0\}$ are parameters of this kernel. This is a special case of more general kernel $k(\boldsymbol{x}, \boldsymbol{y}) = p\left(k(\boldsymbol{x}, \boldsymbol{y})\right)$ where $p(\cdot)$ is any polynomial with positive coefficients. [2]

The example of a polynomial kernel in SVM is shown in Fig. 1.2



(a) Circles problem



(b) XOR problem

Figure 1.2: Classification problems on various data, solved by SVM. Comparison of the same polynomial kernel on various data.

1.6.3 Gaussian

Gaussian kernels are widely used and studied in neighboring fields. [2] This is also called radial basis function (RBF) kernel. The kernel is defined as:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||^2}{2\sigma^2}\right), \tag{1.42}$$

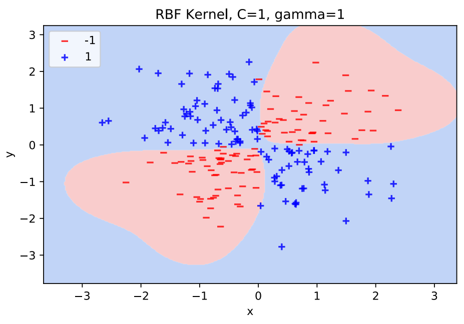
which is a special case of $k(\cdot,\cdot) = \exp(k(\cdot,\cdot))$. It can be simplified to:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma||\boldsymbol{x}-\boldsymbol{y}||^2\right), \tag{1.43}$$

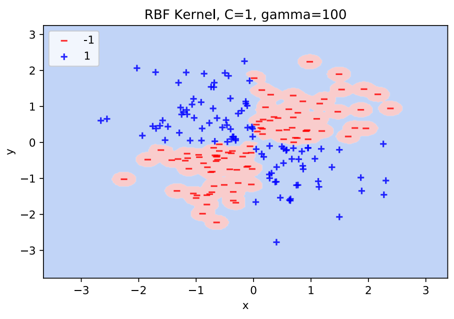
where γ is equal to $\frac{1}{2\sigma^2}$ and by $||\boldsymbol{x}-\boldsymbol{y}||$ we denote a Euclidean distance between two vectors. [2]

The parameter σ controls the flexibility of the kernel in a similar way to the degree d in the polynomial kernel, as we can see in Fig. 1.3. Small values of σ correspond to large values of d since, for example, they allow classifiers to fit any labels, hence risking overfitting, as seen in Fig. 1.3b. In such cases, the gram matrix becomes close to the identity matrix. On the other hand, large values of σ gradually reduce the kernel to a constant function, making it impossible to learn any non-trivial classifier. [2]

The feature space has infinite-dimension for every value of σ, but for large values, the weight decays very fast on the higher-order features. In other words, although the rank of the gram matrix will be full, for all practical purposes the points lie in a low-dimensional subspace of the feature space. [2]



(a) Average generalization



(b) Bad generalization (overfitted model)

Figure 1.3: XOR Classification problem, solved by SVM using RBF kernel, same as in Fig. 1.2.

1.6.4 Delta

This kernel belongs to the category of locally stationary kernels. The name and definition comes from the Kronecker delta

$$k(\boldsymbol{x},\boldsymbol{y}) = \delta_{\boldsymbol{x},\boldsymbol{y}} = \begin{cases} 1, & \text{if } \boldsymbol{x} = \boldsymbol{y}, \\ 0, & \text{if } \boldsymbol{x} \neq \boldsymbol{y}. \end{cases} \tag{1.44}$$

This kernel is positive definite [12].

1.6.5 Other

In previous subsections I introduced only basic kernels in closed form, working mostly with real numbers. There are several other categories of kernels, such as kernels for text, graphs, structured data, kernels from generative models, etc. [2]. However, in this thesis, I am not using any of these.

1.7 Multi-kernel

Simple operations that combine simpler kernels can construct more complex kernels. If the created kernel preserves its positive semi-definiteness, we say that it is closed under such operations. [2]

Closure properties: Let k_1 and k_2 be kernels over $\mathbf{X} \times \mathbf{X}, \mathbf{X} \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^+$, $f(\cdot)$ a real-valued function on $\mathbf{X}$, $\phi : \mathbf{X} \to \mathbb{R}^N$ with k_3 a kernel over $\mathbb{R}^N \times \mathbb{R}^N$, and $\mathbf{B}$ a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels: [2]

i $k(\boldsymbol{x}, \boldsymbol{y}) = k_1(\boldsymbol{x}, \boldsymbol{y}) + k_2(\boldsymbol{x}, \boldsymbol{y})$,

ii $k(\boldsymbol{x}, \boldsymbol{y}) = ak_1(\boldsymbol{x}, \boldsymbol{y})$,

iii $k(\boldsymbol{x}, \boldsymbol{y}) = k_1(\boldsymbol{x}, \boldsymbol{y})k_2(\boldsymbol{x}, \boldsymbol{y})$,

iv $k(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x})f(\boldsymbol{y})$,

v $k(\boldsymbol{x}, \boldsymbol{y}) = k_3(\phi(\boldsymbol{x}), \phi(\boldsymbol{y}))$,

vi $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \mathbf{B} \boldsymbol{y}$.

The proof of these properties for construction of new kernels is in [2].

Chapter 2

Data Preprocessing

In this chapter, I will introduce and describe some of the preprocessing methods used nowadays in machine learning and data analysis. There are many more, but I will explain in detail those, which are used in my designed method. Some of these methods are designed for categorical data:

2.1 Categorical Features

Categorical data (also called *nominal*) occur when we are talking about some information, which we cannot order. Apart from ordering, no basic operations could be done with the categorical values themselves. These can be, e.g., postal zip codes, gender, yes/no answers. [13]

We cannot say that the postal code of Prague 6: 160 00 is higher than Prague 1: 110 00. Same as we cannot add 5000 to get Prague 6's postal code from Prague 1's.The better way to think of postal codes may be looking to a distance to the city center or the distance to the nearest city, etc. When working with these types of data, one should be aware of the problem trying to solve and adjust the data in a correct way. When there is no such transformation available, we can think of some other techniques, which I will describe below. [14]

When talking about a categorical feature, we denote it a set S, which contains c categories. If c is a large number, some techniques such as one-hot encoding can increase the dimensionality of the dataset rapidly. Standard methods and models may have problems with high dimensional datasets – the curse of dimensionality – mostly those using metrics. [15]

2.2 One-Hot Encoding (OHE)

This method is designed mostly for categorical features. We need to add at least c more columns to indicate an occurrence of an item from S in each row.

This technique is pretty old; it is used in electronics, same as statistics and economics. Newly created features are also called dummy variables. [16]

For example, let us have a set of four postal zip codes

$$S = \{160\ 00, 110\ 00, 120\ 00\} \tag{2.1}$$

Which we can encode to:

$$\begin{aligned} 160\ 00 &\quad\rightarrow\quad (1, 0, 0) \\ 110\ 00 &\quad\rightarrow\quad (0, 1, 0) \\ 120\ 00 &\quad\rightarrow\quad (0, 0, 1) \end{aligned}$$

Order of zeros and ones does not matter, but we need to determine it before transforming into one-hot-encoding.

This technique is also well prepared for missing values. If there is any, transformed variable from set S of $c = 3$ unique values would be either $(0, 0, 0)$ or you can add another column as an indicator of missing value: $(0, 0, 0, 1)$.

If the categorical data are of high cardinality, the dimension is getting higher and too big for evaluation (take in mind limited RAM and long runtime). Sparse matrices may come in handy, which can reduce memory consumption. However, sparse matrices, in general, cannot store more than one type, so it is still consuming much memory. If you have a continuous feature in your data, which is of type double, also the dummy variables need to be of type double, which is not very efficient in terms of memory [17].

2.3 Target Encoding

The idea is to transform categorical feature to the means of the target (which can be a binary, n-ary, continuous feature, etc.). For each distinct element x in S, we need to compute the average of the corresponding values in the target. Fig. 2.1 shows a simple example of a transformation of the data. [14]

In Python, we can easily do this, using pandas library (`df` is pandas [18] DataFrame object, `'x'` is the name of the categorical column, `'y'` is the target column):

```python
df['x'] = df['x'].map(df.groupby('x')['y'].mean())
```

This is the most basic approach. However, as you can see in Fig. 2.1b, the last value of column x_1 is zero, because there was no other target class as 0 for this particular value. When dealing with any dataset, we need to take in mind, that we have only a part of the all possible data. In x_1 we can see that the last value is encoded as 0 because there is only one instance of this type of variable. This effect is called overfitting. [3]

There are several approaches on how to deal with overfitting. We can compute the mean in each one of k-folds when cross-validating. Alternatively, we can use additive smoothing. The smoothed mean itself is determined as:

$$\hat{x} = \frac{n \cdot \bar{x} + m \cdot w}{n + m}, \tag{2.2}$$

where $\hat{x}$ is the mean we are trying to compute, n is the number of values, $\bar{x}$ is the mean estimated by the previous group-by function, m is the "weight" we want to assign to our overall mean, and w is the overall mean. In Fig. 2.1c we can see encoded features when additive smoothing is applied. [14] [3]

x_0	x_1	y
a	a	1
a	a	1
a	a	1
a	a	1
a	a	0
b	a	1
b	a	0
b	a	0
b	a	0
b	b	0

(a) Sample data

x_0	x_1	y
0.8	0.556	1
0.8	0.556	1
0.8	0.556	1
0.8	0.556	1
0.8	0.556	0
0.2	0.556	1
0.2	0.556	0
0.2	0.556	0
0.2	0.556	0
0.2	0.0	0

(b) Simple encoding

x_0	x_1	y
0.6	0.526	1
0.6	0.526	1
0.6	0.526	1
0.6	0.526	1
0.6	0.526	0
0.4	0.526	1
0.4	0.526	0
0.4	0.526	0
0.4	0.526	0
0.4	0.455	0

(c) Smoothed encoding

Table 2.1: A simple example of target encoding. [3]

In Tab. 2.1 y is a target variable, x_0, x_1 are the categorical features. Tab. 2.1a shows the sample data, Tab. 2.1b shows a simple transformation of the data, and Tab. 2.1c shows the smoothed target encoding, with "weight" $w = 10$. [3]

2.4 Feature Scaling

Most of the times, the dataset will contain features highly varying in magnitudes, units, and range. Since some of the machine learning algorithms use distance metrics in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary hugely, e.g., between different weight units (5kg and 5000g). The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling. [19]

These common methods are used to perform feature scaling:

- **Standardization** (or **Z-score normalization**) replaces the values in each feature by its Z-scores:

$$z = \frac{x - \overline{x}}{\sigma}, \tag{2.3}$$

where x' is the computed value, $\overline{x}$ is the average of the feature and σ is the standard deviation. Features will have the properties of a standard normal distribution with $\mu = 0, \sigma = 1$. This is important not only if we are comparing measurements that have different units but can also make some of the algorithms faster and more accurate. [20]

- **Rescaling (min-max normalization)** is an alternative approach to Z-score normalization. Here, the data are scaled to a fixed range - usually 0 to 1:

$$x' = \frac{x - \min x}{\max(x), \min(x)}. \tag{2.4}$$

The cost of having the limited range – in contrast to standardization – is that we will end up with smaller standard deviations, which can suppress the effect of outliers. [20]

Chapter 3

Method And Data Description

In this chapter, I will introduce my method on how to deal with categorical features, and heterogeneous data, where the features are mixed. Then I will introduce the datasets used as empirical proof, that my method works with results that are equal to the vastly used methods, with less memory consumption and smaller runtime.

3.1 Analysis

When using the standard linear kernel, which is a dot product, we cannot provide a scalar multiplication of two vectors with various data types. Categorical features are mostly represented as strings. Usually, we will create dummy variables (described in Sec. 2.2), and then we can deal with them appropriately when providing a dot product of two vectors (with the transformed string data type). What will happen in a dot product of two encoded categories is conditioning their equality. Small example:

Let us have a categorical feature containing three values (nationality): *SK, CZ, PL.* If we provide dummy variables for this kind of feature, it will look like:

$$\begin{aligned} \text{SK} &\rightarrow (1,0,0) \\ \text{CZ} &\rightarrow (0,1,0) \\ \text{PL} &\rightarrow (0,0,1) \end{aligned}$$

This set consists of unique values from a feature in our dataset (in this example there are three values). Let us see, how does the dot products of various combinations of the values from the set after OHE look like:

$$\begin{aligned} \langle SK, CZ \rangle &= (1,0,0) \cdot (0,1,0)^T = 0 \\ \langle PL, PL \rangle &= (0,0,1) \cdot (0,0,1)^T = 1 \end{aligned}$$

Here we can see, how the dot product of transformed variables only represents the equality of two categorical values. We can generalize this fact for

p dimensions of categorical features. Let's see, how this may look like, if we will have two categorical features:

$$\begin{aligned}
\langle (SK, CZ), (CZ, CZ) \rangle &= (1,0,0,0,1,0) \cdot (0,1,0,0,1,0)^T &= 1 \\
\langle (PL, SK), (PL, SK) \rangle &= (0,0,1,1,0,0) \cdot (0,0,1,1,0,0)^T &= 2
\end{aligned}$$

When we look closely on the example, this is just the comparison of values on corresponding indexes in the vectors and summing number of ones on the same positions in them.

3.2 OHE Representation

In the example from the previous section, we saw how one-hot-encoding increased the dimensionality. Without loss of generality, assume that the set of categories, in each of the categorical feature in our dataset, is finite. Therefore we can "label" the categories in the set. This is also called string imputing. Categorical data type in Pandas library [18] makes possible to grab the "codes" of unique categories. Therefore we can compare these values faster (with no need of inefficient string comparison), which is the core of the categorical kernel (which I will describe later).

In the scikit-learn library, we can find many encoders. The one used for OHE can be found in `sklearn.preprocessing.OneHotEncoder`. However, I do not use it in my implementation.

In Pandas library [18] we can find Data Frame object, which has many applications. It can store the dataset more comprehensively. Pandas library has a built-in function `get_dummies`, which encodes all columns of type `category` to dummy variables (OHE). This conversion is displayed in Lis. 1.

```python
import pandas as pd

df = pd.DataFrame(X)
for col in categoricals:
    df[col] = df[col].astype('category')
df_ohe = pd.get_dummies(df, dummy_na=True)
```

Listing 1: Generating dummy variables from categorical columns in DataFrame.

In Lis. 1, X may be a dictionary or n-dimensional array. This is how the dataset is roughly represented. Variable `categoricals` is an array of column names, representing the names of categorical features. `df_ohe` is the output DataFrame.

3.3 Naive Heterogeneous kernel

The first implementation (which is rather explanatory) is a for-loop, where I first iterate through all scalar values of given vectors and check if the scalar values are categorical. We can see it in Lis. 2

```
import numpy as np

def naive_heterogeneous_kernel(x, y, categories=None):
    if categories is None:
        return np.dot(x, y)
    acc = 0
    for i in range(0, len(x)):
        if (categories[i]):
            acc += x[i] == y[i]
        elif (x[i] is np.nan) or (y[i] is np.nan):
            acc += x[i] != y[i]
        else:
            acc += x[i] * y[i]
        return acc
```

Listing 2: The naive implementation of the heterogeneous kernel.

If the values (on same positions in both vectors) are equal, I will add 1 to the accumulator (`acc`) value, else 0 (the result of operator `==` in Python). If the value is of a numerical type, I will multiply the value on the same position in the second vector to imitate dot product. If two `NaN` (NumPy's Not a Number) numbers are compared, the output of this comparison is `False`, which is represented as 0. Therefore there is another if statement to mimic equality as `True` when taking the categorical or numerical feature's equality measure. The Python code is shown in Lis. 2.

Here the parameters `x`, `y` are given vectors, `categories` is a mask of categorical features (`True`/`False` values on corresponding positions). `np` stands for NumPy [21] package. As you can see, the condition is also expanded for dealing with missing values. I will treat the missing value as another categorical value.

Python's for-cycles are very slow compared to NumPy's vectorization, broadcasting, etc. We can see this in Fig. 4.2, where I compare OHE and NumPy's dot product over this naive implementation of the heterogeneous kernel. Therefore I needed to move from the implementation in Lis. 2 into another.

3.4 Categorical Kernel

SVM implemented in scikit-learn [4] takes either precomputed gram matrix or a callable object. Precomputed Gram matrix is done by iterating through all vectors of two given matrices, giving the dot product of all combinations of vectors. In Lis. 3 we can see the implementation of the categorical kernel.

The function defined in Lis. 3 is taking two matrices and outputs the gram matrix. First, we iterate through each vector in `X1`, and using NumPy's broadcasting [21] we will sum the number of same values on each same position for each pair of vectors (from `X2`).

```python
def categorical_gram_dot(X1, X2):
    gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i in range(X1.shape[0]):
        gram_matrix[i, :] = np.sum(X1[i, :] == X2, axis=1)
    return gram_matrix
```

Listing 3: The implementation of the categorical dot kernel.

Before using this kernel, one must provide input matrices of labels of the categories, because the comparison of strings is expensive. We can do this simply using pandas DataFrame with categorical columns of data type `category`. We can get the labels of the categories from the given column by calling `column.cat.codes`. First seen category will have a smaller number than others. The missing values will have a label -1, so I treat it as a separate value.

This is how I mimic the dot product of categorical features after OHE. The polynomial kernel uses the dot product, so we can use this categorical dot product also there, as you can see in Eq. 1.41. Here, only two more parameters need to be provided.

3.5 Categorical Euclidean Distances

RBF kernel uses Euclidean distances, which are computed using NumPy's framework. We can find the implementation of Euclidean distances for numerical vectors in scikit-learn:

```
sklearn.metrics.pairwise.euclidean_distances
```

However, this only works for numerical vectors, not for categorical – my implementation in Lis. 4 of Euclidean distances gives the same results, as you would provide OHE of categorical features and then apply the squared numerical version. I am looking for inequalities, summing their number and multiply the resulting array by 2. We will need this kernel when computing the RBF kernel's Gram matrix.

```python
def categorical_eucl_dist(X1, X2):
    dist_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i in range(X1.shape[0]):
        dist_matrix[i, :] = 2 * np.sum(X1[i, :] != X2, axis=1)
    return dist_matrix
```

Listing 4: The implementation of categorical Euclidean distances.

3.6 Heterogeneous kernel

To preserve similarity measure between numerical and categorical data, I compute the gram matrix as an addition of two kernels: numerical and categorical. The algorithm takes two heterogeneous matrices X1, X2 as an input and outputs the gram matrix. We can see this in Lis. 5:

```python
def heterogeneous_kernel(X1, X2):
    gram = np.zeros((X1.shape[0], X2.shape[0]))
    X1_cat, X1_num = split_data(X1, categoricals)
    X2_cat, X2_num = split_data(X2, categoricals)
    gram += categorical_kernel(X1_cat, X2_cat)
    gram += numerical_kernel(X1_num, X2_num)
    return gram
```

Listing 5: Pseudo Python code for the heterogeneous kernel.

Parameter `categoricals` in Lis. 5 is a mask of indexes of categorical features of the given matrices. `split_data` splits the data into numerical and categorical. Delta and dot kernel were described in Sec. 3.4, 1.3

When there is a need for computing polynomial or RBF kernel, simply change the return value to `np.power(coef0 + gram, degree)` for polynomial kernel. For RBF change the dot kernel into Euclidean distance and categorical kernel to nominal Euclidean distance (Lis. 4) and the return value to `np.exp(-gamma * gram)` as defined in Eq. 1.43.

The output from the heterogeneous kernel, described in Lis. 5 is not only correct in terms of equality to the OHE method (as described in Sec. 4.3), but also satisfies the first condition from the list in Sec. 1.7.

3.7 Data Description

To provide evidence, that my method also works on real datasets, I gathered some from UCI Machine Learning Repository [22] and OpenML repository [23]. They are all heterogeneous, which means they contain not only numerical but also categorical features. Every dataset should boil into a binary classification problem. If not, I will take the majority class as positive and the rest as negative class. It would not be a problem to take all classes and use One-against-one or One-against-all technique as discussed in [24]. However, this is not the target of this thesis, and therefore I binarize the target label in multiclass datasets if needed.

All the datasets described in the Tab. 3.1 are used in OpenML [23] challenges. Here, people try to create the best model for classification (or regression) and try to be first in the leaderboard. I found the leading accuracies for each dataset, using SVM (mostly) with RBF or Polynomial kernel. The approximate values of accuracy are shown in Tab. 3.1.

File Name	Dataset Name	Accuracy in %
adult	Adult	76
bands	Cylinder Bands	86
cmc	Contraceptive Method Choice	56
credit-g	Statlog (German Credit Data)	78
crx	Credit Approval	87
diagnosis	Acute Inflammations	100
heart	Heart Diseas	81
hepatitis	Hepatitis	83
horse-colic	Horse Colic	82
house-votes	Congressional Voting	96
ilpd	ILPD (Indian Liver Patient Dataset)	73
irish	Irish Educational Transitions Data	100
kobe	Kobe Bryant Shot Selection	-
labor	Labor Relations	94
post-operative	Post-Operative Patient	48
profb	Pro Football Scores	76

Table 3.1: Datasets gathered from OpenML [23] and UCI [22] with approximate accuracy from rankings.

The accuracy column in Tab. 3.1 is an approximate accuracy gathered from OpenML rankings (hyperlinks only in pdf). Accuracy for "kobe" is unknown.

File Name	samp	num	cat	sum. card	ave. card
adult	32561	6	8	102	12.75
bands	540	20	18	810	45
cmc	1473	3	6	20	3.333
credit-g	1000	7	13	54	4.154
crx	690	6	9	40	4.444
diagnosis	120	1	5	10	2
heart	270	6	7	19	2.714
hepatitis	155	6	13	73	5.615
horse-colic	368	8	15	398	26.53
ilpd	583	9	1	2	2
irish	500	2	3	15	5
kobe	25697	11	9	1733	192.6
labor	57	8	8	21	2.625
post-operative	90	1	7	19	2.714
profb	672	5	4	61	15.25

Table 3.2: Information about gathered datasets.

In Tab. 3.2 I provide some basic information of the datasets shown in Tab. 3.1. From left to right: nr. of samples, nr. of numerical features, nr. of categorical features, the sum of cardinalities of the categorical features, average cardinality of the categorical features. Some of the columns were dropped, because of small to zero informative value, such as IDs, timestamps, etc.

Chapter 4

Evaluation

In test classes, I am comparing my method with already defined kernels. When comparing dot kernel, I first make OHE on randomly generated data, for each feature. Then I apply NumPy's dot function. The output of this function is the gram matrix. Then I apply my kernel for the same data, and the output Gram matrix is compared with the previous one. I do the same in the comparison of the polynomial kernel. Everything was done in the `test_kernels.py` file, attached. All tests are passing and all kernels implemented by me give identical Gram matrices to the methods by the NumPy's library.

4.1 Accuracy, Cohen's Kappa, AUC-ROC, R2

To measure any prediction results, I use three metrics: **Acc** stands for accuracy, **Kappa** for Cohen's kappa and **AUC** for Area Under Curve - Receiver Operating Characteristic (ROC). R2 is called the coefficient of determination [25] and is used as a metric for regression tasks. I use these metrics not only when making a comparison of the kernels, but also further, where I am looking for the best parameters for the given dataset.

4.2 Measurements Description

All numerical features were scaled by the standardization method (Z-scores). I used scaler from: `sklearn.preprocessing.StandardScaler`. After scaling, each numerical feature has zero mean and unit variance.

Missing values in numerical features were filled by the mean of the feature vector (which is zero after scaling). Missing values in categorical features do not have to be treated. After getting the category codes, missing values are labeled as -1. The implementation of the heterogeneous kernel can work with them and treat them as another category as described in Sec. 3.4 OHE was set to produce a separate column for the missing categorical values.

4.3 Kernel Comparison

To demonstrate the equality of heterogeneous kernel and linear kernel after OHE, I split each dataset from Tab. 3.1 into the train (85%) and test (15%) set, passed each of the kernels as a parameter to SVM or kernel ridge (depends on type of problem), then fit them with training data and obtain mentioned metrics when predicting on the test data.

4.3.1 SVM Classification

Tab. 4.1, C.1, and C.2 demonstrate the equality of the heterogeneous kernel, compared to NumPy's or libsvm version (after OHE).

Dataset Name	OHE & Numpy	OHE & libsvm	Heterogeneous
adult	(0.85, 0.56, 0.91)	(0.85, 0.56, 0.91)	(0.85, 0.56, 0.91)
bands	(0.83, 0.65, 0.89)	(0.83, 0.65, 0.89)	(0.83, 0.65, 0.89)
cmc	(0.66, 0.28, 0.67)	(0.66, 0.28, 0.67)	(0.66, 0.28, 0.67)
credit-g	(0.77, 0.39, 0.79)	(0.77, 0.39, 0.79)	(0.77, 0.39, 0.79)
crx	(0.81, 0.62, 0.88)	(0.81, 0.62, 0.88)	(0.81, 0.62, 0.88)
diagnosis	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
heart	(0.85, 0.7, 0.9)	(0.85, 0.7, 0.9)	(0.85, 0.7, 0.9)
hepatitis	(0.79, 0.46, 0.87)	(0.79, 0.46, 0.87)	(0.79, 0.46, 0.87)
horse-colic	(0.82, 0.59, 0.86)	(0.82, 0.59, 0.86)	(0.82, 0.59, 0.86)
ilpd	(0.79, 0.0, 0.74)	(0.79, 0.0, 0.74)	(0.79, 0.0, 0.74)
irish	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
kobe	(0.99, 0.96, 1.0)	(0.99, 0.96, 1.0)	(0.99, 0.96, 1.0)
labor	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
post-operative	(0.91, 0.0, 0.52)	(0.91, 0.0, 0.52)	(0.91, 0.0, 0.52)
profb	(0.7, 0.33, 0.76)	(0.7, 0.33, 0.76)	(0.7, 0.33, 0.76)

Table 4.1: Comparison of prediction metrics, using SVM with various linear kernels.

Each row in Tab. 4.1, C.1, and C.2 consists of comparison of the different kernels and approaches to make a prediction on the given dataset. Each cell consists of three values: accuracy, Cohen's kappa, and AUC-ROC. Each model is constructed with parameters: $C = 1$. Polynomial kernel has parameters: $a = 0, d = 3$, RBF kernel: $\gamma = 0.1$. OHE & Numpy column shows the case when NumPy is computing the gram matrix. OHE & libsvm column means that, e.g., kernel='linear' is passed into the constructor of SVM and libsvm is taking care of computation of the gram matrix. The heterogeneous column states that the heterogeneous kernel is computing the gram matrix. Similar in other tables in the appendix.

4.3.2 Kernel Ridge Regression

To demonstrate the usability of the designed kernel, I provided a comparison of various approaches to regression task. I fitted Kernel Ridge regressor with data from "servo" dataset gathered from UCI repository [22]. I split this dataset into the train (85%) and test (15%) set. The train set was fitted into the regressor, and then the R2 scoring method on the test set was used. In Tab. 4.2 I compare three approaches: pairwise kernel (the one used when, e.g., `kernel='linear'` is passed into the constructor of Kernel Ridge), NumPy's precomputed gram matrix and gram matrix computed by the heterogeneous kernel. The first two are used after OHE.

In Tab. 4.2, λ is the regularization parameter used in Kernel Ridge regression, a, d are parameters for polynomial kernel and γ is the parameter for RBF kernel. Pairwise polynomial kernel is defined as $K(\mathbf{x}, \mathbf{y}) = (\gamma\langle\mathbf{x}, \mathbf{y}\rangle + a)^d$, therefore I used $\gamma = 1$ for comparison.

Method	λ	a	d	γ	R2
linear-hetero	1				0.612
linear-ohe-np	1				0.612
linear-ohe-pw	1				0.612
poly-hetero	1	1.0	2.0		0.871
poly-ohe-np	1	1.0	2.0		0.871
poly-ohe-pw	1	1.0	2.0	1.0	0.871
rbf-hetero	1			0.1	0.728
rbf-ohe-np	1			0.1	0.728
rbf-ohe-pw	1			0.1	0.728

Table 4.2: Comparison of the pairwise kernels, NumPy kernels after OHE, and the heterogeneous kernel.

4.4 Grid Search & Cross Validation

In each dataset from Tab. 3.1, I provided a grid search with 5-fold cross-validation. First of all, I divided each dataset into train/test and validation subsets, where the validation subset contains 15% of the data. The grid search and cross-validation was done on the train/test set, while the true results of the model can be found after predicting the validation set.

4.4.1 SVM Classification

In Tab. 4.3a, 4.3b I show the results on the validation set, compared to the accuracy gathered from openML rankings (Tab. 3.1). In Tab. C.4 I show results from the test folds subsets (average ± std. dev.). Similar to this is Tab. C.3, where I show results from the train subsets of the folds.

AUC-ROC metric was used as "refit" parameter for the grid search. This means that the model, which had the best AUC-ROC from all other candidates was chosen to classify data points from the validation set.

File Name	C	d	a	γ	Kappa	AUC	Acc	Ac*
bands	10			2.15E-02	0.68	0.91	0.84	0.86
cmc	10			2.15E-02	0.40	0.76	0.71	0.56
credit-g	10	1	10		0.42	0.81	0.77	0.78
crx	1			2.15E-02	0.70	0.93	0.85	0.87
diagnosis	1	1	1		1.00	1.00	1.00	1.00
heart	1	1	1		0.80	0.96	0.90	0.81
hepatitis	1	2	1		0.41	0.83	0.79	0.79
horse-colic	100			4.64E-04	0.49	0.80	0.77	0.82
ilpd	1	1	10		0.00	0.73	0.67	0.72
irish	1	1	1		1.00	1.00	1.00	1.00
labor	1	2	1		1.00	1.00	1.00	0.94
post-operative	1	1	1		0.00	0.75	0.71	0.48
profb	1000				0.40	0.78	0.73	0.76

(a) Classification results after hyperparameter tuning and cross-validation. SVM using heterogeneous kernel.

File Name	C	d	a	γ	Kappa	AUC	Acc	Ac*
bands	464.2			3.59E-05	0.58	0.85	0.80	0.86
cmc	1	1	2		0.41	0.70	0.72	0.56
credit-g	10			2.15E-02	0.29	0.66	0.76	0.78
crx	1000				0.71	0.92	0.86	0.87
diagnosis	1	1	1		1.00	1.00	1.00	1.00
heart	1	1	1		0.75	0.95	0.88	0.81
hepatitis	1			4.64E-04	0.00	0.93	0.75	0.79
horse-colic	1	1	1		0.70	0.90	0.88	0.82
ilpd	1	10	1		0.00	0.67	0.80	0.72
irish	1	1	2		0.97	0.97	0.99	1.00
labor	0.1778				0.57	1.00	0.78	0.94
post-operative	1	1	2		0.05	0.63	0.57	0.48
profb	3.162				0.00	0.59	0.62	0.76

(b) Classification results after hyperparameter tuning and cross-validation. SVM using default kernels, target encoded categorical features. Parameter m for additive smoothing is always 50.

Table 4.3: Grid search and cross-validation results using (a) heterogeneous kernels and (b) other kernels on target encoded categorical features.

In Tab. 4.3 C is the parameter for errors of the SVM; a, d are parameters (coef0, degree) of the polynomial kernel; and γ is the parameter for RBF kernel. If only C is filled in a row, this means the linear kernel was used. If also a, d were filled, this stands for polynomial kernel. Both C and γ parameters mean the RBF kernel was used. The accuracy of the model on the validation set is compared to the one gathered from OpenML rankings, as shown in the Ac* column.

For each kernel, I used different C, for computational efficiency (no cluster for these computations was considered). I provided the same parameters for heterogeneous kernels and basic kernels after target encoding. Parameters for each kernel:

- linear: $C \in \{0.01, 0.17, 3.16, 56.23, 1000\}$
- polynomial: $C, a \in \{1, 10\}, d \in \{1, 2\}$
- rbf: $C \in \{1, 10, 100, 1000\}, \gamma \in \{1 \cdot 10^{-5}, 4.64 \cdot 10^{-4}, 2.15 \cdot 10^{-2}, 1\}$

I looked for target encoding's smoothing hyperparameter m with values $\{50, 100, 300, 500\}$ but found out the best models chosen after grid search tend to be "overfitted" as explained in Sec. 2.3. Therefore the smallest value 50 was chosen as the best value for m and is set throughout the measurements shown in Tab 4.3b.

Datasets "adult" and "kobe" were not included, because they were too large to find the optimal hyperparameters by grid search defined above. My machine is not that powerful to compute the gram matrix in reasonable time, so I just provide one-fit predictions in comparison tables (4.1, C.1, C.2).

4.4.2 Kernel Ridge Regression

I also provided grid search and cross-validation on "servo" dataset. The parameters for the grid search are: $\lambda \in$ `np.logspace(-4, 4, 30)` [1], parameters for polynomial kernel: $a \in \{0, 5, 10, 15, 20, 25, 30, 35\}$, $d \in \{1, 2, 3\}$, parameters for RBF kernel: $\gamma \in$ `np.logspace(-4, 1, 30)`. For best parameter m for target encoding I searched through $\{50, 300, 500, 700, 1000\}$. The best value was again 50.

I was not able to obtain rankings to this dataset from OpenML, because there is no rank for this dataset. Therefore there is no "true" R2 value to compare with.

[1] https://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html

Model	Method	R2	λ	a	d	γ	m
Kernel Ridge	Hetero Kernel	0.891	0.0303	35	3		
Kernel Ridge	Target encoding	0.870	0.0045	35	3		50

Table 4.4: Results of grid search and cross-validation on regression task.

4.5 Memory Consumption

I used `memory_profiler` package [26], which provided results in Fig. 4.1. I used IPython's magic function `%memit` from this package. I specified the datatype of the output gram matrix, same as a parameter in NumPy's sum function. I compared NumPy's dot after pandas `get_dummies` on two various sized, randomly generated categorical data (input matrices). Each measurement was done in a separate python environment: after each run, I started a new interpreter measuring different functions, and the results were put into the graph, as shown in Fig. 4.1. Here, "inc" means the amount of memory increased during measurement.

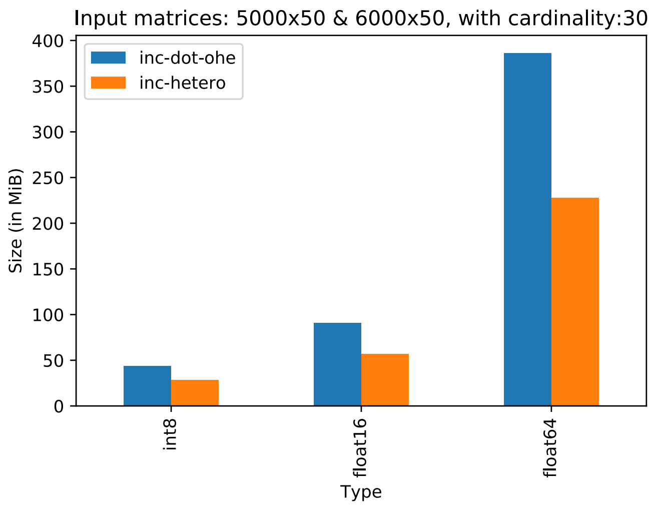


Figure 4.1: Memory comparison of NumPy's dot & OHE, and heterogeneous kernel on various input data types.

4.6 Runtime On Random Data

In this section I provide a runtime comparison on random generated, categorical data. In Fig. 4.2, I compare naive heterogeneous kernel, dot product & OHE, and heterogeneous kernel. Each runtime measurement is done at least 30 times, then the average of all the measurements (for the same parameters) is plotted on the graph. All measurements were done on a machine with specifications shown in Lis. 6.

In Fig. 4.2 we can see, that this naive implementation would not scale on large datasets (even though it is working correctly). For the naive heterogeneous kernel, it takes much longer time to compute the gram matrix, even for a small number of samples. This graph has a logarithmic y-axis.

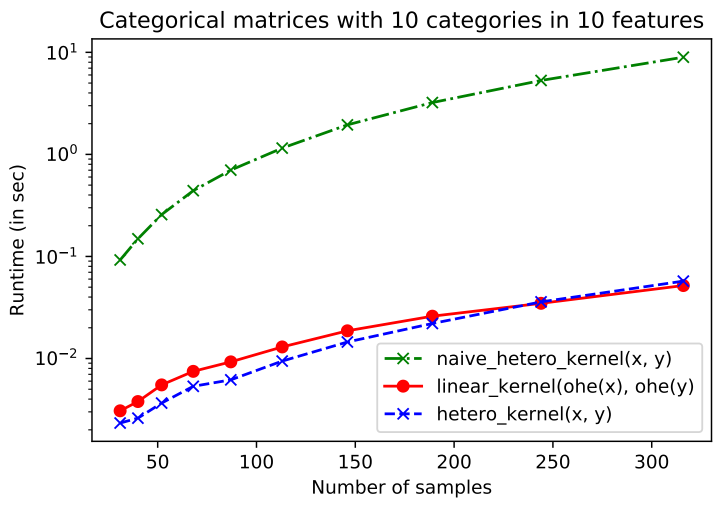


Figure 4.2: Runtime comparison of naive heterogeneous kernel, heterogeneous kernel and NumPy's dot product after OHE on data with a various number of samples.

In Fig. 4.3 we can see that with the rising number of samples, the heterogeneous kernel works faster. The categorical data are of higher cardinality.

In Fig. 4.4 we can see, how the rising number of categorical features also makes computing of dot product after OHE slower than the heterogeneous kernel. Even the cardinality is relatively small.

In Fig. 4.5 we can see, how rising cardinality affects the runtime of dot product and OHE. This is the edge-case measurement, because the number of categorical features is one. We can see that with higher number of samples, the intersection of the two compared methods is moving to smaller number of categories in the feature.

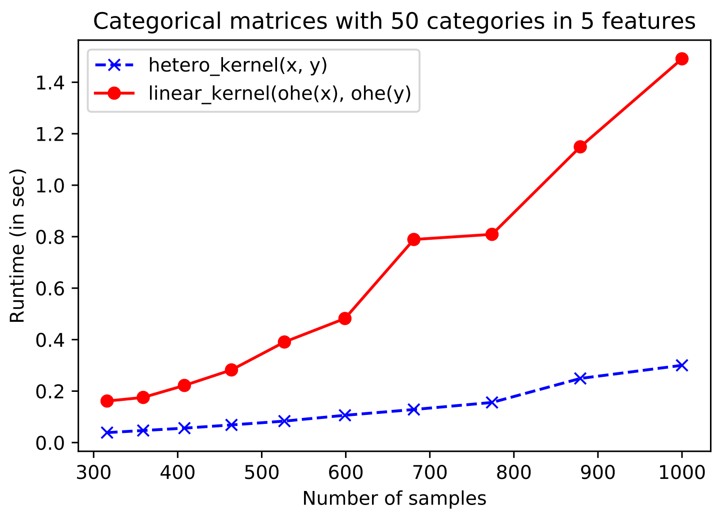


Figure 4.3: Runtime comparison of the heterogeneous kernel and linear kernel (dot product) after OHE on data with a various number of samples.

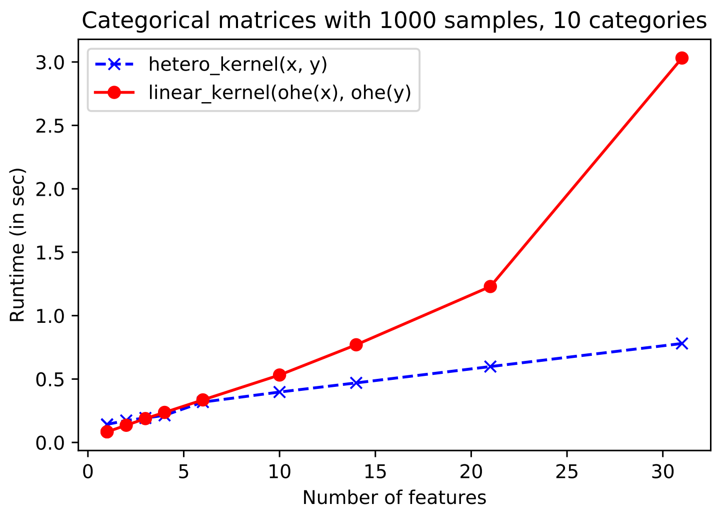


Figure 4.4: Runtime comparison of the heterogeneous kernel and linear kernel (dot product) after OHE on data with a various number of features.

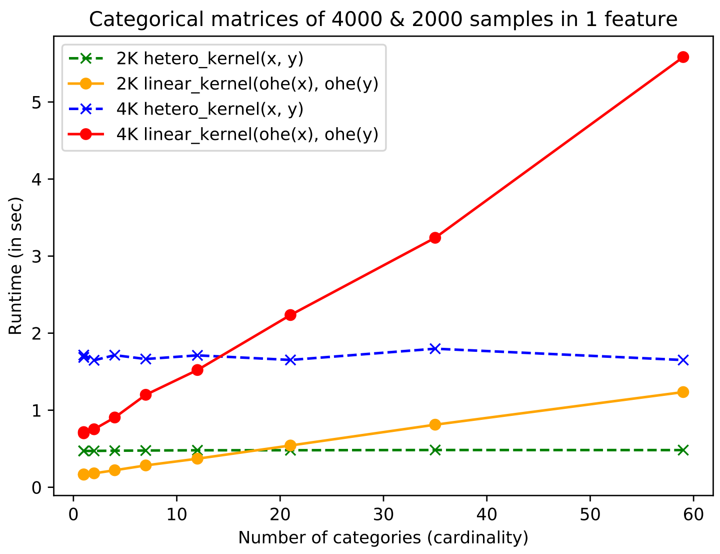


Figure 4.5: Runtime comparison of the heterogeneous kernel and linear kernel (dot product) after OHE on data with various cardinality in each feature.

4.7 Runtime On Real Datasets

In Fig. 4.6, C.2, C.3 I show runtime comparison (with symmetrical logarithmic y-axis) of various approaches to real datasets with mixed data, mentioned in Tab. 3.1. I split each dataset into train and test set (85% and 15%) and trained SVM with various kernel functions.

I fitted each dataset ten times and measured the time. Then I took the average from the measurements and plotted it. OHE & libsvm means I put `kernel='linear'` into the SVM and fitted with train set after OHE. OHE dot means I used NumPy's dot product as a kernel function and fitted with data after OHE. Hetero linear means I used (linear) heterogeneous kernel for the measurements. This description of Fig. 4.6 is similar for Fig. C.2, C.3 (in appendix).

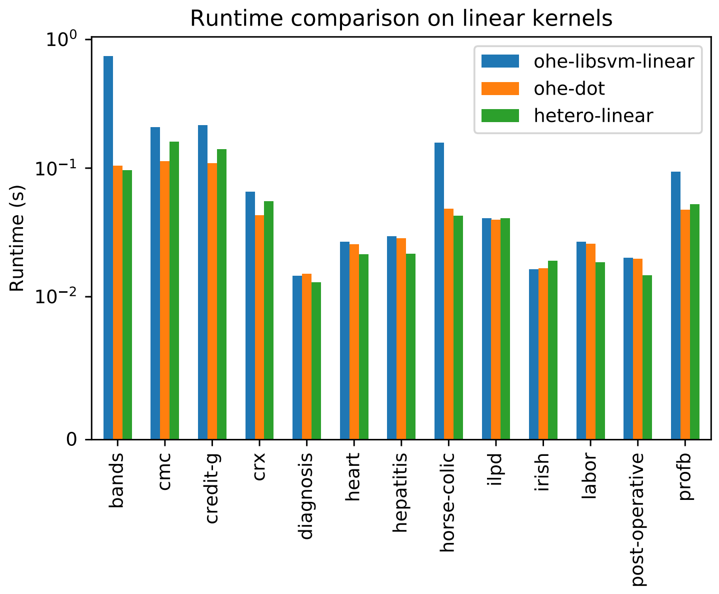


Figure 4.6: Runtime comparison on real datasets, comparing the runtime of linear kernels.

Fig. 4.7 is a comparison of all types of kernels, but on bigger datasets. It is more clear to show the runtime of these datasets separately. Other graphs are more comprehensive and clean.

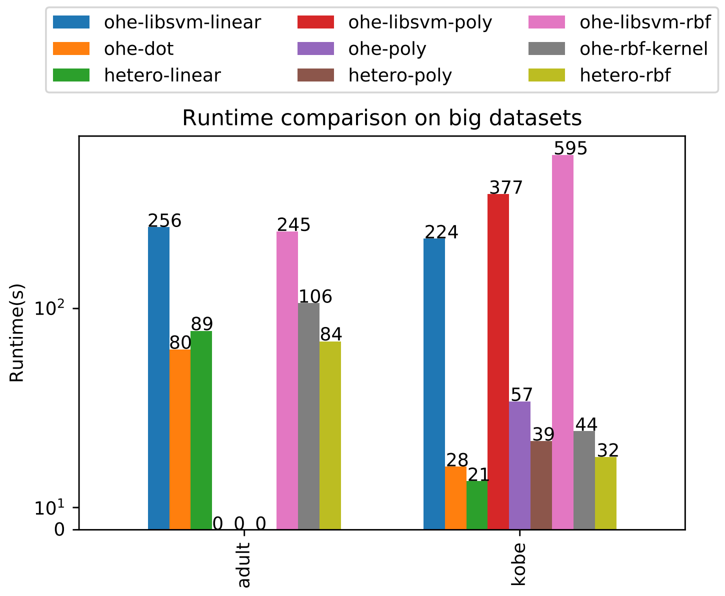


Figure 4.7: Runtime comparison on bigger real datasets.

I needed to interrupt the polynomial kernel run, when computing gram matrix for “adult” dataset. It took almost four hours, and no output for no polynomial kernel was given. Therefore there are missing bars in Fig. 4.7.

4.8 Discussion

Heterogeneous kernel gives the same output gram matrix, as if one-hot-encoding was applied on given data. This equivalence is demonstrated in unit tests, in attached CD. This kernel can be used for classification (e.g., SVM), as we can see in Tab. 4.1, C.1, C.2, and other tasks (e.g. regression with kernel ridge), shown in Tab. 4.2.

As we can see from the results in Tab. 4.3, 4.4 it depends, which approach of treating the categorical features (OHE or target-encoding) is the best. The metrics are sometimes better after target encoding, sometimes worse. So again, it is always good to try each approach before choosing the best model for given task. For some of the datasets the kappa is zero not because it is an error in computation, but because they are difficult datasets. I was not able to train such a model, which would generalize better on these datasets.

The proposed method saves almost double amount of memory, when working on greater amount of categorical features with high cardinality. We can see the increment in Fig. 4.1. The saved amount of memory is significant, if given dataset contains both continuous and categorical features. The data type of gram matrix also affects memory requirements. Particularly whenever we need double typed gram matrix (e.g.: because of double typed numerical features), memory requirements of the proposed method are significantly lower than if we used OHE.

Designed kernel scales well on randomly generated categorical data. With rising number of samples containing a few features of high cardinality, the runtime is almost four times smaller, as we can see in Fig. 4.3. In Fig. 4.4 we can notice almost exponential runtime, when applying OHE for the dot product, while the heterogeneous kernel tends to have linear runtime. This can be seen even on one thousand samples. In general, higher cardinality (30 and more) means better performance, comparing to OHE and dot product. Smaller cardinality can be simply treated with OHE, as we can see in Fig. 4.5, where OHE dot kernel tends to be linear with rising cardinality, while heterogeneous kernel stays in constant time. The heterogeneous kernel is invariant to cardinality in terms of runtime. If we are given a dataset containing at least one feature with minimum cardinality 50, we can expect savings on runtime. This saving is almost 5 times less than OHE on 1000 samples, as seen in Fig. 4.3.

In runtime measure on the real datasets, we can see that in some cases, classical OHE and linear kernel outperform the heterogeneous kernel in runtime. It can be caused by the fact, that the heterogeneous kernel is designed to work mainly with mixed data containing high cardinality categorical features. In some datasets, there are only a few categories or a small number of categorical features. This fact can be seen in Fig. 4.5 on the measurements with small amount of categories or in Fig. 4.4 on measurements with small amount of categorical features. For example, heterogeneous kernel applied on cmc, credit-g, and crx datasets are acting worse in terms of runtime (see Fig. 4.6). If we look on Tab. 3.2, we can see, that the sum of categories throughout these datasets is not very high. Therefore I assume, this approach is not suitable for these types of datasets. On “kobe” dataset, containing high cardinality features, the heterogeneous kernel saved almost ten seconds of computation, see Fig. 4.7. What is surprising, that libsvm is not acting well in comparison to either NumPy’s approach, or heterogeneous kernel. It may be caused by better memory handling of NumPy, which may use BLAS[2] packages. It provides subroutines and subprograms written in Fortran or C, working differently on different machines/distributions/processors/etc. [27]

If using this kernel in practice, we can check if given dataset contains at least one categorical feature with cardinality at least 30. If so, we better use the heterogeneous kernel, if not, OHE is sufficient. Pandas DataFrame can tell us very fast, how many unique values are in each feature, therefore this “test” is very fast.

[2]`https://docs.scipy.org/doc/scipy/reference/linalg.blas.html`

Conclusion

In this thesis, I introduced kernel models used for pattern recognition, such as *support vector machine* and *kernel ridge regression*. Various kernels commonly used were introduced. I described preprocessing methods such as scaling, one-hot-encoding, and target encoding.

Then I described in detail the whole process of creating the heterogeneous kernel, its implementation, and usability not only as a linear kernel but also polynomial and RBF.

Afterward, I provided evidence that the designed kernel can mimic one-hot-encoding and dot product, the same as Euclidean distances on categorical data. Heterogeneous kernel works the same is if you first provide OHE and then apply dot kernel or Euclidean distances (for RBF). We saw that target encoding is not always suitable when looking for the best model; we need to try each approach to choose the best. I also showed that this kernel is suitable not only for classification (SVM) but also for regression (Kernel Ridge).

The memory consumption is decreased, when there is a need for a kernel working with mixed data. We saw how the memory increased almost two times when working with high cardinality features. This kernel can decrease the runtime while working with high cardinality features. This can be significant if the dataset is larger. The heterogeneous kernel was able to cut down almost ten seconds of computation on larger dataset .

The heterogeneous kernel is prepared for mixed data (categorical & numerical). If the cardinality of a categorical feature is higher, this kernel stays invariant to the cardinality, in terms of runtime. It can work with missing values in categorical features because it treats it as another categorical value (similar to OHE).

Bibliography

[1] Corinna, C.; Vladimir, V. Support-Vector Networks. *Machine Learning*, volume 20, no. 3, 1995: pp. 273–297, ISSN 08856125, doi:10.1007/BF00994018, arXiv:1011.1669v3. Available from: http://www.ncbi.nlm.nih.gov/pubmed/19549084

[2] John Shawe-Taylor; Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press 2004, 2004, ISBN 9780521813976, 195–211, 289–297 pp.

[3] Halford, M. Target Encoding Done The Right Way. 2018. Available from: https://maxhalford.github.io/blog/target-encoding-done-the-right-way/

[4] Pedregosa Fabian; Michel, V.; et al. Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830, ISSN 15324435, doi:10.1007/s13398-014-0173-7.2, 1201.0490. Available from: http://scikit-learn.sourceforge.net.

[5] Villegas García, M. A. *An investigation into new kernels for categorical variables*. Dissertation thesis, Universitat Politecnica de Catalunya, 2013. Available from: http://upcommons.upc.edu/pfc/handle/2099.1/17172

[6] An, S.; Liu, W.; et al. Face Recognition Using Kernel Ridge Regression. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, jun 2007, ISSN 1063-6919, pp. 1–7, doi:10.1109/CVPR.2007.383105.

[7] Kennard, R. W.; Hoerl, A. E. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, volume 12, no. 1, 1970: pp. 55–67. Available from:

http://internet.math.arizona.edu/hzhang/math574m/Read/RidgeRegressionBiasedEstimationForNonorthogonalProblems.pdf

[8] Vašata, D. MI-ADM lecture 6 handout. 2019. Available from: https://courses.fit.cvut.cz/MI-ADM/lectures/files/MI-ADM-06-en-handout.pdf

[9] Hofmann, T.; Schölkopf, B.; et al. Kernel methods in machine learning. *Annals of Statistics*, volume 36, no. 3, 2008: pp. 1171–1220, ISSN 00905364, doi:10.1214/009053607000000677.

[10] Vašata, D. MI-ADM lecture 8 handout. 2019. Available from: https://courses.fit.cvut.cz/MI-ADM/lectures/files/MI-ADM-08-en-handout.pdf

[11] Olšák, P. Lineární algebra, 2007. Available from: http://petr.olsak.net/ftp/olsak/linal/linal.pdf

[12] Genton, M. G.; Cristianini, N.; et al. Classes of kernels for machine learning: a statistics perspective. *Journal of Machine Learning Research*, volume 2, no. 2, 2001: pp. 299–312. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.7887

[13] Dorogush, A. V.; Ershov, V.; et al. CatBoost: gradient boosting with categorical features support. 2018: pp. 1–7, 1810.11363. Available from: http://arxiv.org/abs/1810.11363

[14] Micci-Barreca, D. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, volume 3, no. 1, 2007: p. 27, ISSN 19310145, doi:10.1145/507533.507538.

[15] Jukes, E. *Encyclopedia of Machine Learning and Data Mining (2nd edition)*, volume 32. New York, NY, USA: Springer Science+Business Media, 2018, ISBN 9781489976857, 3–4 pp., doi:10.1007/978-1-4899-7687-1, 0005074v1.

[16] Garavaglia Susan, S. A. A Smart Guide To Dummy Variables. 2016. Available from: https://stats.idre.ucla.edu/wp-content/uploads/2016/02/p046.pdf

[17] Duff, I. S.; Erisman, A. M.; et al. *Direct methods for sparse matrices*. Oxford University Press, 2017.

[18] McKinney, W. Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference. *Proceedings of the 9th Python in Science Conference*, volume 1697900, 2010: pp.

50–56. Available from: http://conference.scipy.org/proceedings/scipy2010/pdfs/proceedings.pdf

[19] Asaithambi, S. Why, How and When to Scale your Features. 2017. Available from: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

[20] Rashka, S. About Feature Scaling and Normalization. 2014. Available from: https://sebastianraschka.com/Articles/2014{_}about{_}feature{_}scaling.html

[21] Oliphant, T. NumPy: A guide to NumPy. USA: Trelgol Publishing. Available from: http://www.numpy.org/

[22] Dua, D.; Graff, C. UCI Machine Learning Repository. 2017. Available from: http://archive.ics.uci.edu/ml

[23] Vanschoren, J.; van Rijn, J. N.; et al. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, volume 15, no. 2, 2013: pp. 49–60, doi:10.1145/2641190.2641198. Available from: http://doi.acm.org/10.1145/2641190.2641198

[24] Chih-Wei Hsu Chih-Jen Lin. A Comparison of Methods for Multi-class Support Vector Machines. *IEEE Transactions on Neural Networks*, 2002: pp. 1–11, doi:10.1109/72.991427. Available from: https://www.csie.ntu.edu.tw/{~}cjlin/papers/multisvm.pdf

[25] Nagelkerke, N. J. D. A Note on a General Definition of the Coefficient of Determination Miscellanea A note on a general definition of the coefficient of determination. *Biometrika*, volume 78, no. 3, 2008: pp. 691–692, ISSN 00063444, doi:10.1093/biomet/78.3.691.

[26] Pedregosa, F.; Gervais, P. Memory Profiler, Python's package index. 2018. Available from: https://pypi.org/project/memory-profiler/

[27] Beuckelmann, M. Boosting numpy: Why BLAS Matters. 2017. Available from: https://markus-beuckelmann.de/blog/boosting-numpy-blas.html

[28] Chih-Chung, C.; Chih-Jen, L. LIBSVM – A Library for Support Vector Machines. 2018. Available from: https://www.csie.ntu.edu.tw/{~}cjlin/libsvm/

Appendix A
Acronyms

SVM	Support Vector Machine
RBF	Radial Basis Function
OHE	One-Hot-Encoding
AUC	Area Under Curve
ROC	Receiver Operating Characteristics
np	NumPy [21]
libsvm	A Library for Support Vector Machines [28]

Appendix B

Notation

In this chapter, I want to introduce notation for every kind of variables, vectors, or statements, to preserve consistency throughout this thesis.

In general, matrices are bold uppercase (e.g., $\mathbf{A}, \mathbf{B}$), vectors are bold (mathematical) lowercase (e.g., $\boldsymbol{x}, \boldsymbol{w}$), and scalars are lowercase Greek letters or cursive Latin letters. Sets are denoted as uppercase italic letters (e.g., X, S). Number sets are denoted as double bold. E.g., $\mathbb{R}$ means a set of real numbers.

When writing numbers, I use a dot (.) as decimal separator. If the number is too small, I use E to denote powers of 10 (e.g. 1.23E-4 $= 1.23 \times 10^{-4}$).

Dot product is denoted as $\langle \cdot, \cdot \rangle$, or as matrix multiplication (e.g. $\boldsymbol{x}^T\boldsymbol{y}$), where $\boldsymbol{x}^T$ means transposition of $\boldsymbol{x}$.

Appendix **C**

Details Of Evaluation

Each row in Tab. 4.1, C.1, C.2 contains tuples with three values: accuracy, Cohen's kappa, and AUC-ROC (Sec. 4.4). OHE & NumPy is NumPy [21] dot product. OHE & libsvm is when "linear" is passed as a kernel into constructor of SVM, therefore libsvm is responsible for the kernel matrix. Similar in other tables (polynomial and RBF).

For polynomial kernel the task of prediction on "adult" dataset took too long, so I interrupted it. This is why there is missing row in Tab. C.1. It is explained as well in Sec. 4.7.

Dataset Name	OHE & NumPy	OHE & libsvm	Heterogeneous
adult			
bands	(0.81, 0.61, 0.9)	(0.81, 0.61, 0.9)	(0.81, 0.61, 0.9)
cmc	(0.62, 0.21, 0.61)	(0.62, 0.21, 0.61)	(0.62, 0.21, 0.61)
credit-g	(0.75, 0.35, 0.7)	(0.75, 0.35, 0.7)	(0.75, 0.35, 0.7)
crx	(0.79, 0.58, 0.84)	(0.79, 0.58, 0.84)	(0.79, 0.58, 0.84)
diagnosis	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
heart	(0.72, 0.44, 0.8)	(0.72, 0.44, 0.8)	(0.72, 0.44, 0.8)
hepatitis	(0.82, 0.54, 0.89)	(0.82, 0.54, 0.89)	(0.82, 0.54, 0.89)
horse-colic	(0.86, 0.67, 0.9)	(0.86, 0.67, 0.9)	(0.86, 0.67, 0.9)
ilpd	(0.66, 0.11, 0.68)	(0.66, 0.11, 0.68)	(0.66, 0.11, 0.68)
irish	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
kobe	(0.99, 0.95, 1.0)	(0.99, 0.95, 1.0)	(0.99, 0.95, 1.0)
labor	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
post-operative	(0.7, -0.14, 0.52)	(0.7, -0.14, 0.52)	(0.7, -0.14, 0.52)
profb	(0.65, 0.25, 0.67)	(0.65, 0.25, 0.67)	(0.65, 0.25, 0.67)

Table C.1: Comparison of prediction metrics, using SVM with various polynomial kernels.

Dataset Name	OHE & NumPy	OHE & libsvm	Heterogeneous
adult	(0.85, 0.57, 0.9)	(0.85, 0.57, 0.9)	(0.85, 0.57, 0.9)
bands	(0.76, 0.49, 0.88)	(0.76, 0.49, 0.88)	(0.76, 0.49, 0.88)
cmc	(0.69, 0.36, 0.72)	(0.69, 0.36, 0.72)	(0.69, 0.36, 0.72)
credit-g	(0.78, 0.36, 0.79)	(0.78, 0.36, 0.79)	(0.78, 0.36, 0.79)
crx	(0.83, 0.67, 0.91)	(0.83, 0.67, 0.91)	(0.83, 0.67, 0.91)
diagnosis	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
heart	(0.84, 0.67, 0.9)	(0.84, 0.67, 0.9)	(0.84, 0.67, 0.9)
hepatitis	(0.82, 0.48, 0.9)	(0.82, 0.48, 0.9)	(0.82, 0.48, 0.9)
horse-colic	(0.88, 0.71, 0.9)	(0.88, 0.71, 0.9)	(0.88, 0.71, 0.9)
ilpd	(0.79, 0.0, 0.74)	(0.79, 0.0, 0.74)	(0.79, 0.0, 0.74)
irish	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
kobe	(0.99, 0.95, 1.0)	(0.99, 0.95, 1.0)	(0.99, 0.95, 1.0)
labor	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)
post-operative	(0.91, 0.0, 0.52)	(0.91, 0.0, 0.52)	(0.91, 0.0, 0.52)
profb	(0.67, 0.15, 0.75)	(0.67, 0.15, 0.75)	(0.67, 0.15, 0.75)

Table C.2: Comparison of prediction metrics, using SVM with various RBF kernels.

C.1 Details Of Grid Search

In Tab. C.3, C.4 I provide details of cross validation and grid search using SVM with parameters described in Sec. 4.4

File Name	Accuracy	Kappa	AUC-ROC
bands	0.91 ± 0.00	0.79 ± 0.01	0.96 ± 0.00
cmc	0.73 ± 0.01	0.42 ± 0.01	0.79 ± 0.01
credit-g	0.85 ± 0.01	0.56 ± 0.02	0.89 ± 0.01
crx	0.88 ± 0.00	0.75 ± 0.01	0.96 ± 0.00
diagnosis	0.94 ± 0.00	0.85 ± 0.00	1.00 ± 0.00
heart	0.88 ± 0.01	0.74 ± 0.01	0.95 ± 0.00
hepatitis	0.94 ± 0.01	0.75 ± 0.03	0.97 ± 0.01
horse-colic	0.92 ± 0.00	0.81 ± 0.01	0.97 ± 0.00
ilpd	0.77 ± 0.00	0.20 ± 0.01	0.80 ± 0.01
irish	0.94 ± 0.00	0.87 ± 0.00	0.99 ± 0.00
labor	0.92 ± 0.01	0.79 ± 0.01	1.00 ± 0.00
post-operative	0.81 ± 0.02	0.31 ± 0.06	0.80 ± 0.03
profb	0.83 ± 0.00	0.55 ± 0.01	0.90 ± 0.01

Table C.3: Train folds results. Each column stands for the mean ± standard deviation of the given metric on all train folds.

File Name	Accuracy	Kappa	AUC-ROC
bands	0.73 ± 0.02	0.39 ± 0.05	0.83 ± 0.02
cmc	0.66 ± 0.02	0.26 ± 0.04	0.70 ± 0.03
credit-g	0.72 ± 0.02	0.21 ± 0.05	0.74 ± 0.04
crx	0.78 ± 0.02	0.55 ± 0.05	0.89 ± 0.02
diagnosis	0.94 ± 0.01	0.85 ± 0.01	1.00 ± 0.00
heart	0.75 ± 0.04	0.46 ± 0.08	0.86 ± 0.03
hepatitis	0.81 ± 0.06	0.28 ± 0.18	0.81 ± 0.09
horse-colic	0.78 ± 0.03	0.47 ± 0.07	0.86 ± 0.04
ilpd	0.72 ± 0.01	0.06 ± 0.03	0.70 ± 0.04
irish	0.93 ± 0.01	0.86 ± 0.02	0.99 ± 0.01
labor	0.84 ± 0.06	0.56 ± 0.12	0.99 ± 0.02
post-operative	0.70 ± 0.04	-0.02 ± 0.08	0.49 ± 0.13
profb	0.69 ± 0.02	0.19 ± 0.04	0.71 ± 0.03

Table C.4: Test fold results. Each column stands for the mean ± standard deviation of the given metric on test folds.

C.2 Runtime

In this section, I provide more runtime measurements. Fig. C.1 is an extension to Fig. 4.5, where only one categorical feature was provided. Fig. C.2, C.3 show comparison of heterogeneous kernel working on real datasets, but as polynomial and RBF kernel.

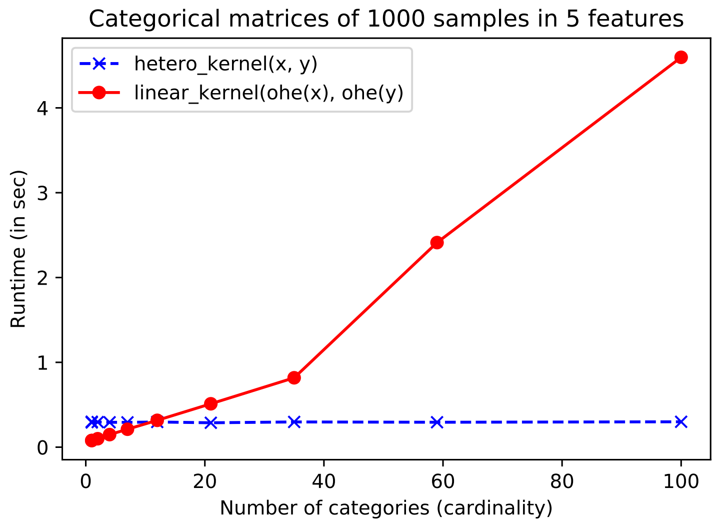


Figure C.1: Runtime comparison of the heterogeneous kernel and linear kernel (dot product) after OHE on data with higher cardinality in each feature.

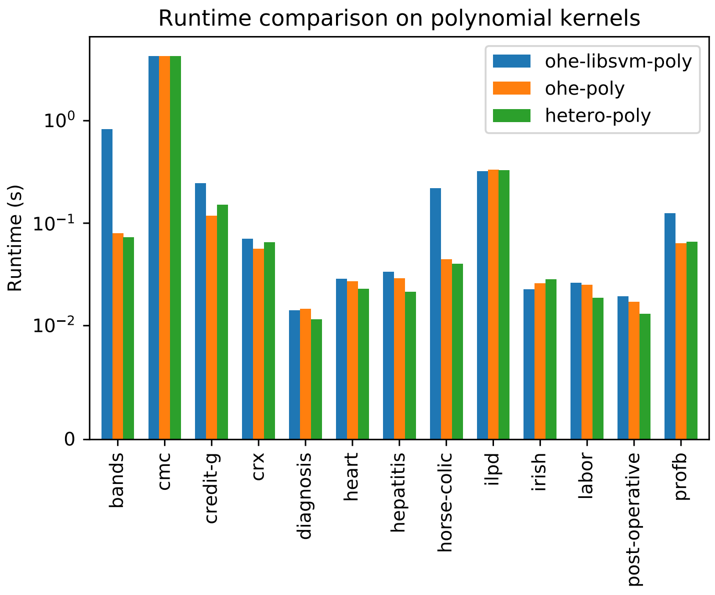


Figure C.2: Runtime comparison on real datasets, comparing the runtime of polynomial kernels.

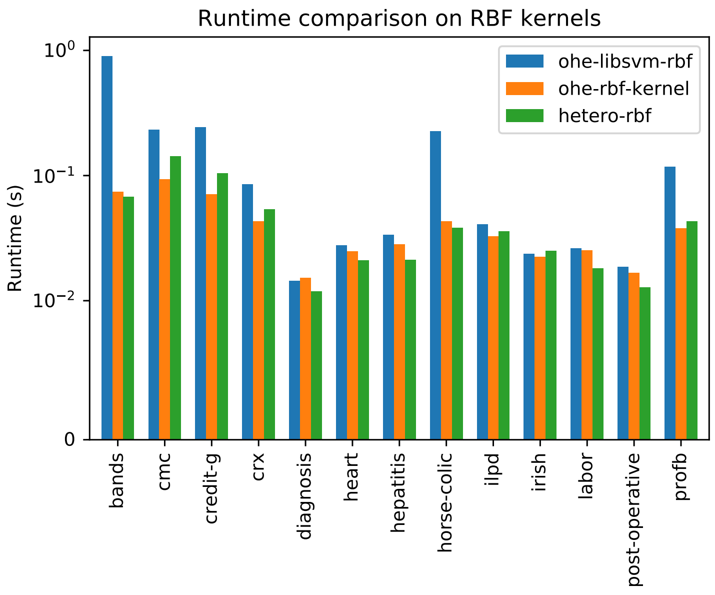


Figure C.3: Runtime comparison on real datasets, comparing the runtime of RBF kernels

C.3 Specifications

The hardware and software specifications of the machine, where time and memory measurements were done. Shown in Lis. 6.

```
Hardware Overview:
    Model Name: MacBook Pro
    Model Identifier: MacBookPro14,1
    Processor Name: Intel Core i7
    Processor Speed: 2,5 GHz
    Number of Processors: 1
    Total Number of Cores: 2
    L2 Cache (per Core): 256 KB
    L3 Cache: 4 MB
    Memory: 16 GB
    Boot ROM Version: 190.0.0.0.0
    SMC Version (system): 2.43f6
    Serial Number (system): C02V70MHJ9K9
    Hardware UUID: 0BEF40BF-97E7-5089-8C02-BBB835651879
```

Listing 6: Hardware specifications of the machine where all measurements were done.

Appendix **D**

Contents of enclosed CD

```
  README.md ............................. file with CD contents description
├── IMPLEMENTATION ...................... directory with the implementation
│   ├── src ............................. directory of source codes
│   ├── data ............................ directory of used datsets
│   ├── notebooks ....................... directory of Jupyter Notebooks
│   └── requirements.txt ...... dependencies for Python 3.6 implementation
└── THESIS ..................... directory of LaTeX source codes of the thesis
    ├── BP_Fabo_Samuel_2018.pdf ................ thesis text in PDF format
    ├── chapters ................... directory with .tex files – chapters
    ├── images ................................ directory with .pdf images
    ├── tables .................................. directory with .csv tables
    └── BACHELOR.bib ..................... BibTex file with bibliography
```