



Ulm University | 89069 Ulm | Germany

**Faculty of
Engineering, Computer Science
and Psychology**
Neural Information Processing

Approximating Kernels with Random Fourier Features

Master Project at Ulm University

Submitted by:

Mathis Rost , Tanja Zast

mathis.rost@uni-ulm.de , tanja.zast@uni-ulm.de

Reviewer:

Prof. Dr. Friedhelm Schwenker

Supervisor:

Sebastian Gottwald

2023

Version from July 22, 2023

Abstract

Random Fourier Features are a well known technique for approximating kernel methods. Particularly for large and high dimensional datasets, they can be used as a faster alternative to traditional kernel methods. This work aims to provide a comprehensive theoretical background on Random Fourier Features. Additionally, we will evaluate the performance and computation time of the approximation method and compare it with other dimensionality reduction methods. We will address both regression problems and classification problems. Lastly, we will combine individual Support Vector Machines trained with the approximation of Random Fourier Features into a combined classifier and evaluate its performance. These evaluations yield interesting and promising results.

Contents

1. Introduction	1
2. Used Methods	3
2.1. Principal Component Analysis	3
2.1.1. Singular Value Decomposition	5
2.2. Random Projection	6
2.3. Support Vector Machine	8
2.4. Ensemble Learning	10
2.4.1. Wisdom of the Crowd	10
3. Kernel Machines	12
3.1. Feature Space and Kernels	13
3.2. Kernel Trick	14
3.2.1. Radial Bias Function Kernel	15
3.2.2. Scaling of Kernel Machines	16
4. Random Fourier Features	19
4.1. Random Features	19
4.1.1. Approximating Gaussian Kernel	19
4.2. Generating Random Fourier Features	23
4.2.1. Sampling Random Fourier Features	23
4.2.2. Nyström Method	24
5. Experiments and Results	26
5.1. Experimental Setup	26
5.1.1. Datasets	27
5.1.2. Hyperparameter	28
5.2. Approximating Kernel Regression	28
5.3. Approximating Kernel of Kernel SVM	29
5.3.1. Comparing Random Fourier Features with Nyström Features	30

Contents

5.3.2. Dimensionality Reduction with PCA	31
5.3.3. Noisy Data	32
5.4. Ensemble Learning	34
6. Discussion	36
6.1. Evaluating the Results	36
6.1.1. Approximating Kernel Regression	36
6.1.2. Approximating Kernel SVM	37
6.1.3. Evaluation of Ensemble Learning	38
6.2. Conclusion	39
6.3. Future Work	39
A. Appendix	42
A.1. Plots for Fashion-MNIST	42
A.2. Ensemble Learning	45

1

Introduction

In recent years, kernel methods have emerged as powerful techniques for various machine learning tasks, including classification, regression, and dimensionality reduction. These methods leverage the notion of a kernel function to implicitly map input data into high-dimensional feature spaces, where linear models can effectively operate. However, their computational complexity increases significantly with the size of the dataset, limiting their applicability to large-scale problems [19].

To address this limitation, researchers have proposed a promising technique known as Random Fourier Features (RFFs) [16]. RFFs provides an efficient approximation of the kernel method by approximating the kernel function using random feature mappings. By leveraging the Fast Fourier Transform (FFT), RFF enables a significant reduction in computational complexity, making kernel methods more scalable and accessible for large datasets.

In this paper, we present a theoretical analysis behind the theory of common dimensionality reduction methods. We also provide insights into the theory of Reproducing Kernel Hilbert space (RKHS) and kernel methods. Furthermore, we derive the theory behind random features, which in total, gives us the theoretical insides for our experiments we want to perform.

We also present a comprehensive experimental analysis that investigates the computational efficiency of RFFs compared to the traditional kernel method. Our primary focus is to assess the computing time required by both approaches when applied to various machine learning tasks, including classification, regression, and dimensionality reduction. By quantifying the computational benefits of RFFs, we aim to provide empirical evidence of its effectiveness in accelerating kernel methods.

1. Introduction

Furthermore, we delve into the realm of ensemble learning and explore how RFFs can enhance the performance of ensemble models. Ensembles are known for their ability to improve predictive accuracy by combining multiple models, and we investigate how incorporating RFF based models into ensemble frameworks can yield better results. By training different models using features based on RFF, we aim to assess the impact of RFFs on ensemble learning, providing insights into its potential for boosting ensemble performance. This is a proposed continuation of the work of [18].

To conduct our experiments, we employ benchmark datasets from various domains to assess the performance and computational efficiency of RFFs compared to the traditional kernel method or other dimensionality reduction methods which save computing time. Our findings shed light on the trade-offs between accuracy, computing time, and scalability, ultimately helping practitioners make informed decisions when choosing between these two approaches.

In summary, this paper contributes to the research on RFFs by providing an in-depth analysis of its computational benefits compared to the traditional kernel method and other dimensionality reduction methods. Additionally, we explore the potential of RFFs in enhancing ensemble learning. Our experimental results and insights aim to guide researchers and practitioners in effectively utilizing RFFs for large-scale machine learning tasks, improving both efficiency and performance.

2

Used Methods

2.1. Principal Component Analysis

The Principal Component Analysis (PCA) is a widely used linear dimensionality reduction technique that aims to find orthogonal axes (principal components) that capture the maximum variance in the data. It projects the data onto these components, allowing dimensionality reduction while preserving the most important information.

Consider a given dataset, denoted as $\{x_1, \dots, x_n\} = X \subset \mathbb{R}^d$. To maximize the variance we now have to find our first axis w_1 with,

$$w_1 = \operatorname{argmax}_{\|w\|=1} \sum_{x \in X} (xw)^2 = \operatorname{argmax}_{\|w\|=1} \|Xw\|^2 \quad (2.1)$$

We can then rewrite this into

$$w_1 = \operatorname{argmax}_{\|w\|=1} \|Xw\|^2 = \operatorname{argmax}_w \frac{w^T X^T X w}{w^T w} \quad (2.2)$$

by using the fact that $x^T x = \|x\|^2$. With dividing by $w^T w$ we ensure that our vector w is normalized. We now have our first principal component. Now using the fact from linear algebra that $X^T X$ is positive semi-definite [11], we now know that the eigenvector of largest eigenvalue of $X^T X$ maximizes equation 2.1.

We also know that $X^T X$ is symmetric. Therefore the eigenvectors of different eigenvalues are orthogonal to each other. Since our requirement was to find an orthogonal system we can choose w_2 as the eigenvector with respect to the second largest eigen-

2. Used Methods

value maximizes this condition i.e.

$$w_2 = \underset{\|w\|=1, w \perp w_1}{\operatorname{argmax}} \|Xw\|^2 \quad (2.3)$$

By continuing this argumentation, the i -th principal component is then given by the eigenvector of the i -th largest eigenvalue of the covariance matrix $X^T X$ [11].

We than can project our data X into lower dimensional space be keeping principal components with high variance. If our data has dimension d we can take our first $k < d$ principal components and can compute a projection,

$$\hat{X} = X(w_1, \dots, w_k) \in \mathbb{R}^{k \times n} \quad (2.4)$$

With the most possible variance. An example can be seen in figure 2.1.

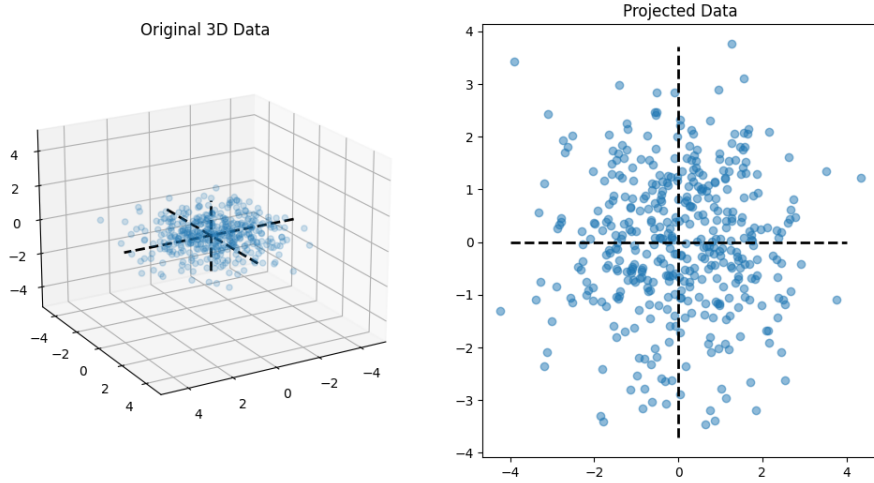


Figure 2.1.: Original data was sampled out of a multivariate gaussian distribution. The black lines indicate the the principal components of the sampled data. By taking the components with highest variance we can project our data in a lower dimensional space.

2. Used Methods

2.1.1. Singular Value Decomposition

Singular Value Decomposition (SVD) is often used as a basis for PCA to reduce the dimensionality of data sets. In this case the dimension of the data-matrix stays the same, the rank however is lowered.

With the prior knowledge that $X^T X$ is symmetric, we can even go further and diagonalize $X^T X$ with help of the spectral theorem of linear algebra [11]. Therefore it exists a $V \in \mathbb{R}^{n \times n}$ with,

$$X^T X = V^T \Lambda V, \Lambda = \text{diag}(\lambda, \dots, \lambda_r, 0, \dots, 0) \quad (2.5)$$

The rank of the matrix is r . Obviously r needs to be smaller or equal than n and d . Note that the columns of V are the eigenvectors of $X^T X$ and therefore it holds that for $i \in \{1, \dots, r\}$

$$X^T X v_i = \lambda_i v_i = \sigma_i^2 v_i$$

Where we define $\sigma_i = \sqrt{\lambda_i}$ as the i -th singular value. For $i \in \{1, \dots, r\}$, we then set

$$U = (u_1, \dots, u_r) \text{ with } u_i = \frac{X v_i}{\sigma_i}$$

We immediately see that u_i is a eigenvector for XX^T with eigenvalue one. Therefore U forms a orthonormal system because XX^T is symmetric. Since $X v_i \in \mathbb{R}^d$ it holds $U \in \mathbb{R}^{r \times d}$. Let Σ be the diagonal matrix of the non-zero singular-values, then we get

$$U = X V \Sigma^{-1} \Leftrightarrow U \Sigma = X V \Leftrightarrow X = U \Sigma V^T$$

As V is a orthonormal matrix and therefore it holds that $V^T V = I$. We now can extend U to a orthonormal basis of \mathbb{R}^d by using Gram-Schmidt [12]. We set $\sigma_{r+1} = \dots = \sigma_d = 0$ and append it on Σ . We then add additional $n - d$ columns to Σ and fill them with zeros. We then get a representation of X with,

$$X = U \Sigma V^T \quad (2.6)$$

2. Used Methods

with $U \in \mathbb{R}^{d \times d}$ orthogonal, $V \in \mathbb{R}^{n \times n}$ orthogonal and $\Sigma \in \mathbb{R}^{d \times n}$. Note that the last $n - r$ rows of V and the last $n - r$ columns of U can be ignored, since they only get multiplied by zero.

What is the gain out of it? Note that the singular values in Σ are related to our principal component from last subsection. We now can sort the singular-values and get a low rank representation out of our matrix or data X . By just taking the first $l < r$ singular-values, first l columns of U and first l rows of V . We then obtain

$$\hat{X} = (u_1, \dots, u_l) \cdot \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_l \end{pmatrix} \cdot (v_1, \dots, v_l)^T \in \mathbb{R}^{d \times n} \quad (2.7)$$

Our matrix X could also be a image. An example of low rank approximation of an image can be seen in figure 2.1.1.

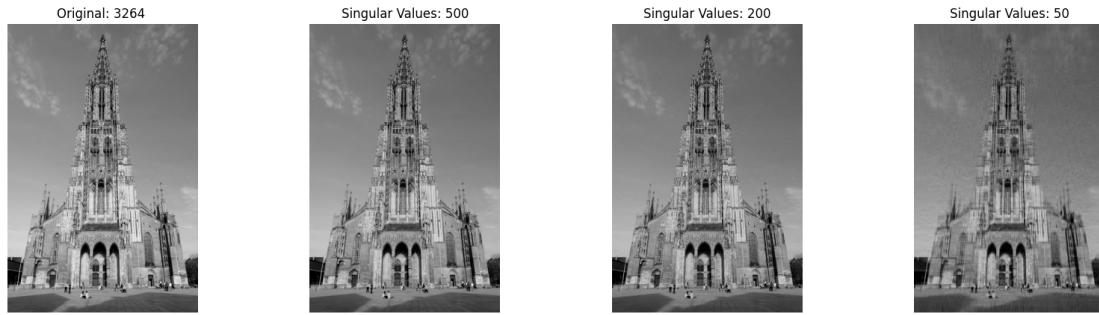


Figure 2.2.: Reconstructed image. On the left side the original image can be seen. The other images where reconstructed with help of the singular value decomposition. For the second image 500, third image 200, fourth image 50 singular-values where used. The image was taken from [2].

2.2. Random Projection

The so far introduced methods are deterministic and will leads to the same result after each computation. However the computation time is very large. Even if there are elegant algorithm for computing the SVD [7] the computation does not scale well.

2. Used Methods

The idea of random projections is, to save computing time with the trade off of not being perfectly accurate. In an optimal case, we want our random projection to approximate a good deterministic projection.

A linear random projection can be seen as a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^p$,

$$f(x) = U^T x = \sum_{i=1}^p u_i^T x \quad (2.8)$$

Where the columns of U with $(u_1, \dots, u_p) \in \mathbb{R}^{d \times p}$ are sampled from a specific distribution with zero mean. One example would be a gaussian distribution. Note that such a random projection matrix U is computationally efficient to sample. A theoretical justification why such a proper U exists is given in [20].

A random projection does not have to be linear. To obtain a non linear random projection. We can take $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^p$ non linear and apply it to our random linear projection, i.e.

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^p, f(x) = \phi(U^T x) \quad (2.9)$$

Where the columns from U are again sampled out of a distribution, which is independent to our data X . A detailed analysis on a specific non linear random projection will be given in the next chapter.

2.3. Support Vector Machine

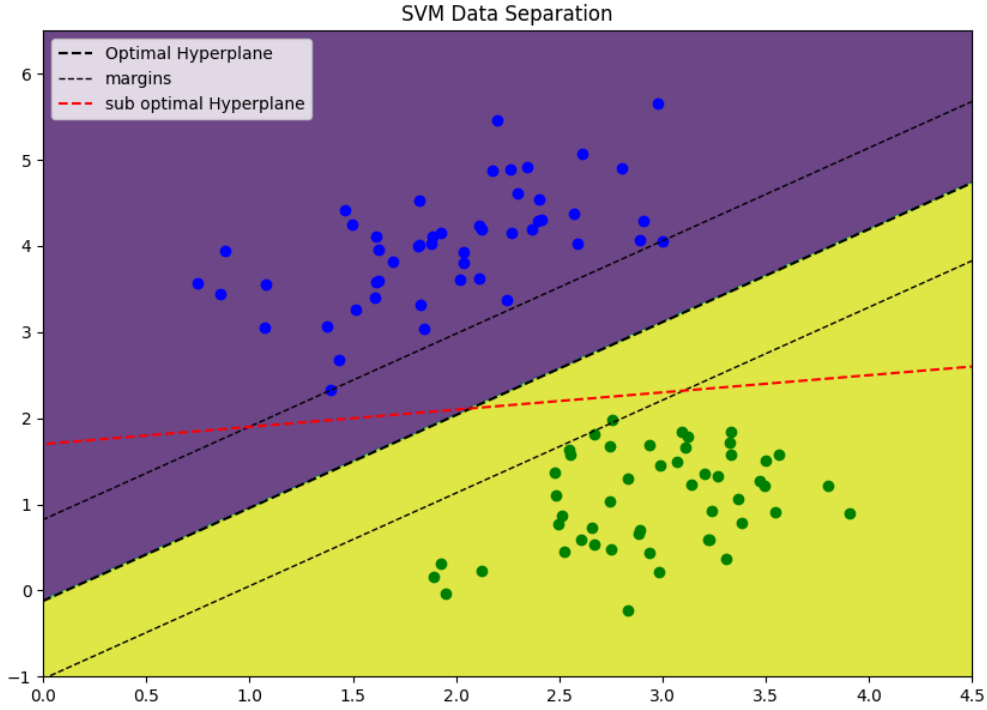


Figure 2.3.: SVM linear dividing the space in to two regions to classify new data. The black line is the best possible line resulting out of a SVM. The red line also separates data. The margins of the red line however would be smaller than the margins of the black line.

Support Vector Machines (SVMs) are used to classify new given data by finding a hyperplane which separates the training set.

In the simple case of a SVM we only have two labels i.e. $Y \in \{0, 1\}^n$ and the space for our training data is given by $X, Y \in \mathbb{R}^n \times \{0, 1\}$. The goal of the SVM is now to find a vector $w \in \mathbb{R}^n$ and a offset $b \in \mathbb{R}^n$ s.t.

$$\text{sign}(w^T x_i + b) = y_i \quad (2.10)$$

Therefore $w^T x + b = 0$ describes a hyperplane in \mathbb{R}^n .

In addition we want the hyperplane as far away as possible of both classes of the data.

2. Used Methods

An illustration can be observed in figure 2.3. More precise we want,

$$(w, b) = \operatorname{argmax}_{(w, b)} \left(\min_{(x, y)} \frac{w^T x + b}{\|w\|} y \right) \quad (2.11)$$

To optimize this is not easy at all, therefore we define our minimum distance as κ . Which means that

$$\kappa \leq (w^T x + b)y \quad \forall x \in X \iff 1 \leq \left(\frac{w^T x}{\kappa} + \frac{b}{\kappa} \right) y \quad \forall x \in X \quad (2.12)$$

Therefore it is sufficient to optimize $\operatorname{argmax}_w \frac{1}{\|w\|}$ with the constraint $1 \leq (w^T x + b)y$. This can be optimized with help of Lagrange multipliers and gradient descent [6]. For technical reasons we optimize an equivalent problem, which is given by,

$$\operatorname{argmin}_{(w, b)} \frac{1}{2} \|w\|^2 \text{ with } 1 \leq (w^T x + b)y \quad \forall x \in X \quad (2.13)$$

Using the Lagrange multipliers method we obtain a linear combination which gives us the best possible hyperplane [6]. This can be expressed by,

$$w^T z + b = \sum_{(x, y)} y \alpha_x x^T z + b \quad (2.14)$$

Where z is a new data point and the α_x are the Lagrange multipliers. Note that our hyperplane is only depending on our training data.

But what is the approach if we have more than two classes? The solution for this problem is to use multiple SVMs, which then in combination can distinguish a single class. There are two common approaches: "one-against-all" and "one-against-one" [10].

In the one-against-all approach we need m classifier for m classes. The i -th classifier separates the i -th class from the remaining other $m - 1$ classes.

The one-against-one approach is to use a classifier to linearly separate all pairs of classes. This would need $\frac{m(m-1)}{2}$ classifier and is therefore computationally more costly. However it can solve classification problems, which the one-against-all approach can't solve.

2.4. Ensemble Learning

In case of a classification task we can train multiple computational efficient, non deterministic models and make a majority vote for the label of a data point. let h_1, h_2, \dots, h_t be such classifiers. With $h_{i \in \{1, \dots, t\}} : X \rightarrow Y$. We can express the majority vote $V(x)$ as

$$V(x) = \max_{y \in Y} (h_1(x), h_2(x), \dots, h_t(x)) \quad (2.15)$$

This diversity of the weak classifiers can lead to astonishing results [21]. Examples for weak classifiers are Random-Forest and AdaBoost which are both really popular in practice [21].

To illustrate why Ensemble Learning can produce excellent results, consider the following example

2.4.1. Wisdom of the Crowd

Let $Y = \{0, 1\}$, let $x_i \in X$ be a data point with label y_i . Let $\{h_1, \dots, h_t\} = \mathcal{H}$ be the set of classifiers. We assume that,

$$P(h(x) = y) > \frac{1}{2} \quad \forall h \in \mathcal{H} \quad (2.16)$$

Which means that a classifier will be correct more than half of the time. We then see with Hoeffdings inequality [21], which is given by: let X_1, \dots, X_n be i.i.d. and it exist $a, b \in \mathbb{R}$ such that $a < X_i < b$, then

$$P \left[\left| \sum_{i=1}^n (X_i - \mathbb{E}[X_i]) \right| > \epsilon \right] < 2 \exp \left(-\frac{2n\epsilon^2}{(b-a)^2} \right) \quad (2.17)$$

that for large enough t the majority vote will be most likely correct. Therefore consider the random variable h_i with

$$h_i = \begin{cases} 1, & \text{if } h_i(x) = y \\ 0, & \text{otherwise} \end{cases} \quad (2.18)$$

2. Used Methods

Since equation 2.16 hold, we know there exist a $\delta > 0$ with $P(h_i) = \frac{1}{2} + \delta = \mathbb{E}[h_i]$. We now want to know what the is the probability that the majority vote makes the correct decision i.e. $P\left[\sum_i^t h_i > \frac{t}{2}\right]$. Taking the opposite event

$$P\left[\sum_i^t h_i \leq \frac{t}{2}\right] = P\left[\frac{1}{t} \sum_i^t h_i - \frac{1}{t} \leq 0\right] \leq P\left[\left|\frac{1}{t} \sum_i^t h_i - \left(\frac{1}{2} + \delta\right)\right| \leq \delta\right]$$

And with Hoeffding it now follows that,

$$P\left[\sum_i^t h_i \leq \frac{t}{2}\right] \leq 2 \exp\left(-2t\delta^2\right) \xrightarrow{t \rightarrow \infty} 0$$

This result shows us, that under the described condition the probability that the majority vote is wrong goes to zero for a large amount of classifiers. We however assumed that our voters are independent, which is usually not the case. It however still gives a good justification why Ensemble Learning can be strong.

3

Kernel Machines

So far in this work we presented linear projections. However it might be sometimes useful to first apply a non linear mappings to our data. By doing this we can extract some non linear characteristics of our data. As motivation we introduce the XOR example. let our data be,

$$X = \{(0, 0), (0, 1), (1, 0), (1, 1)\} \subset \mathbb{R} \quad (3.1)$$

with labels $Y = \{0, 1, 1, 0\} \subset \mathbb{R}$. It is known that there is no line which can separate the two given classes. The idea now is to map our data into another space, where it is linear separable. Therefore consider,

$$\phi(x_1, x_2) = (x_1, x_2, x_1x_2) \in \mathbb{R}^3 \quad (3.2)$$

The projected data is now separable. This can be observed in figure 3.2. Note that for linear separating our data, our image space for ϕ does not necessarily have higher dimension than the data space. Furthermore, this is usually the case. There is however still a big disadvantage. The computation of ϕ can be very costly. The projection has to be calculated for every data point in our dataset. Luckily there is a work around, the so called kernel trick, which will be explained in the following sections.

3. Kernel Machines

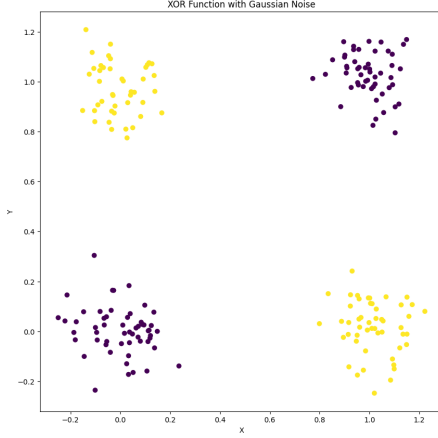


Figure 3.1.: original data

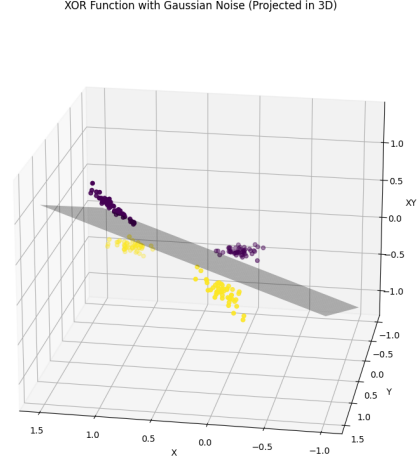


Figure 3.2.: projected data

3.1. Feature Space and Kernels

First of all we need to define an inner product to further investigate the theory.

Definition 1. Let \mathcal{H} be a vector space over \mathbb{R} . A map $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}$ is called inner product on \mathcal{H} if,

1. $\langle \lambda x + \mu y, z \rangle = \lambda \langle x, z \rangle + \mu \langle y, z \rangle \quad \forall x, y, z \in \mathcal{H}$
2. $\langle x, y \rangle = \langle y, x \rangle \quad \forall x, y \in \mathcal{H}$
3. $0 \leq \langle x, x \rangle$ and $0 = \langle x, x \rangle$ if and only if $x = 0$

We can now also define a norm¹ on \mathcal{H} with $\|x\|_{\mathcal{H}} = \sqrt{\langle x, x \rangle_{\mathcal{H}}}$. A complete vector space on which an inner product is defined is also called Hilbert space. We now define the so called kernel as follows.

Definition 2. Let X be our data set. A function $k : X \times X \rightarrow \mathbb{R}$ is called kernel if there exists an Hilbert space \mathcal{H} and a map $\phi : X \rightarrow \mathcal{H}$ such that $\forall x, y \in X$,

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}} \quad (3.3)$$

¹One can check that the defined norm really fulfills the conditions for being a norm.

3. Kernel Machines

These so-called kernels have many beautiful properties that can be shown. One of this properties is, that a kernel function is positive definite which means that for all data sets X the matrix K is positive definite. This will later be useful. The kernel matrix K is given by,

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \quad (3.4)$$

However, the complete theory behind this properties is beyond the scope of this paper. We also call the space \mathcal{H} the feature space [8]. A linear decision boundary in \mathcal{H} for separating the data and labeling new data z can than be expressed with by $w \in \mathcal{H}$ with,

$$w^T \phi(z) + b = \langle w, \phi(z) \rangle_{\mathcal{H}} + b = \sum_{x,y} \alpha_x y \phi(x) \phi(z) + b = \sum_{x,y} \alpha_x y k(z, x) + b \quad (3.5)$$

Where w now describes the best possible decision boundary and the α_x are Lagrange multipliers resulting from the optimization problem [4].

The dimension of \mathcal{H} however can be arbitrary large. It even can be infinity. This makes the computation of the inner product in \mathcal{H} extremely computational costly or even impossible to perform in \mathcal{H} . Luckily there is a work around.

3.2. Kernel Trick

This work around is the so called kernel trick. It allows to compute the inner product in \mathcal{H} indirectly. Therefore we need the Mercer's theorem [19]

Mercer's Theorem: A symmetric function k can be expresses as an inner product, i.e.

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \quad (3.6)$$

for some $\phi : X \rightarrow \mathcal{H}$ if and only if k is positive semi definite.

3. Kernel Machines

This implies that for any given $\phi : X \rightarrow \mathcal{H}$ we do not need to compute the mapping and calculate the inner product afterwards. We directly can find a function $k : X \times X \rightarrow \mathbb{R}$, which computes $\langle \phi(x), \phi(y) \rangle$ [14]. This gives us the existence of k to find the hyperplane in equation 3.5.

Note that this equation is similar to the function of a linear SVM 2.14. We just use the kernel function k instead of an inner product. We call this optimization problem kernelized SVM or kernel SVM.

The given kernel function for our example in equation 3.2 would be ,

$$\langle \phi(x_1, x_2), \phi(y_1, y_2) \rangle = x_1 y_1 + x_2 y_2 + x_1 x_2 y_1 y_2 = k(x, y) \quad (3.7)$$

3.2.1. Radial Bias Function Kernel

A common kernel which is often used and will also be later relevant for our definition of Random Fourier Features is the radial bias function (RBF) kernel. This kernel is defined as:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad (3.8)$$

Not that our kernel k is depending on the distance between two points. This can express neighborhood relations [19]. There are also a lot of different kernels out there [19], which can express non linear dependencies. An example can be seen in 3.2.1.

3. Kernel Machines

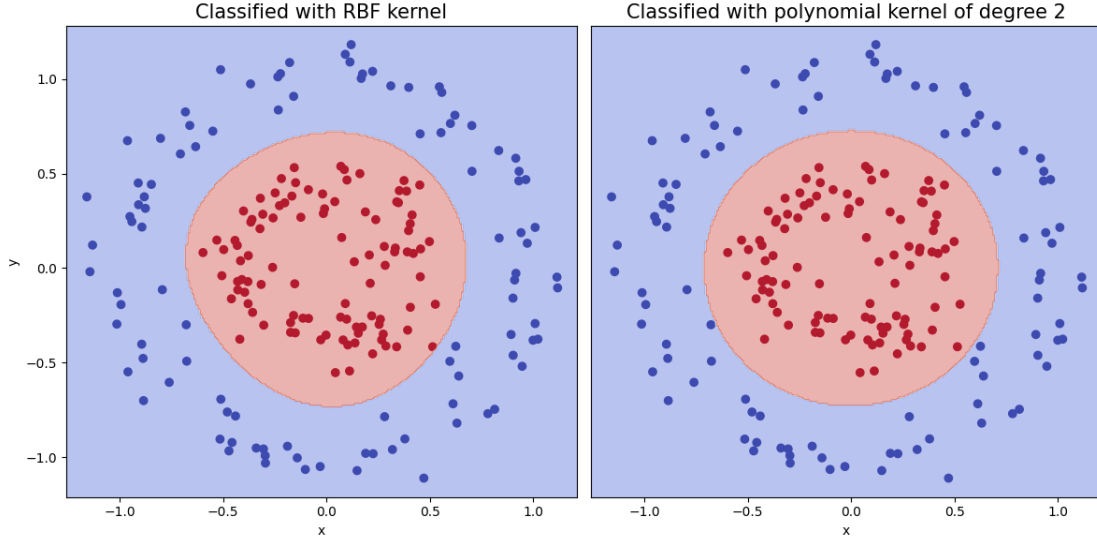


Figure 3.3.: Data was classified with the RBF-kernel SVM (left). Data was classified with polynomial kernel (right).

3.2.2. Scaling of Kernel Machines

The idea behind the kernel trick is a nice and beautiful. However with large data set the computation also rises immensely. Equation 3.5 shows us how to find a label for a new data point z . Since we do not compute $\phi(z)$ directly we have to iterate over every $x \in X$ and compute $\alpha_x k(x, z)$.

In case for linear regression in \mathcal{H} this gets even harder. From linear algebra its known that the optimal regression line \hat{w} is given by,

$$\hat{w} = (X^T X)^{-1} X^T y \quad (3.9)$$

Considering this problem in the feature space \mathcal{H} . with $\Phi = \phi(X)$ component wise. We get,

$$\hat{w} = (\Phi^T \Phi)^{-1} \Phi^T y = \langle \Lambda, \Phi \rangle \text{ with } \Lambda = (\Phi^T \Phi)^{-1} y \quad (3.10)$$

3. Kernel Machines

Which than can be rewritten as $\sum_{x \in X} \Lambda_x \phi(x)$. Note that we not have to compute $\phi(z)$. By simply rearranging the equation we get,

$$y = w^T \phi(z) = y(\Phi^T \Phi)^{-1} \Phi^T \phi(z) = (\Phi^T \Phi)^{-1} \sum_{x \in X} k(x, z) \quad (3.11)$$

However it still remains the computation of the inverse kernel matrix $K = \Phi^T \Phi$, which causes problems. This calculation brings the heave computation time since, it scales more than squared with the amount of data points. K is given by,

$$K = \Phi^T \Phi = (\phi(x_1), \dots, \phi(x_n))^T \cdot (\phi(x_1), \dots, \phi(x_n)) \quad (3.12)$$

This results into,

$$K = \begin{pmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \cdots & \langle \phi(x_1), \phi(x_n) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{pmatrix} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

which is the above defined kernel matrix. Note that this also shows that regression can be kernelized. This is often done with a regularization factor and is known as kernel ridge regression [19].

It is now obvious with this example that the computation time can be very large. Another example using gaussian processes is given in [4]. We therefore need another approach which does not need as much computation time. We therefore try to approximate the kernel k with random features, which does lead to astonishing results. We will discuss this approach in the next chapter.

3. Kernel Machines

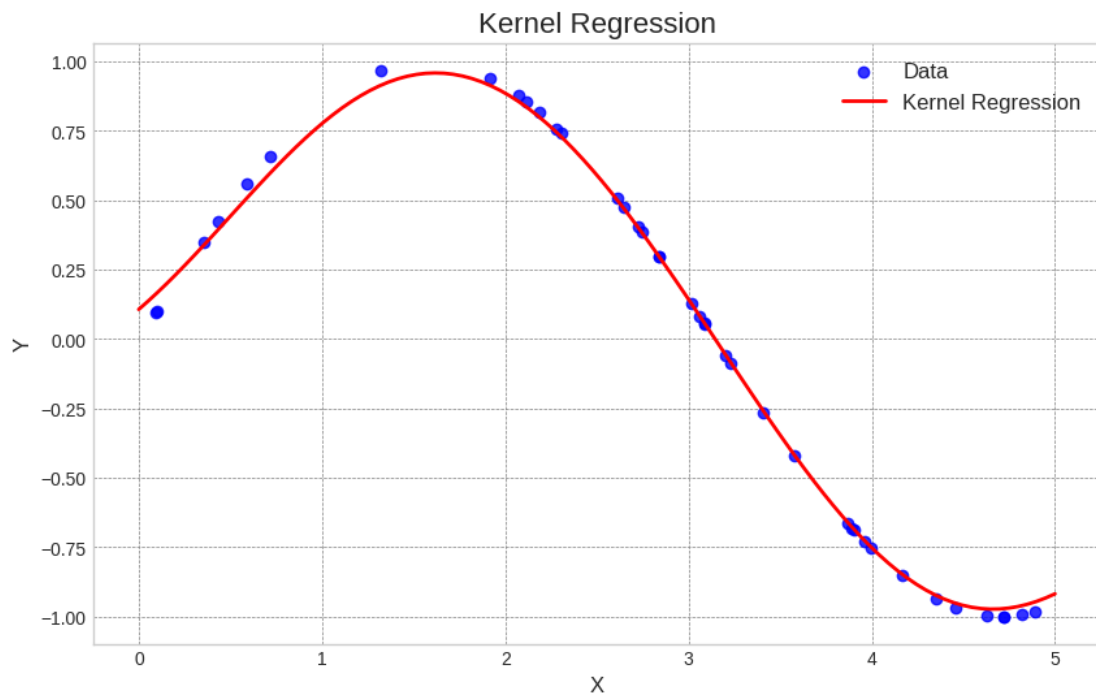


Figure 3.4.: Data was sampled out of a sin wave and then transformed with the RBF-kernel. The linear regression was then implicitly performed in the feature space.

4

Random Fourier Features

4.1. Random Features

Again consider a kernel function $k : X \times X \rightarrow \mathbb{R}$ and the corresponding map $\phi : X \rightarrow \mathcal{H}$, with $X \subset \mathbb{R}^d$ which maps in our feature space. Let now for simplicity $\mathcal{H} = \mathbb{K}^N$ which is usually the case when projecting data ¹. Our goal now is to find a randomized map $z : X \rightarrow \mathbb{K}^M$ with $M \ll N$ i.e.

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x)^T z(y) = \langle z(x), z(y) \rangle \quad (4.1)$$

The vector $z(x)$ is then called random feature [16]. It would then be possible to approximate equation 3.5 to find a label for a new data point x^* with,

$$w^T \phi(x^*) + b = \sum_{x \in X} \alpha_x k(x, x^*) + b \approx \sum_{x \in X} y \alpha_x z(x)^T z(x^*) + b \quad (4.2)$$

Also the kernel matrix in equation 3.12 could be approximated and calculated with less computing time.

4.1.1. Approximating Gaussian Kernel

Before deriving the exact definition of a Random Fourier Features (RFFs) first consider the following example, where we approximate a kernel with random features. We will later see that the random features in this example are actually RFFs.

¹ \mathbb{K} could be either the real space \mathbb{R} or the complex space \mathbb{C} .

4. Random Fourier Features

Consider again the RBF or also called gaussian kernel from equation 3.8. We now sample w from a gaussian distribution,

$$w \sim \mathcal{N}(0, I_d) \quad (4.3)$$

Where I_d is the identity matrix with dimension $d \times d$. This means that the entries of $w \in \mathbb{R}^d$ are not correlated.

Now consider $z : X \rightarrow \mathbb{R}$ with,

$$z : x \rightarrow \exp(iw^T x) \quad (4.4)$$

with i as the imaginary unit. We now want to find out if $\langle z(x), z(y) \rangle$ does approximate our gaussian kernel. We can observe that $\langle z(x), z(y) \rangle^2$ is indeed unbiased.

$$\begin{aligned} \mathbb{E}_w[\langle z(x), z(y) \rangle] &= \mathbb{E}_w[z(x)^T z(y)^*] = \mathbb{E}_w[\exp(iw^T(x - y))] \\ &= \int_{\mathbb{R}^d} g(w) \exp(iw^T(x - y)) dw = \exp\left(-\frac{1}{2}(x - y)^T(x - y)\right) \\ &= \exp\left(-\frac{\|x - y\|^2}{2}\right) = k_{RBF}(x, y) \end{aligned}$$

where $g(w)$ describes the PDF of the multivariate gaussian defined in equation 4.3, which is given by ³,

$$g(w) = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{\|w\|^2}{2}\right) \quad (4.5)$$

The dimension of the random feature space is then given by the amount of samples. By sampling R i.i.d w_r our map $z : \mathbb{R}^d \rightarrow \mathbb{K}^R$ is given by,

$$z : x \rightarrow \left(\frac{1}{\sqrt{R}} z_{w_1}(x), \frac{1}{\sqrt{R}} z_{w_2}(x), \dots, \frac{1}{\sqrt{R}} z_{w_R}(x) \right)^T \quad (4.6)$$

²Note that in case of the complex space \mathbb{C} the inner product $\langle z(x), z(y) \rangle$ is defined as $\sum_i x_i \cdot y_i^*$ to fulfill the conditions in definition 1 where y^* is the complex conjugate of y . in case of $y = e^{iwx}$, y^* would be e^{-iwx}

³We used the fact that the covariance matrix Σ is the identity matrix I with $\det(I) = 1$ and the fact that $\mu = 0$.

4. Random Fourier Features

And the inner product, which approximates the kernel is given by

$$\langle z(x), z(y) \rangle = \frac{1}{R} \sum_r z_{w_r}(x) z_{w_r}(y)^* \quad (4.7)$$

It's also possible to show the convergence in probability of the inner product of our approximated features towards the true kernel function, if R is large enough i.e

$$\mathbb{P} \left[\sup_{x,y \in X} |\langle z(x), z(y) \rangle - k(x,y)| \geq \epsilon \right] \xrightarrow{R \rightarrow \infty} 0 \quad (4.8)$$

for $\epsilon > 0$ and under the condition that X is compact. For the prove we refer to [16]. An empiric demonstration can be seen in figure 4.1.1. By increasing the amount of random features the resulting Moore-Penrose inverse for the regression task gets more similar, i.e.

$$(Z_X^T Z_X)^{-1} Z_X^T y \xrightarrow{R \rightarrow \infty} (\Phi^T \Phi)^{-1} \Phi^T y$$

where Φ is the kernel matrix, defined as in 3.12 and Z_X is the approximated kernel matrix.

4. Random Fourier Features

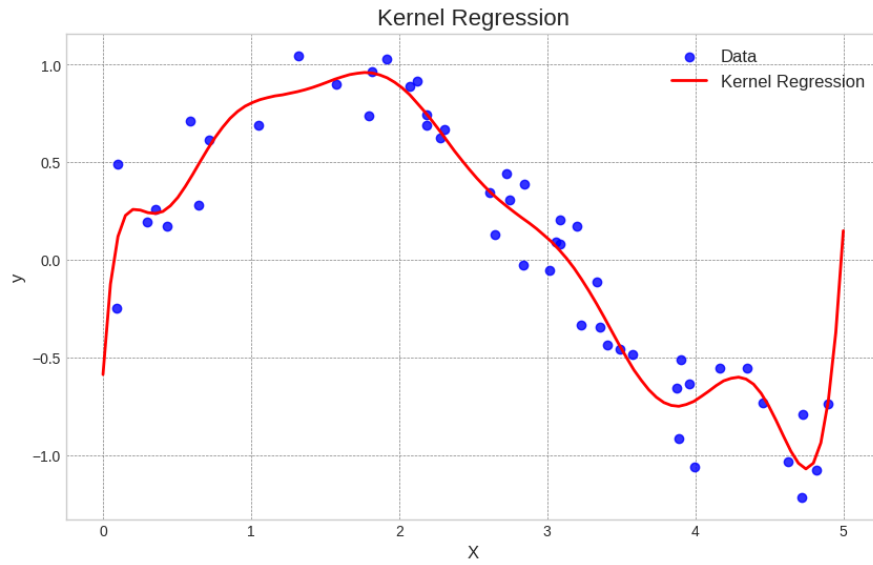


Figure 4.1.: Regression with original kernel. No regularization was used. Data was sampled out of a sin wave with additional gaussian noise.

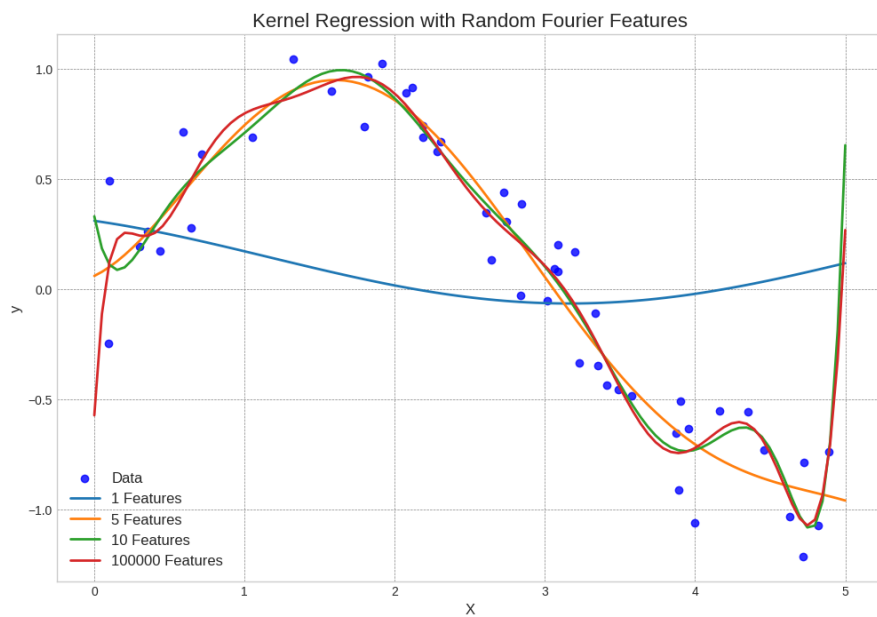


Figure 4.2.: Regression with approximated kernels. To approximate the kernel 1,5, 10, 100000 Random Fourier Features were used.

4.2. Generating Random Fourier Features

We still need a proper definition for RFFs. To properly define it we first need Boncher's theorem [17].

Boncher's theorem: A continuous and translations-invariant kernel $k(x, y) = k(x - y)$ is positive definite if and only if $k(\Delta)$ is the Fourier transform of a non-negative measure $p(w)$ i.e.

$$k(\Delta) = \int p(w) \exp(iw\Delta) dw$$

Translations-invariant means that $\forall x, y, z \in X \ k(x, y) = k(x - z, y - z)$. This means that the kernel function does only depend on the distance between x and y . We also write $k(x - y)$

Definition 3. We call the map $z : \mathbb{R}^d \rightarrow \mathbb{K}^R$ a Random Fourier Feature if it approximates a translations-invariant positive definite kernel k , i.e.

$$\mathbb{E} [\langle z(x), z(y) \rangle] = k(x - y) = k(x, y) \quad (4.9)$$

A kernel is positive definite, if it's kernel matrix defined in 3.12 is positive definite. Note that this implies that our random features given in the example 4.1.1 are indeed Random Fourier Features. Since the RBF kernel does only depend on the distance between x and y . The measure $p(w)$ in this case was the PDF of a multivariate gaussian.

4.2.1. Sampling Random Fourier Features

Definition 3 does allow various valid Random Fourier Features. In our experiment the sampled features are inspired by [16]. z is then constructed as follows, we first sample w and b with,

4. Random Fourier Features

$w \sim p(w)$ Where we usually will use a gaussian

$b \sim U(0, 2\pi)$ Where U describes the uniform distribution

$$z_w(x) = \sqrt{2} \cos(w^T x + b)$$

This does indeed approximate a translations-invariant kernel since,

$$\begin{aligned} \mathbb{E}_w [z_w(x), z_w(y)] &= \mathbb{E}_w [\sqrt{2} \cos(w^T x + b), \sqrt{2} \cos(w^T y + b)] \\ &= \mathbb{E}_w [\cos(w^T(x + y) + 2b)] + \mathbb{E}_w [\cos(w^T(x - y))] \\ &= \mathbb{E}_w [\cos(w^T(x - y))] \end{aligned} \quad (4.10)$$

This is also the expected value of our RBF-kernel, since w and k are both real valued. It therefore follows out of Euler's formula. Which gives us that

$$\begin{aligned} \mathbb{E} [\exp(iw^T(x - y))] &= \mathbb{E} [\cos(w^T(x - y))] + \mathbb{E} [i \sin(w^T(x - y))] \\ &= \mathbb{E} [\cos(w^T(x - y))] \\ &= \mathbb{E}_w [z_w(x), z_w(y)] \end{aligned} \quad (4.11)$$

4.2.2. Nyström Method

Another way to approximate the kernel matrix with kernel k is given by the Nyström method. Therefore we sample $\hat{x}_1, \dots, \hat{x}_l \in X$ with $l \leq |X| = n$. We then generate $\hat{K} = K_b \hat{K}^+ K_b^T$ which approximates our original kernel matrix [13] where

$$K_b = \begin{pmatrix} k(\hat{x}_1, x_1) & \cdots & k(\hat{x}_1, x_n) \\ \vdots & \ddots & \vdots \\ k(\hat{x}_l, x_1) & \cdots & k(\hat{x}_l, x_n) \end{pmatrix}, \hat{K} = \begin{pmatrix} k(\hat{x}_1, \hat{x}_1) & \cdots & k(\hat{x}_1, \hat{x}_l) \\ \vdots & \ddots & \vdots \\ k(\hat{x}_l, \hat{x}_1) & \cdots & k(\hat{x}_l, \hat{x}_l) \end{pmatrix}$$

and \hat{K}^+ describes the pseudo inverse [15]. Since \hat{K} is symmetric and positive semi-definite, because it's a kernel matrix, we can find an eigenvalue decomposition with $\hat{K} = \hat{V} \hat{\Lambda} \hat{V}^T$. Here are \hat{V} the eigenvectors and $\hat{\Lambda}$ is a diagonal matrix with eigenvalues.

4. *Random Fourier Features*

We then define our Nyström Feature

$$z(x) = \hat{\Lambda}^{-\frac{1}{2}} \hat{V}^T (k(\hat{x}_1, x) \cdots k(\hat{x}_1, x))^T \quad (4.12)$$

We see, that our Nyström Features also approximate our kernel, since

$$z(x_i)^T z(x_j) = \hat{K}_{i,j} \approx k(x_i, x_j) \quad (4.13)$$

5

Experiments and Results

5.1. Experimental Setup

In the experimental part of this work we want to investigate two different applications of Random Fourier Features. We have two important metrics, which we want to evaluate. First we are interested in the performance of our model with respect to amount of Random Fourier Features. We are also interested in the time, which is needed to train our model. Also with respect to the amount of Random Fourier Features.

We first want to experiment on a regression task, where we approximate the kernel of a kernel ridge regression with help of Random Fourier Features. As a metric to evaluate the performance of the approximation we use the mean squared error (MSE), which is given by,

$$\mathcal{L}(X) = \sum_{x,y \in X \times Y} (w^T \phi(x) - y)^2 \quad (5.1)$$

We will train our models on two different data sets.

In the second part we will do several experiments on classification. As a score we use accuracy, which is just the proportion of correctly classified data points.

We will use SVM for classification where the kernel is approximated with RFFs. The approximated kernel will be the RBF-kernel. We will compare the performance and computing time of Random Fourier Features, with Nyström Features and kernel SVM with a prior dimensionality reduction with help of PCA.

We also investigate the influence of noise in our datasets.

At the end of this section we will use Ensemble Learning with help of RFFs. This is a follow up work to [18]. We therefore train multiple weak classifiers with help of RFFs and

5. Experiments and Results

then take the majority vote for classification. We also evaluate accuracy and computing time.

5.1.1. Datasets

In the following experiments we will need two types of datasets. One type is for regression and the other type is for classification. For kernel regression we use "Avocado Price" data set which is suitable for kernel regression [1] and describes various features, which influences the price. We also use the wine data set from Kaggle, which describes the amount of various chemicals in wine and their effect on it's quality [3]. The fact about the data sets can be viewed in the table 5.1.

Dataset	#Entries	#Features
Avocado	18 200	8
Wine	1 143	11

Table 5.1.: Dataset Avocado and Wine Quality for Regression

Dataset	#Entries	#Classes
MNIST	70 000	10
Fashion-MNIST	70 000	10

Table 5.2.: Dataset MNIST and Fashion-MNIST for Classification

For the classification task, we use the MNIST and Fashion-MNIST datasets 5.2. We however modify the data set in some ways. First we apply a 2×2 max-pooling operation to every image of the dataset. This does reduce the computing time and more importantly the amount of RAM needed to compute the RBF-kernel, since the dimension is reduced. Even though SVM works well for very high dimensional data [9]. An illustration on MNIST samples can be seen in figure 5.1.1. After applying max-pooling the data is normalized for training.

5. Experiments and Results

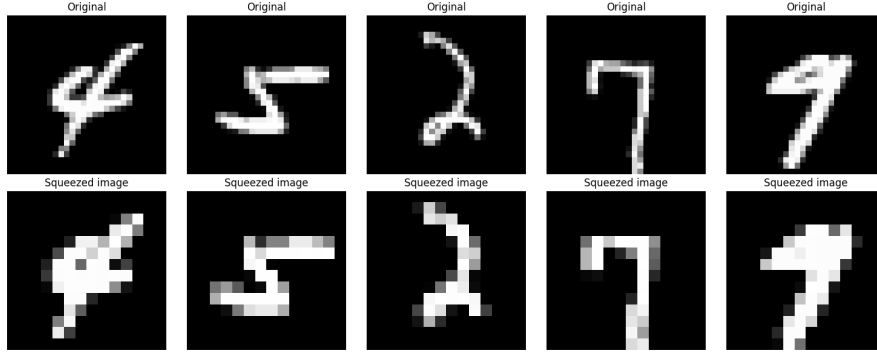


Figure 5.1.: Max pooling operation on MNIST data points. On the top are the original images. On the bottom are the compressed images.

5.1.2. Hyperparameter

For regression we use ridge regression with regularization factor .

$$\mathcal{L}(X) = \sum_{x,y \in X \times Y} (w^T \phi(x) - y)^2 + \frac{\lambda}{2} \|w\|^2 \quad (5.2)$$

$\lambda = 1$ is used for both datasets.

As a kernel we used RBF-kernel. The variance σ of our RBF-kernel is set to $\sigma = 0.5$ for the regression task and $\sigma = 2$ for the classification task. The values were found by experimenting with different settings for σ .

For the Ensemble Learning we used 13 classifiers. It is important to have more classifiers than classes to give a proper voting.

5.2. Approximating Kernel Regression

As already introduced in the previous chapter, in the first experiment we want to do a regression task. Therefore we approximate our kernel, in this case a RBF-kernel, by different amount of features. We calculate the loss and the time needed to process the regression with the random features. We then compare the results with the normal RBF-kernel regression. This experiment was executed on two datasets. The first dataset [1] is the "Avocado Dataset". The second dataset is the "Wine Dataset" [3]. The

5. Experiments and Results

experiment was executed 5 times to calculate the average and reduce the variance. The results can be observed in figure 5.2.

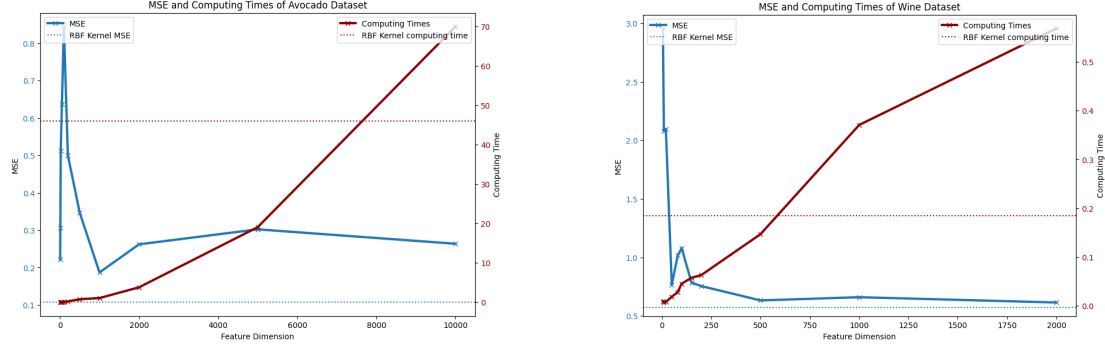


Figure 5.2.: This figure shows the approximated kernel regression. The datasets avocado pricing (left) wine-quality (right) are used. The x-axis indicates the amount of RFFs used and the y-axis represents the MSE. The dotted lines represent time and MSE of RBF-kernel regression

5.3. Approximating Kernel of Kernel SVM

In the next experiment we want to work with SVM. SVMs are often combined with kernel function [19] [9]. We now want to approximate the RBF-kernel with Random Fourier Features. We therefore train a linear SVM of the feature space \mathcal{Z} which contains the random projections of our data points. For the experiment the Random Fourier Features introduced in 4.1 are used. For the density function q we use a multivariate gaussian. We are first interested in the classification accuracy of our SVM trained on the feature space with respect to the amount of RFFs. We also want to compare the accuracy to a SVM learned on our original data with a RBF-kernel.

The time needed to fit the SVM is also of big interest. We therefore investigate the computing time with respect to the amount of random features as well. The results can be observed in figure 5.3.

5. Experiments and Results

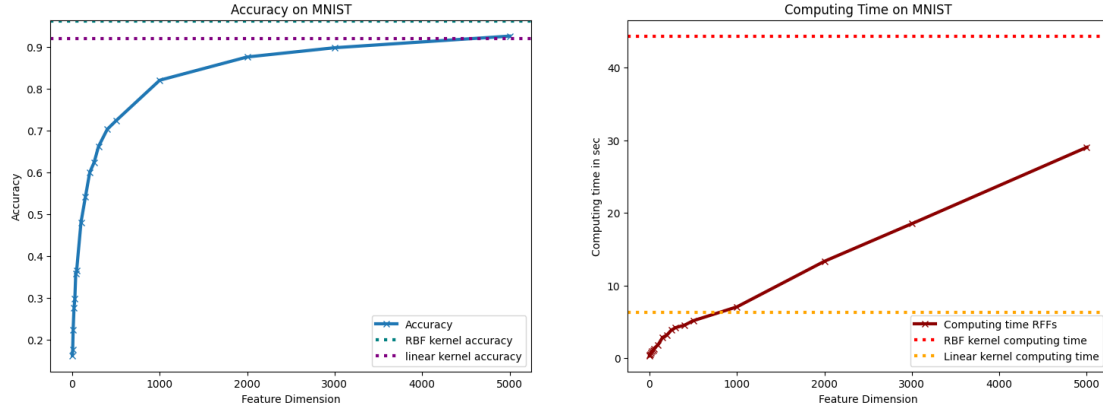


Figure 5.3.: SVM trained on feature space with was induced by RFFs. The left plot shows the accuracy with respect to the amount of RFFs. The right plot shows the amount of time needed to train the models also with respect to the amount of RFFs. Accuracy and execution time of SVM with RBF-kernel and linear kernel can also be observed.

5.3.1. Comparing Random Fourier Features with Nyström Features

In this subsection we want to compare Random Fourier Features with Nyström Features 4.2.2, which is also a kernel approximation method. We therefore compare accuracy and execution time with respect to the dimension of the feature space. Again we use a linear SVM which is then trained on the feature space. The results can be observed in figure 5.4.

5. Experiments and Results

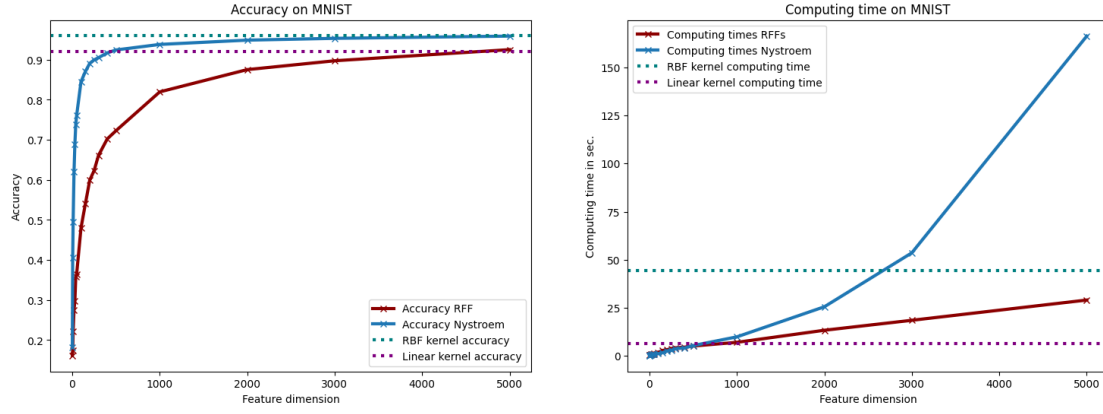


Figure 5.4.: Test-accuracy and execution time with respect to the dimension of the feature space. The blue line represents a SVM trained with Nyström Features and the red line represents SVM trained with Random Fourier Features. Accuracy of RBF-kernel SVM and linear SVM can also be observed.

5.3.2. Dimensionality Reduction with PCA

In this experiment we want to compare the performance of RBF-kernel SVM on compressed data. Therefore the data is reduced with help of PCA and afterwards a kernel SVM is trained on the compressed data.

We evaluate the test-accuracy with respect to the dimension of the compressed data. We also measure the time which is needed to combine PCA with a kernel SVM. The results can be seen in figure 5.5

5. Experiments and Results

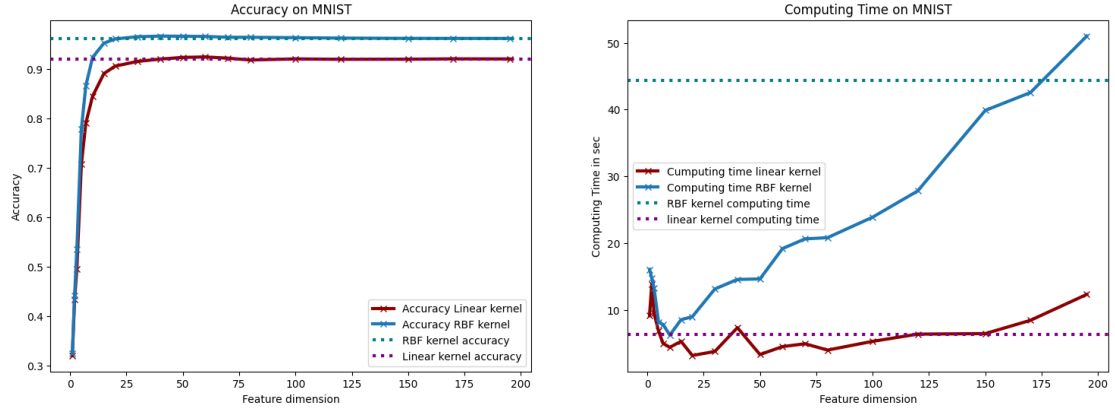


Figure 5.5.: Accuracy with respect to the input dimension for a RBF-kernel SVM (blue) and a linear kernel SVM (red). The dimensional reduction was done with help of PCA.

The computing time with respect to the feature dimension can be seen in the right picture. The dotted lines indicate the performance without dimensional reduction.

5.3.3. Noisy Data

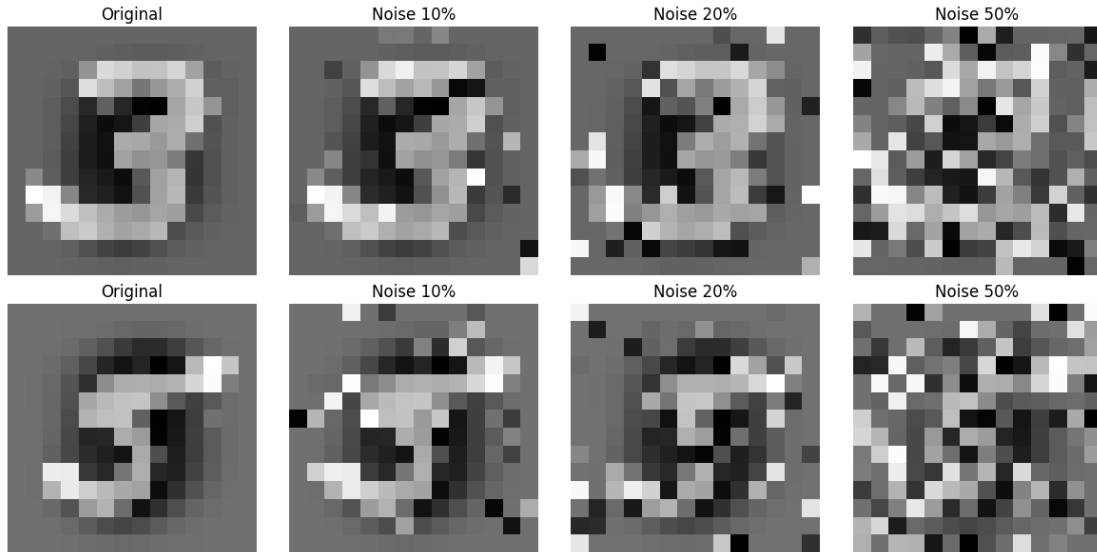


Figure 5.6.: Data points with noise $p \in \{0, 0.1, 0.2, 0.5\}$.

To investigate the impact of noise on our model we generate artificial noise in our dataset. Therefore we pick $p \in [0, 1]$ which is now the probability that a value of pixel of our

5. Experiments and Results

training images is now chosen uniform at random. An illustration of the resulting training data with different level of noise can be seen in figure 5.6.

We again measure performance and time with respect to the amount of features used for approximate the kernel. We train our models with different amount of noise in the training data. The results can be seen in 5.7. We also evaluate the impact of noise on the RBF-kernel. this can be observed in figure 5.7 (right). A dimensional reduction with help of PCA is also performed after applying noise.

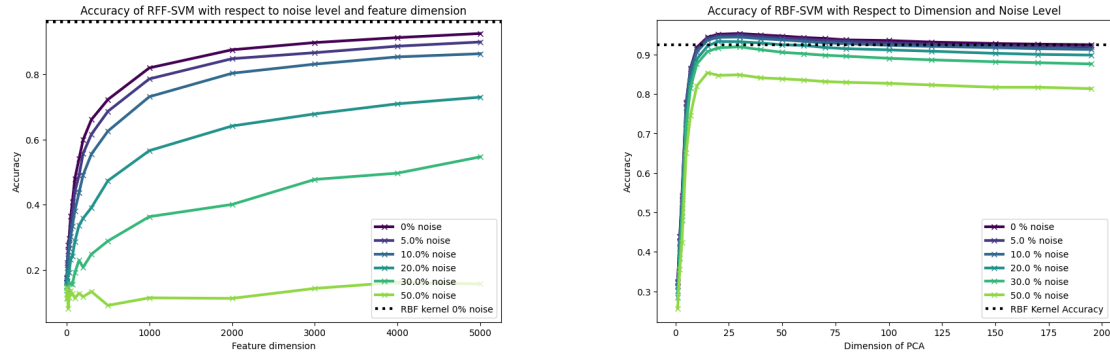


Figure 5.7.: In the left image we see the impact of noise to SVM with approximated RBF-kernel. The x axis indicates the feature dimension of the feature space, where the y axis measures the accuracy of the models. The experiment was executed six times with different amount of noise.

In the right image we see the impact of noise to kernel SVM. The data is projected into lower dimensional space with help of PCA. The x axis indicates the dimension after projection.

5.4. Ensemble Learning

We now want to investigate the performance if we combine multiple simple classifiers. One simple classifier is a approximated kernel SVM as described in experiment 5.3. We use 13 classifiers and majority vote to define the ensemble classifier. We measure the accuracy of the combined classifier with respect to the used RFFs in the simple classifiers. We also measure the average accuracy of the classifiers. This can be seen in figure 5.4

In addition we also track the computing time needed to train the combined classifier in total. This can be also observed in figure 5.4.

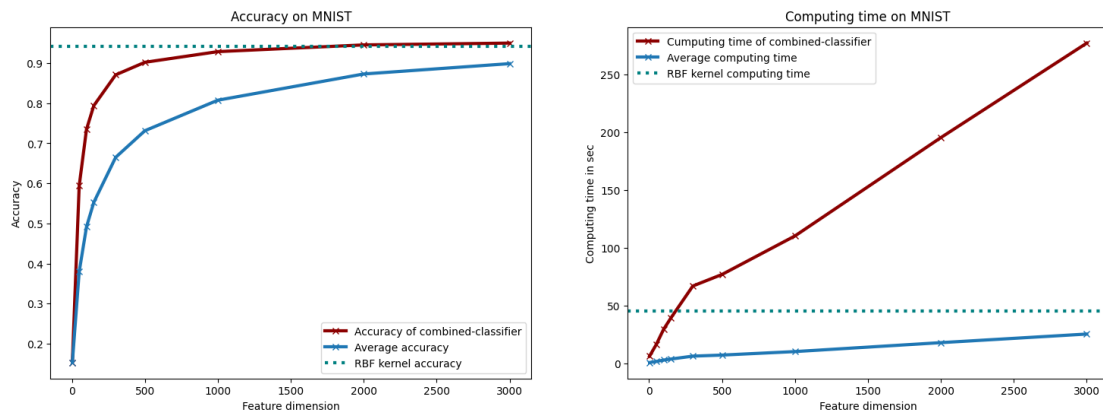


Figure 5.8.: Accuracy and computing time of the majority vote of 13 classifiers (red). Average accuracy and computing time of a single classifier (blue). The dotted line indicates accuracy and computing time of a single RBF-kernel SVM.

In addition we measure the impact of the amount of classifiers used for the combined classifier. We also compare the results to a single RBF-kernel SVM. The results can be observed in figure 5.4.

5. Experiments and Results

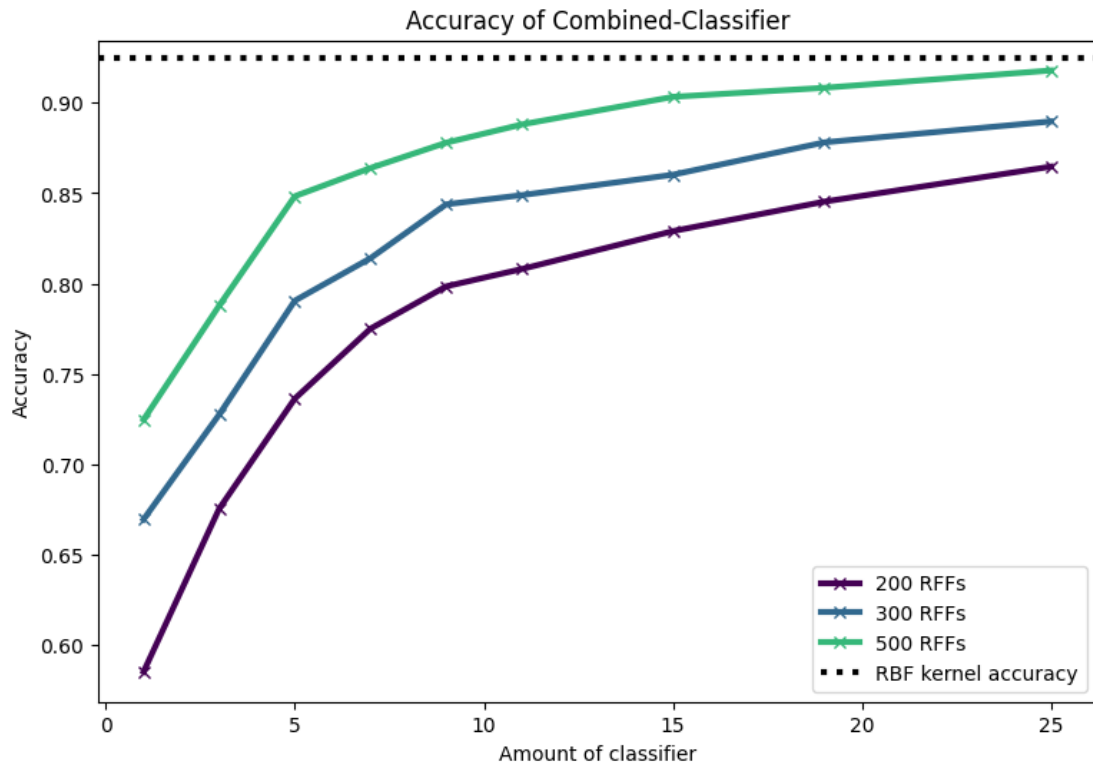


Figure 5.9.: Accuracy with respect to the amount of voters in the combined classifier. We evaluate the accuracy for 200, 300 and 500 RFFs. The dotted line indicates the accuracy of a single RBF-kernel SVM.

6

Discussion

6.1. Evaluating the Results

In the previous chapter, we conducted three fundamentally different experiments. First, we presented a regression problem. For this purpose, we approximated an RBF-kernel with the help of RFFs and then performed a linear regression in feature space.

In the second part of the experiments we presented classification problems. These were to be solved with SVMs. Again, the kernel was approximated with RFFs.

Finally, we took advantage of the randomness of the kernel approximation and used majority voting to merge several simple classifiers into a combined classifier.

6.1.1. Approximating Kernel Regression

For both regression problems, it can be seen that the MSE is lower for the non-approximated kernel regression. However, the performance with the approximated kernel is still good and comes very close to the original kernel regression, especially in the case of the Wine dataset.

It can also be observed that for both datasets the phenomenon occurs that a smaller dimension of the feature space can lead to better performance. This is probably due to a regularizing effect with a smaller dimension of the feature space. Similar observations have already been made in [5].

6. Discussion

We also make the observation that with less calculation time, solid results can still be obtained. For example, with the Avocado dataset, a good performance can be achieved with 1000 RFFs. The computation time at 1000 RFFs is still many times less than the time needed to compute a normal RBF-kernel regression.

6.1.2. Approximating Kernel SVM

In 5.3 we see that as the number of RFFs increases, the accuracy of the classification increases. This was to be expected as our RBF-kernel becomes better approximated with increased feature space dimension. With a feature dimension of 5000, our model has better accuracy than a linear SVM. However, the accuracy of the RBF-kernel SVM is still better. On the other hand, the computation time of the RBF-kernel SVM is much longer than the computation time of SVMs with approximated kernel. The calculation time increases linearly in dependence of the RFFs. This can also be seen in figure 5.3.

In 5.4 we compare the performance between Nyström Features and RFFs. We observe that the Nyström approximation leads to better accuracy than the approximation using RFFs. However, the computation time of the Nyström approximation is longer and does not grow linearly. This is also to be expected since with increasing Nyström Features more and more complex matrix multiplications and diagonalization procedures have to be performed, which do not necessarily scale well. We also observe that the Nyström approximation performs almost as well as the RBF-kernel SVM. This is due to the fact that the approximated kernel matrix becomes more and more similar. When the same number of Nyström Features as data points are used, the approximated kernel matrix is the original kernel matrix. This also shows equation 4.12.

As a good and time-saving alternative to the approximation using RFFs, the method presented in 5.5 can be used. First, reducing the dimension using PCA and then using an RBF-kernel SVM can lead to astonishing results. Even with a feature dimension of 25, in the case of MNIST, the performance is similar to that of an RBF-kernel SVM trained on the original data. However, this effect might be particularly strong in our datasets, as many pixels in the images have little variance. This effect for example is not as strong in

6. Discussion

the case of Fashion-MNIST, as shown in appendix in A.2.

It can even be observed that the projection into lower-dimensional spaces using PCA can have a regularizing effect. In some cases, the accuracy is even higher when the projected data is used instead of the original data for training an RBF-kernel SVM.

The PCA into RBF-kernel SVM method is also more robust against noise, as can be observed in figure 5.6. With high levels of noise, the kernel approximation using RFFs becomes almost random. However, this is not the case with the other method. The expected possible regularization of noise with RFFs is not observed.

6.1.3. Evaluation of Ensemble Learning

The performance of the combined classifier is very good, and the accuracy of the combined classifier increases much faster with the increase in RFFs compared to the individual classifier. This clearly demonstrates the usability of RFFs for Ensemble Learning. The strong increase of accuracy with low amount of RFFs in the combined classifier is do to the fact that there is already a small sense of the data given. However the variance between single classifier might be still very large with less features. This can boost Ensemble Learning, since we have more variety in the voters. For large amount of RFFs the gap between the single classifier and the combined classifier gets smaller. The variance between classifiers will be smaller for high feature dimension. For extremely large amount of features, we would approximate the RBF-kernel perfectly. In this case majority voting and single voting would yield the same results. Since the voters would be totally correlated. A low feature dimension for Ensemble Learning is therefore recommended, since it also saves computing time.

With a high number of RFFs, the performance of the ensemble classifier even slightly surpasses that of the individual RBF-kernel SVM. The influence of the number of voters is also well observed, as expected. With a feature dimension of 500 and 25 voters, the accuracy of the combined classifier approaches the accuracy of the RBF-kernel SVM.

6.2. Conclusion

In conclusion, our experiments have shown promising results using RFFs for different machine learning tasks.

We have shown that the approximation using RFFs is a time-saving and good alternative to normal kernel methods, even though the performance may not always match that of the original kernel methods completely.

Additionally, using PCA, efficient and sometimes even better models can be built, which are deterministic and always lead to the same results.

On the other hand, the kernel approximation with RFFs is non-deterministic, allowing the creation of a combined classifier. This combined classifier, which takes the majority vote of multiple weak classifiers, exhibits excellent performance and provides a compelling alternative to traditional Ensemble Learning methods such as Random Forests.

6.3. Future Work

For future work it would be interesting to combine the kernel approximation Method with other machine learning algorithm. An interesting approach would be to investigate the performance of deep Neural Networks (DNNs) where the input data is the projected data in the feature space induced by the RFFs.

We only used one type of RFFs. Also the performance of different RFFs could be evaluated.

It would be also interesting to compare our combined classifier with other common ensemble classifiers and evaluate computing time and performance.

Bibliography

- [1] Avocado price, howpublished = <https://www.kaggle.com/datasets/neuromusic/avocado-prices>, year = , note = [Accessed 15-Jul-2023],.
- [2] Ulmer Münster – Wikipedia — [de.wikipedia.org](https://de.wikipedia.org/wiki/Ulmer_M%C3%BCnster). https://de.wikipedia.org/wiki/Ulmer_M%C3%BCnster. [Accessed 15-Jul-2023].
- [3] Wine Quality Dataset — [kaggle.com](https://www.kaggle.com/datasets/yasserh/wine-quality-dataset). <https://www.kaggle.com/datasets/yasserh/wine-quality-dataset>. [Accessed 16-Jul-2023].
- [4] *Introductory Functional Analysis with Applications*. Wiley classics library. Wiley India Pvt. Limited, 2007.
- [5] Daniel Beaglehole, Mikhail Belkin, and Parthe Pandit. On the inconsistency of kernel ridgeless regression in fixed dimensions, 2023.
- [6] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, jun 1998.
- [7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.
- [8] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, 2000.
- [9] Bissan Ghaddar and Joe Naoum-Sawaya. High dimensional data classification and feature selection using support vector machines. *European Journal of Operational Research*, 265(3):993–1004, 2018.
- [10] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on neural networks*, 13(2):415—425, 2002.
- [11] Ian Jolliffe. *Principal component analysis*. Springer Verlag, New York, 2002.
- [12] D.C. Lay. *Linear Algebra and Its Applications*. Addison-Wesley, 2012.

Bibliography

- [13] T. Leen, T. Dietterich, Volker Tresp, Christopher Williams, and Matthias Seeger. Using the nystroem method to speed up kernel machines. 01 2000.
- [14] James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A*, 209:415–446, 1909.
- [15] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.
- [16] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.
- [17] W. Rudin. *Fourier Analysis on Groups*. Wiley Classics Library. Wiley, 1991.
- [18] Alon Schclar and Lior Rokach. Random projection ensemble classifiers. volume 24, pages 309–316, 05 2009.
- [19] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, 2002.
- [20] S.S. Vempala. *The Random Projection Method*. DIMACS Series. American Mathematical Soc.
- [21] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*, volume 14. 06 2012.

A

Appendix

A.1. Plots for Fashion-MNIST

The plots below are from the same experiment as the plots in the section approximating kernel SVM 5.3. Here we used Fashion-MNIST instead of MNIST. The results are very similar to the results above. We also applied max-pooling to Fashion-MNIST. An example can be seen in figure A.1.

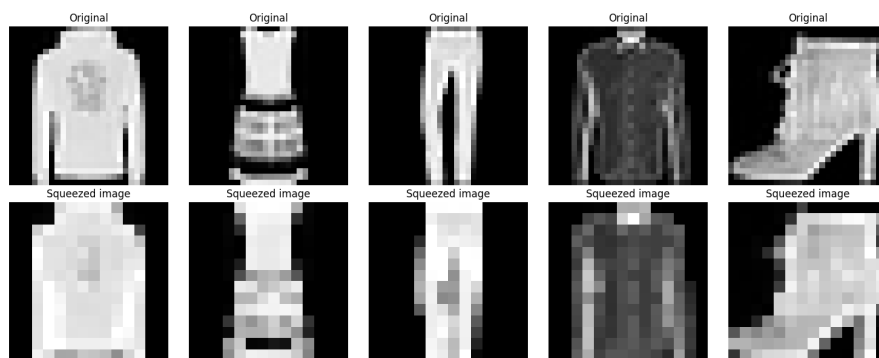


Figure A.1.: Max pooling operation on Fashion-MNIST data points. On the top are the original images. On the bottom are the compressed images.

A. Appendix

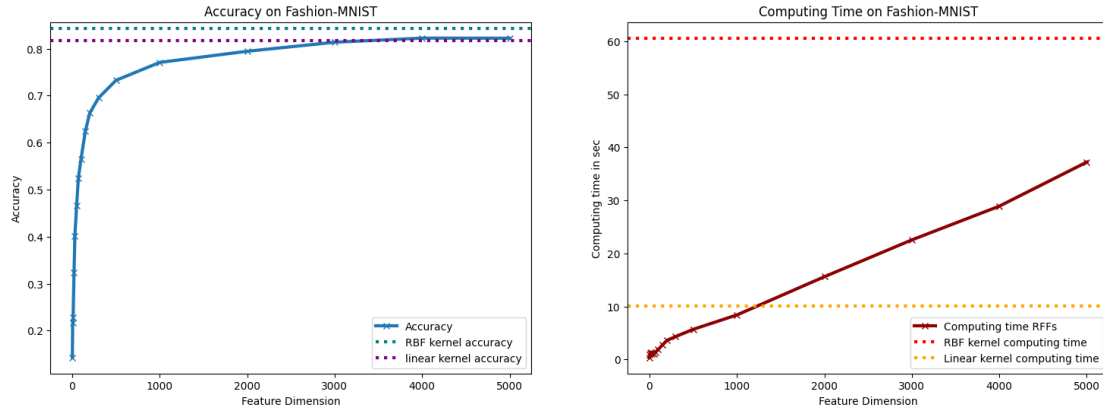


Figure A.2.: SVM trained on Feature Space with was induced by Random Fourier Features. The left plot shows the accuracy with respect to the amount of Random Fourier Features. The right plot shows the amount of time needed to train the models also with respect to the amount of Random Fourier Features. Accuracy and execution time of SVM with RBF-kernel and linear kernel can also be observed.

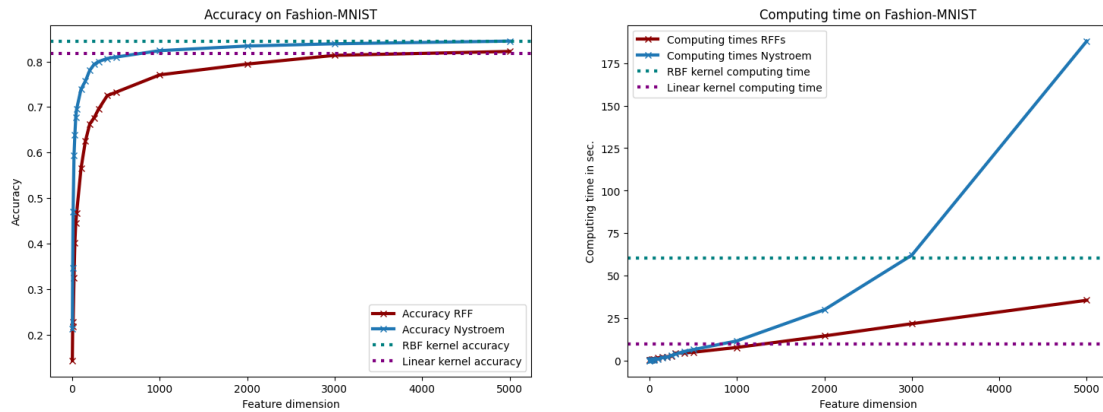


Figure A.3.: Test-accuracy and execution time with respect to the dimension of the feature space. The blue line represents a SVM trained with Nyström Features and the red line represents SVM trained with Random Fourier Features. Accuracy of RBF-kernel SVM and linear SVM can also be observed.

A. Appendix

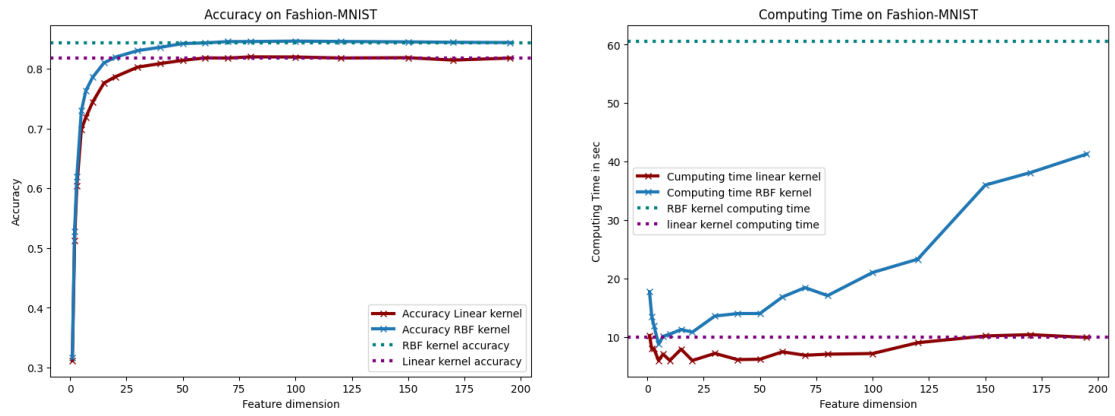


Figure A.4.: Accuracy with respect to the input dimension for a RBF-kernel SVM (blue) and a linear-kernel SVM (red). The dimensional reduction was done with help of PCA.

The computing time with respect to the feature dimension can be seen in the right picture. The dotted lines indicate the performance without dimensional reduction.

A.2. Ensemble Learning

We also made an experiment on even weaker classifier. In figure 5.4 we used more than 300 RFFs, however we only used up to 25 classifier. In figure A.2 we used up to 100 classifier to make the combined voting.

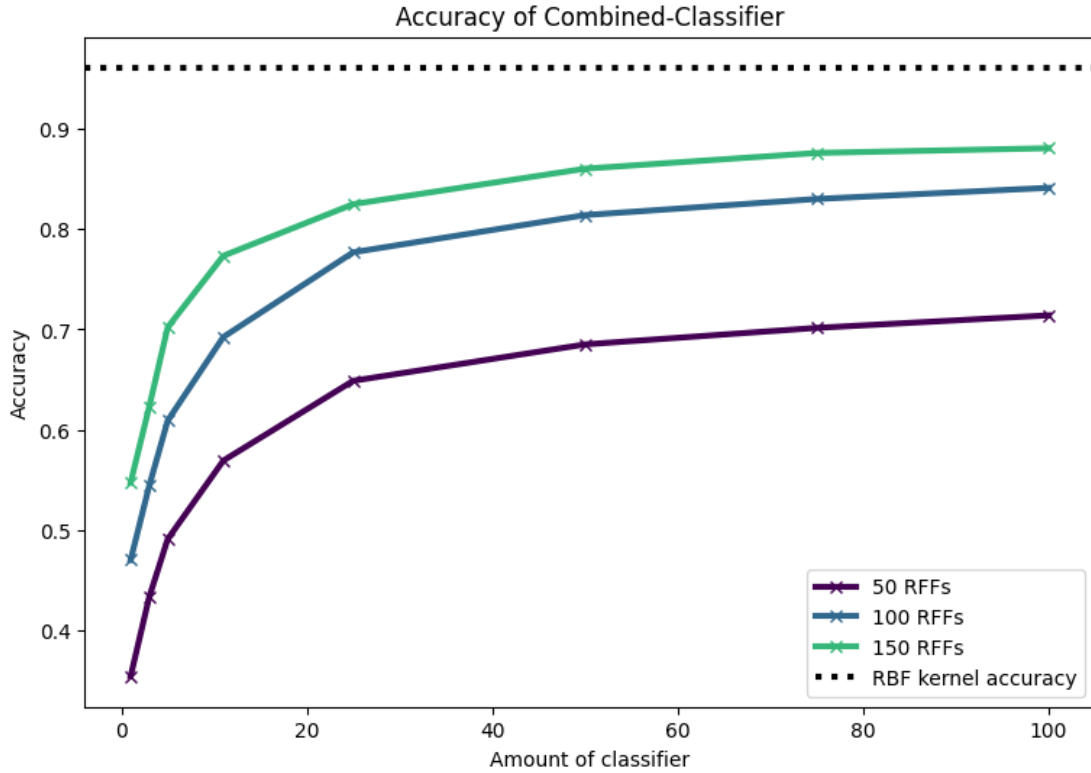


Figure A.5.: Accuracy with respect to the amount of voters in the combined classifier. We evaluate the accuracy for 50, 100 and 150 RFFs. The dotted line indicates the accuracy of a single RBF-kernel SVM

List of Figures

2.1. Original data was sampled out of a multivariate gaussian distribution. The black lines indicate the the principal components of the sampled data. By taking the components with highest variance we can project our data in a lower dimensional space.	4
2.2. Reconstructed image. On the left side the original image can be seen. The other images where reconstructed with help of the singular value decomposition. For the second image 500, third image 200, fourth image 50 singular-values where used. The image was taken from [2].	6
2.3. SVM linear dividing the space in to two regions to classify new data. The black line is the best possible line resulting out of a SVM. The red line also separates data. The margins of the red line however would be smaller than the margins of the black line.	8
3.1. original data	13
3.2. projected data	13
3.3. Data was classified with the RBF-kernel SVM (left). Data was classified with polynomial kernel (right).	16
3.4. Data was sampled out of a sin wave and then transformed with the RBF-kernel. The linear regression was then implicitly performed in the feature space.	18
4.1. Regression with original kernel. No regularization was used. Data was sampled out of a sin wave with additional gaussian noise.	22
4.2. Regression with approximated kernels. To approximate the kernel 1,5, 10, 100000 Random Fourier Features where used.	22
5.1. Max pooling operation on MNIST data points. On the top are the original images. On the bottom are the compressed images.	28

List of Figures

- 5.2. This figure shows the approximated kernel regression. The datasets avocado pricing (left) wine-quality (right) are used. The x-axis indicates the amount of RFFs used and the y-axis represents the MSE. The dotted lines represent time and MSE of RBF-kernel regression 29
- 5.3. SVM trained on feature space with was induced by RFFs. The left plot shows the accuracy with respect to the amount of RFFs. The right plot shows the amount of time needed to train the models also with respect to the amount of RFFs. Accuracy and execution time of SVM with RBF-kernel and linear kernel can also be observed. 30
- 5.4. Test-accuracy and execution time with respect to the dimension of the feature space. The blue line represents a SVM trained with Nyström Features and the red line represents SVM trained with Random Fourier Features. Accuracy of RBF-kernel SVM and linear SVM can also be observed. 31
- 5.5. Accuracy with respect to the input dimension for a RBF-kernel SVM (blue) and a linear kernel SVM (red). The dimensional reduction was done with help of PCA.
The computing time with respect to the feature dimension can be seen in the right picture. The dotted lines indicate the performance without dimensional reduction. 32
- 5.6. Data points with noise $p \in \{0, 0.1, 0.2, 0.5\}$ 32
- 5.7. In the left image we see the impact of noise to SVM with approximated RBF-kernel. The x axis indicates the feature dimension of the feature space, where the y axis measures the accuracy of the models. The experiment was executed six times with different amount of noise.
In the right image we see the impact of noise to kernel SVM. The data is projected into lower dimensional space with help of PCA. The x axis indicates the dimension after projection. 33

List of Figures

5.8. Accuracy and computing time of the majority vote of 13 classifiers (red). Average accuracy and computing time of a single classifier (blue). The dotted line indicates accuracy and computing time of a single RBF-kernel SVM.	34
5.9. Accuracy with respect to the amount of voters in the combined classifier. We evaluate the accuracy for 200, 300 and 500 RFFs. The dotted line indicates the accuracy of a single RBF-kernel SVM.	35
A.1. Max pooling operation on Fashion-MNIST data points. On the top are the original images. On the bottom are the compressed images.	42
A.2. SVM trained on Feature Space with was induced by Random Fourier Features. The left plot shows the accuracy with respect to the amount of Random Fourier Features. The right plot shows the amount of time needed to train the models also with respect to the amount of Random Fourier Features. Accuracy and execution time of SVM with RBF-kernel and linear kernel can also be observed.	43
A.3. Test-accuracy and execution time with respect to the dimension of the feature space. The blue line represents a SVM trained with Nyström Features and the red line represents SVM trained with Random Fourier Features. Accuracy of RBF-kernel SVM and linear SVM can also be observed.	43
A.4. Accuracy with respect to the input dimension for a RBF-kernel SVM(blue) and a linear-kernel SVM (red). The dimensional reduction was done with help of PCA. The computing time with respect to the feature dimension can be seen in the right picture. The dotted lines indicate the performance without dimensional reduction.	44
A.5. Accuracy with respect to the amount of voters in the combined classifier. We evaluate the accuracy for 50, 100 and 150 RFFs. The dotted line indicates the accuracy of a single RBF-kernel SVM	45

List of Tables

5.1. Dataset Avocado and Wine Quality for Regression	27
5.2. Dataset MNIST and Fashion-MNIST for Classification	27

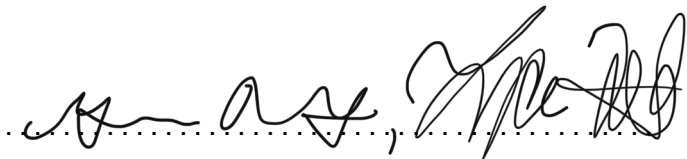
Name: Mathis Rost , Tanja Zast

Matriculation number: 945661 , 935593

Honesty disclaimer

I hereby affirm that I wrote this thesis independently and that I did not use any other sources or tools than the ones specified.

Ulm, 22.07.2023...

A handwritten signature in black ink, consisting of two distinct parts separated by a comma. The first part is a stylized, cursive signature, and the second part is a more complex, bold signature. The signature is written over a horizontal dotted line.

Mathis Rost , Tanja Zast