# Kernel Methods: Data Challenge

Moritz HALM, Fynn BACHMANN

February 18, 2021

## 1 Introduction

In this Data Challenge we are given 6000 biological DNA sequences with their respective labels whether they bind a certain protein or not. The objective is to learn a model that generalizes well, i.e. predicts the labels of 3000 test sequences well. To achieve a 3rd-place with 72% accuracy we implemented a Support Vector Machine (SVM) as well as various kernels of which the Mismatch Kernel proved to be best.

The structure of our report reflects our approach to the challenge. First we set up the working space, define ways to estimate test accuracies and explain ways to tackle the computational difficulties (Section 1.1). In the next section (2) we explore certain kernels that are able to digest DNA sequences. In Section 3 we describe how we implemented our SVM and how it compares to an official package like scipy.sklearn. Then we finetune our final model and show the results we get on the leaderbord.

The code to reproduce our results can be found on GitHub[1].

### 1.1 Methodology

To evaluate kernels without having to upload the predictions to Kaggle we use 10-fold cross validation. This can be done efficiently on a given kernel matrix by dividing the matrix in training and evaluation blocks. So only one kernel matrix computation is necessary to get an arbitrary amount of cross validations for this specific kernel and its hyperparameters. As a quantity that corresponds to the expected accuracy on the test data we define the score function

$$\text{ACC} := \frac{\hat{r}_0 + \hat{r}_1 + \hat{r}_2}{3} \tag{1}$$

where $\hat{r}_i$ is the maximum accuracy the i-th dataset achieved in 10-fold cross-validation.

It is also worth mentioning that we cache the kernel matrices and vector representations on hard disk as most of the computation time is due to kernel computations. This requires some storage capacity but allows for a more versatile fine tuning.

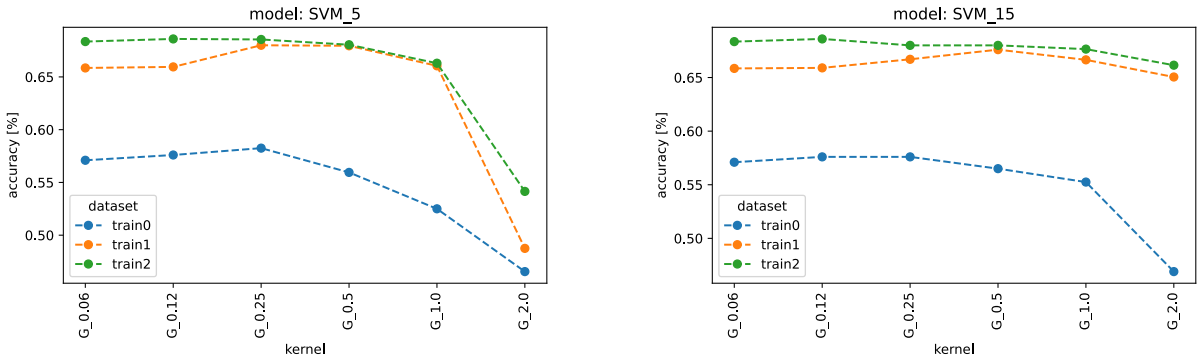
## 2 Kernels

We give here an overview of the different kernels we examined and show the respective cross validation plots to estimate their performance. From the next section on we will focus on the Mismatch Kernel only, which had the most promising results.

### 2.1 Gaussian Kernel

For the first model, the Gaussian Kernel SVM, we used the bag-of-words vector provided with the data. For efficient computation we implemented a vectorized kernel function which exploits the efficient matrix multiplication methods of numpy. A quick analysis of the performance for different $\sigma$ and $C$ yields the results shown in Figure 1.

[1] `https://github.com/fsvbach/KernelMethods.git`




(a) SVM with C=5

(b) SVM with C=15

Figure 1: Accuracy of the Gaussian kernel for different standard deviations $\sigma$ indicated by the x-axis on all datasets.

We observe in both Figure 1a and 1b that the first dataset train0 performs worse that the other two. In average we expect a test accuracy of

$$\text{ACC}_{\text{gauss}} = \frac{0.66 + 0.66 + 0.59}{3} \approx 0.637 \tag{2}$$

which is exactly the score we obtained on the public leaderboard for this particular model ($\sigma = 0.25$ and $C = 5$).

## 2.2 Weighted Degree Kernel

As a next kernel we studied the *weighted degree kernel*, which was proposed by Sonnenburg, Rätsch, and Rieck for genomice sequence analysis tasks [5]. The kernel function sums up the number of identical substrings of length $k$ (kmers) that occur at the same position in both sequences. These results are then weighted for different values of $k$ with weights $\beta_k$. Formally, the kernel for two sequences $x, x'$ is given by

$$\kappa(x, x') = \sum_{k=1}^{d} \beta_k \sum_{i=1}^{l-k+1} \mathbb{1}_{(x_i, \ldots x_{i+k}) = (x'_i, \ldots x'_{i+k})} \tag{3}$$

In order to find values for the weights $\beta_k$, we studied the accuracy for single values for $k$, i.e.

$$\beta'_k = \begin{cases} 1 & \text{if } k' = k \\ 0 & \text{else} \end{cases}$$

The results are shown in Figure 2. It is evident that small values for $k$ (i.e. $k = 1, 2$) exhibit a poor performance since matches are quite probable given the little alphabet size. On the other hand, especially for the training sets 0 and 2 the accuracy drops as well. This may be due to many sequencing errors, which make the occurrence of matching kmers unlikely for large values of $k$.

In fact, we observed that the kernel achieves best performance when putting all weight on $k = 6$ rather than using more complex weight functions as suggested by the authors. The main drawback of the weighted degree kernel is its focus on fixed string positions. For instance, two identical sequences, shifted by 1 letter would be assigned a poor score.

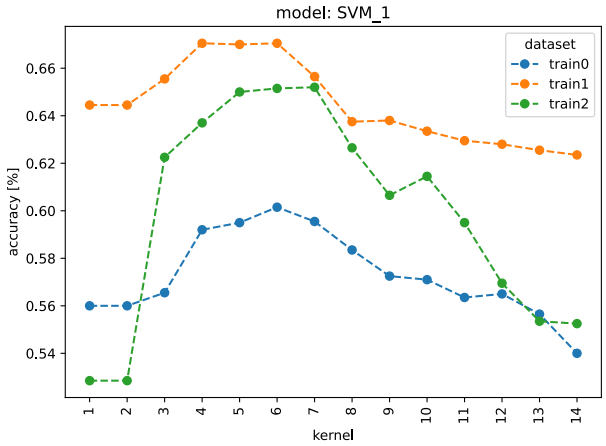



Figure 2: Accuracy for the weighted degree kernel for different kmers lengths $k$ obtained by cross-validation over the training data set. The x-axis denotes the index $k$ of the single non-zero weight $\beta_k$. We used a SVM with $C = 1$.

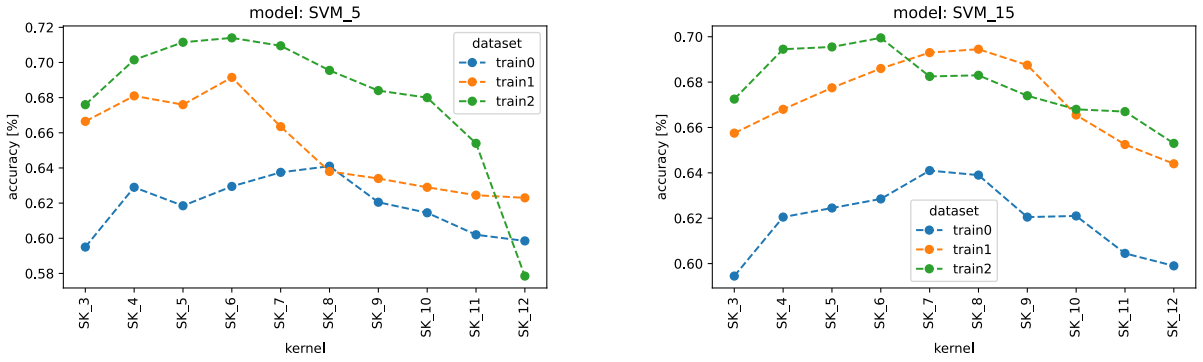
**Implementation Notes** The weighted degree kernel can be implemented efficiently by encoding kmers as integers and using bitwise shifts and comparisons. This results in a constant runtime for computing $\mathbb{1}_{(x_i,\dots x_{i+k})=(x'_i,\dots x'_{i+k})}$. We further improved performance by implementing the kernel in C++ and calling it using ctypes[2].

## 2.3 Spectrum Kernel

While the weighted degree kernel compared kmers at fixed positions, the spectrum kernel compares occuring kmers completly independent of their positions in the sequence [2]. More precisely, the $k$-spectrum $\Phi(x)$ of a sequence $x$ is a vector in $\mathbb{R}^{|\mathcal{A}|^k}$ where each entry denotes the number of occurrences of a certain kmer in $x$. The kernel is then given by the standard scalar product of the spectra.

$$\kappa_k^{\text{spec}}(x, x') = \Phi(x) \cdot \Phi(x') \tag{4}$$

We precompute the spectrum of each vector. Note that $\Phi(x) \cdot \Phi(x')$ can be computed in $\mathcal{O}(|x| + |x'|)$ using sparse vector representations, e.g. in `scipy.sparse`.




(a) SVM with C=5

(b) SVM with C=15

Figure 3: Accuracy of the spectrum kernel for different values of $k$ (x-Axis).

[2] `https://docs.python.org/3/library/ctypes.html`

Figure 3 depicts the accuracy of the spectrum kernel for different values of $k$. We observe a similar behaviour as for the weighted degree kernel (Section 2.2): While cooccurence of short kmers is relatively arbitrary and does not provide much information about the similarity of two sequences, large kmers are too rare – presumably due to sequencing errors – to be useful. Under this assumption we might deduce, that dataset 0 contains less sequencing errors than the other two datasets, since the peak of the accuracy curve is more shifted towards higher values of $k$.

Overall, the spectrum kernel outperforms the weighted degree kernel with an accuracy of

$$\mathrm{ACC}_{\mathrm{spec}} = \frac{0.71 + 0.69 + 0.64}{3} \approx 0.68 \tag{5}$$

This indicates that comparing kmers independent of their positions is beneficial.

## 2.4 Mismatch Kernel

The mismatch kernel (proposed by Leslie et al. in [3]) is an extension of the spectrum kernel that also matches kmers that are identical expect for up to $m$ letters. This method take into account falsely read nucleotides that occur regularly during DNA sequencing.

We define the $(k,m)$-spectrum $\Phi_{(k,m)} : \mathcal{A}^n \to \mathbb{R}^{|\mathcal{A}|^k}$ of a sequence $x$. Every dimension in the feature space $\mathbb{R}^{|\mathcal{A}|^k}$ represents a kmer $u_i \in \mathcal{A}^k$. We define:

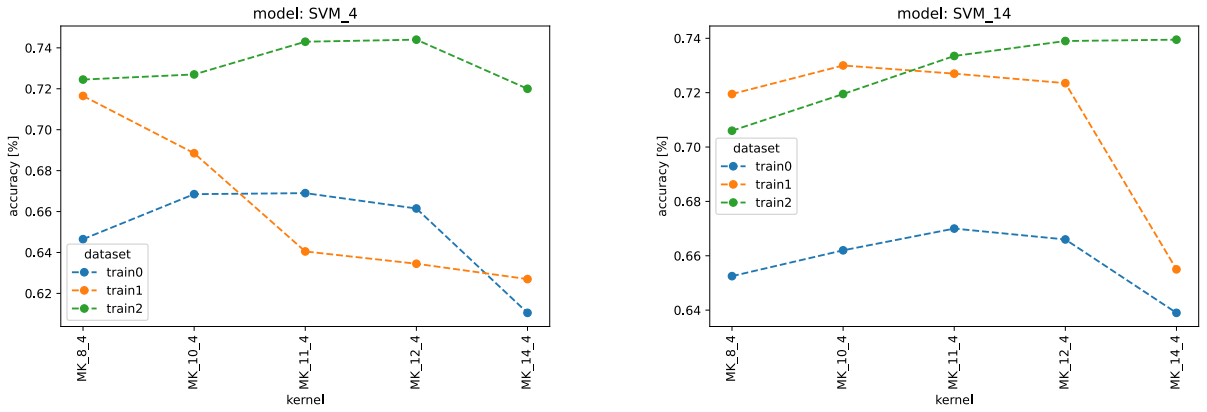
$$(\Phi_{(k,m)}(x))_i = \sum_{j=1}^{n-k+1} \sum_{m'=0}^{m} (x_j, ..., x_{j+k}) =_{m'} u_i \tag{6}$$

where $u =_{m'} v$ ,iff $u$ and $v$ are identical expect for exactly $m'$ letters. Note that our definition slightly differs from the one presented in [3], because we sum up the number of matching kmers over all $m'$ $(0 \leq m' \leq m)$, whereas use logical disjunction.

For every kmer $v$ in $x$, we have $\sum_{m'=0}^{m} \binom{k}{m'} (|\mathcal{A}| - 1)^{m'}$ kmers $u \in \mathcal{A}^k$ such that $u =_{m'} v$. The density of the feature space vectors $\Phi_{(k,m)}(x)$ thus grows exponentially in $m$. Since we store the vectors explicitly using a sparse vector representation, we had to restrict ourselves less or equal to $m = 2$. Again deviating from [3], we define the $(k, 2m)$-*mismatch kernel* for two sequences $x, x'$ as the following inner product:

$$\kappa^{\mathrm{mismatch}}_{(k,2m)}(x, x') = \Phi_{(k,m)}(x) \cdot \Phi_{(k,m)}(x') \tag{7}$$

The intuition behind our definition is the following. Suppose there are kmers $u$ in $x$ and $u'$ in $x'$ that differ in exactly $2m$ letters. Then there exists a kmer $w''$ such that the corresponding entries are non-zero in both feature vectors $\Phi_{(k,m)}(x)$ and $\Phi_{(k,m)}(x')$. Hence the $2m$-mismatch is taken into account on our kernel function.




(a) Mismatch Kernel with SVM C=4

(b) Mismatch Kernel with SVM C=14

Figure 4: Accuracies for differnet $(k, 2m)$-mismatch kernels. A label on the x-axis MK_a_b indicates a mismatch kernel with $k = a$ and $2m = b$.

Figure 4 shows the results of the mismatch kernel for different combinations of $k$ and $2m$. Previous experiments showed that increasing $m$ generally improved the accuracy of the mismatch-kernel, which is why we only plot the accuracies for $2m = 4$ here. By comparison with the spectrum kernel (Equation 5 we conclude that allowing for mismatches indeed increases the accuracy of our predictions:

$$\text{ACC}_{\text{mismatch}} = \frac{0.74 + 0.72 + 0.67}{3} \approx 0.71 \tag{8}$$

# 3 SVM

While we started using the SVM method from sklearn in the previous experiments, we now replace this by our own implementation.

## 3.1 Theory and Implementation

The objective function of an SVM can be written as the following minimization problem (following the lecture slides [4, 147 ff.]), given a kernel matrix $[\mathbf{K}]_{ij} = \kappa(x_i, x_j)$ for training data $\mathbf{x}$ and labels $\mathbf{y}$:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ q(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{y} \right\} \quad \text{s.t.} \quad \forall i, \quad 0 \leq y_i \alpha_i \leq C \tag{9}$$

By the representer theorem this yields the classificator $\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \hat{\alpha}_i K(\mathbf{x}_i, \mathbf{x})$.

Equation 9 is a quadratic problem and can be solved efficiently. Our method `fit` in the `models.our_SVM` class takes a kernel matrix $\mathbf{K}$ and a label vector $\mathbf{y} \in (-1, 1)$ an input. We use the package cvxopt[3] to minimize the quadratic program yielding the unique solution. The method `predict` then applies the classificator.

## 3.2 Comparison to scipy.sklearn

Comparing our solution with an official package with respect to their resulting accuracy in cross validation leads to Figure 5.

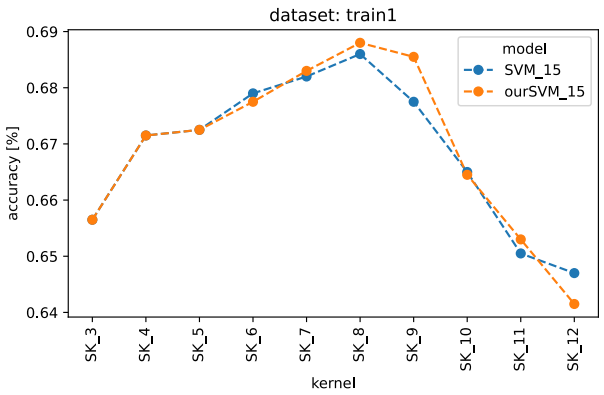


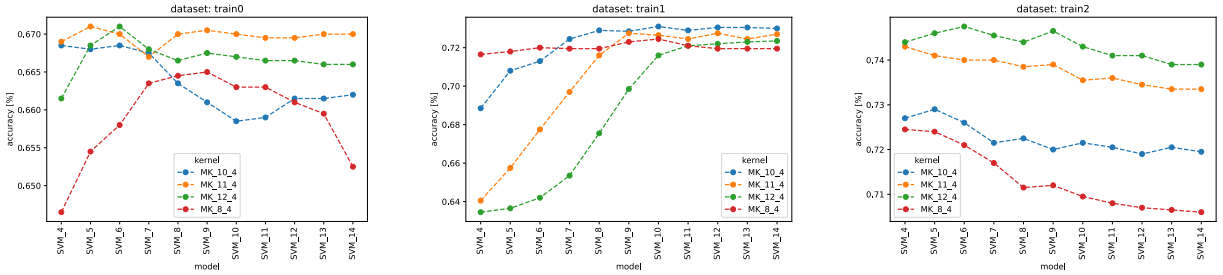

Figure 5: Comparison between our SVM and SKlearn with different kernel functions (spectrum kernel with $k \in (3, 4, ..., 12)$ and an example value $C = 15$)

We can see here, that within small fluctuations the results are equivalent. However, in comparison the SVM from sklearn is about an order of magnitude faster. This is mainly due to the optimization package they use. So in order to save time in future experiments we left both versions in the final submission. Note that they work completely independent of each other and that our final predictions were executed with our SVM, as indicated by the model name in Figure 7 in section 5.

[3] https://cvxopt.org/examples/tutorial/qp.html

# 4 Hyperparameter Finetuning

Having decided for the Mismatch kernel we now need to find the right hyperparameter $k$ and $m$ and moreover the model's hyperparameter $C$ for the SVM. For this query we plot the most promising kernel functions with $m = 4$ and $k \in (8, 10, 11, 12)$ for each dataset separately in Figure 6.




(a) Dataset 0 (b) Dataset 1 (c) Dataset 2

Figure 6: Accuracy of the classification using the Mismatch kernel in different parameter configurations over a range of $C$-values for the SVM (x-axis). MK_a_b indicates a mismatch kernel with $k = a$ and $2m = b$.

The resulting cross validation indicates that different datasets behave differently with respect to the hyperparameters. The dataset train2 prefers $k = 12$ and $C = 6$, while train1 is pretty much indifferent with respect to $k$ as long as $C \geq 11$ is sufficiently high. For the first dataset train0 it is difficult to choose the hyperparameters, since the curves are somewhat unintuitive. A promising candidate could be $k = 11$ with $C = 9$. Altogether these hyperparameters lead then to an approximate accuracy of

$$\mathrm{ACC}_{\mathrm{opt}} = \frac{0.75 + 0.73 + 0.67}{3} \approx 0.715 \tag{10}$$

# 5 Results

In the previous section we discussed the effect of different hyperparameters on the accuracy after cross validation. However when uploading the predictions of different configurations we noticed that the score on the test data does not necessarily coincide with the expected score after cross validation. For example, the mismatch kernel with $k = 10, m = 4$ yielded the best results on the public leaderbord with an accuracy of

$$\mathrm{ACC}_{\mathrm{Public}} = 0.7166 \tag{11}$$

while the proposed choice of the previous section yielded one percent less. Prior to the publication of the private leaderbord we assumed that this phenomenon was due to the randomness in cross validation and the resulting overfitting to the highest C that we select – that we overestimate the true accuracy when doing the exhaustive cross validation. We thus decided to stay with the less expected but better performing configuration in Figure 7 which eventually gave us the third-place in the ranking with a score of

$$\mathrm{ACC}_{\mathrm{Private}} = 0.72 \tag{12}$$

We have to note however, that our preferred option in the previous section ($k = 11, 2m = 4, C = 9$) would have predicted even 72.5% of the test data correctly. Having such a large variance between the first half and the second half of the test data seemed suspicious to us. Now we learned that this is possible after all and that we should have trusted our hyperparameter optimization.

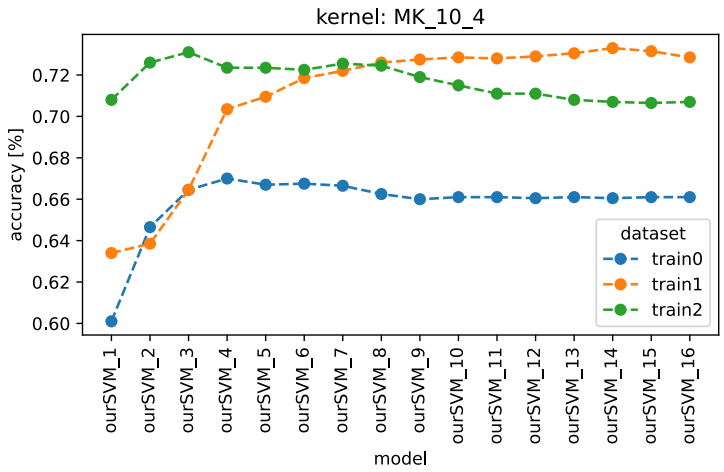



Figure 7: Cross validation for Final Kernel. All lines corerspond to the Mismatch kernel with $k = 10$ and $2m = 4$. The model name on the x-axis indicates that this experiment was done with our SVM implementation.

### 5.1 Outlook

While our proposed analysis resulted in a 3rd place of the challenge, we could still improve the method by extending our hyperparameter optimization to higher k and m. For computational reason we had to stop at $k = 14$ and $2m = 4$, however we could theoretically extend this to $k = 16$ with $2m = 6$. To enable the computation for higher values of $k$ and $m$ one could implement a mismatch trie [3] or approximate the kernel matrix with a method proposed by Blakely et al.[1].

Another improvement that we could think of was to combine different kernels. A weighted sum of the mismatch kernel and another, position-based kernel could lead to even better results. However, studying and optimizing such a model is very tedious due to the high number of hyperparameters.

## References

[1] Derrick Blakely et al. "FastSK: fast sequence analysis with gapped string kernels". In: *Bioinformatics* 36.Supplement_2 (2020), pp. i857–i865.

[2] Christina Leslie, Eleazar Eskin, and William Noble. "The Spectrum Kernel: A String Kernel for SVM Protein Classification". In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* 7 (Feb. 2002), pp. 564–75. DOI: 10.1142/9789812799623_0053.

[3] Christina Leslie et al. "Mismatch String Kernels for SVM Protein Classification". In: vol. 20. Jan. 2002, pp. 1417–1424.

[4] Julien Mairal and Jean-Philippe Vert. *Machine Learning with Kernel Methods*. Jan. 2020.

[5] Sören Sonnenburg, Gunnar Rätsch, and Konrad Rieck. "Large Scale Learning with String Kernels". In: Jan. 2007, pp. 73–104.