Multiclass Adaptive Subspace Learning

Peter Preinesberger[1], Maximilian Münch[1,2,3] and Frank-Michael Schleif[1] *

1- Techn. Univ. of Appl. Sc. Würzburg-Schweinfurt, Würzburg, Germany
2- Dept. of CS, University of Groningen, Groningen, Netherlands
3- Center for Artificial Intelligence and Robotics, Würzburg, Germany

Abstract. In modern data analysis, there is an increasing trend towards the integration of information across diverse input formats and perspectives. If the available data is not given in large quantities deep learning is in general impractical. The recently introduced Adaptive Subspace Kernel Fusion (ASKF) technique provides an efficient solution for binary classification, facilitating the effective integration of diverse views throughout the learning process. In this paper, we extend ASKF by employing a vector-labeled multi-class model, eliminating the need for multiple individual models typically required in conventional one-vs-rest or one-vs-one approaches. We also evaluated the effect of using GPU-based numerical solvers, optimizing our problem formulation and the generated code for better efficiency. The approach is evaluated on various kernel functions, highlighting our methods ability of robustly dealing with multi-view data.

1 Introduction

In recent years, machine learning has become a pivotal component of various industries, including healthcare [1], multimedia [2], and environmental sciences [3]. These domains often derive heterogeneous data from multiple sources, capturing objects across multiple modalities from different perspectives. Learning classifiers from such heterogeneous data poses significant challenges due to the data being represented by differing structures, such as text, graphs, and time series [4]. Whereas most machine learning challenges today are tackled using deep learning [5], this approach is limited by the need for large amounts of data across the different modalities. Considering this restriction, proximity-based learning offers an effective alternative, having shown notable results for non-vectorial data compared to deep learning embeddings [6, 4, 7].

The recently proposed Adaptive Subspace Kernel Fusion (ASKF) [8] additionally leverages the spectral properties of multiple proximity functions to create a information-rich representation across multiple modalities, which has shown promising results for binary classification tasks. As a main contribution we propose extending ASKF using a vector-labeled multi-class strategy to reduce model complexity and enhance efficiency. Additionally we demonstrate the suitability of our approach for GPU-enabled numerical solvers. Our experimental evaluation, conducted on simulated and benchmark datasets with diverse kernel matrix parameterizations that represent distinct data modalities, demonstrates the efficacy of our method in a controlled setting.

2 Background

Consider a collection of N objects $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ in an input space $\mathcal{X}$. Each object $\mathbf{x}_i$ is described in M different modalities For each modality m, we assume a proximity

*peter.preinesberger@study.thws.de, {maximilian.muench,frank-michael.schleif}@thws.de function k^m which processes pairs of points $\mathbf{x}_i^m$ and $\mathbf{x}_j^m$. These proximity functions measure either similarity or dissimilarity, constructing respective matrices.

Kernel methods leverage the reproducing kernel Hilbert space (RKHS), induced by a positive semi-definite (PSD) kernel function [9]. Each kernel function $k^m : \mathcal{X}^m \times \mathcal{X}^m \to \mathbb{R}$ is defined as $k^m(\mathbf{x}_i^m, \mathbf{x}_j^m) = \langle \phi(\mathbf{x}_i^m), \phi(\mathbf{x}_j^m) \rangle_{\mathcal{H}}$, where $\phi : \mathcal{X} \to \mathcal{H}$ maps the input data into a high-dimensional Hilbert space $\mathcal{H}$. The resulting kernel matrix $\mathbf{K}^m$ has entries $K_{i,j}^m = k^m(\mathbf{x}_i^m, \mathbf{x}_j^m)$. Due to the potential variance in the range of the scores calculated by different kernel functions, it is crucial to normalize the kernel matrices: $K_{i,j}^m = \frac{k^m(\mathbf{x}_i^m, \mathbf{x}_j^m)}{\sqrt{k^m(\mathbf{x}_i^m, \mathbf{x}_i^m) \cdot k^m(\mathbf{x}_j^m, \mathbf{x}_j^m)}}$. As noted in [10], certain kernel functions become non-positive semi-definite (non-psd) as a result of this normalization. Non-psd kernel functions are more prevalent than anticipated and require appropriate attention [10, 11]. [1]

2.1 Learning with Multiple Kernel Functions

Learning classification models from multi-modal, heterogeneous data is complex as it requires integrating information from various sources to achieve good generalization.

By combining multiple base kernels into a single, information-rich kernel matrix, MKL in the past has shown great performance for processing and integrating diverse data representations[13, 14, 15]. However, traditional MKL methods often rely on linear combinations of kernels, require the solving of expensive optimization problems and demand psd kernel functions. ASKF addressed these limitations by handling non-PSD input kernel matrices and leveraging spectral properties to uncover hidden information not accessible by a linear weighting of kernels.

3 Methodology

3.1 Adaptive Subspace Kernel Fusion

For all kernel matrices, ASKF makes use of an eigen-decomposition: $\mathbf{K}^m = \mathbf{Q}_m \mathbf{\Lambda}_m \mathbf{Q}_m^T$, with $\mathbf{\Lambda}_m$ containing the eigenvalues and $\mathbf{Q}_m$ the corresponding eigenvectors of $\mathbf{K}^m$. ASKF makes use of the top t dominating (absolute) eigenvalues and stores their respective eigenvectors as $\tilde{\mathbf{Q}}$ as well as their original (potentially negative) eigenvalues as $\tilde{\mathbf{\Lambda}}$. For further details and an in-depth analysis of ASKF, we refer the reader to [8]. In ASKF-SVM the eigenvectors $\tilde{\mathbf{Q}}$ of the selected eigenvalues are used, but do not retain the original eigenvalues (and not even parts of them). Instead, the eigenvalues are treated as learning parameters integrated into an optimization problem (see Eq. 1). Therefore, we extend the classical SVM by additional regularization terms and constraints:

$$
\begin{aligned}
\min_{\alpha, \tilde{\Lambda}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{Y} \underbrace{\tilde{\mathbf{Q}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{Q}}^T}_{\tilde{\mathbf{K}}} \mathbf{Y}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} - \underbrace{\beta \cdot \mathbf{1}^T \tilde{\mathbf{\Lambda}} \mathbf{1}}_{R_1} + \underbrace{\gamma \cdot ||\tilde{\mathbf{Q}}\mathbf{\Lambda}\tilde{\mathbf{Q}}^T - \tilde{\mathbf{Q}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{Q}}^T||_*}_{R_2} \\
\text{subject to} \quad & 0 \le \alpha_i \le C \quad \text{and} \quad \sum_i y_i \alpha_i = 0, \qquad i = 1, \ldots, N; \quad C \in \mathbb{R} \\
& \underbrace{\tilde{\lambda}_i \ge 0}_{C_1} \quad \forall \tilde{\lambda}_i \in \tilde{\Lambda} \quad \text{and} \quad \underbrace{\sum_i |\tilde{\lambda}_i| \le \delta \cdot \sum_i |\lambda_i|}_{C_2},
\end{aligned}
\tag{1}
$$

[1]As outlined in [4, 12] also most domain specific proximity measures that are employed in real-world data analysis tasks lead to a non-psd setting.

with class labels $\mathbf{Y} = \mathrm{diag}(y_1, \ldots, y_N)$. In contrast to the classical SVM, $\mathbf{K}$ is replaced by a new kernel matrix $\tilde{\mathbf{K}} = \tilde{\mathbf{Q}}\tilde{\Lambda}\tilde{\mathbf{Q}}^T$. The additional constraints C_1 and C_2 and the new regularisation terms R_1 and R_2 ensure that the new eigenvalues $\tilde{\lambda}_i$ are of appropriate size. After optimization, one obtains the optimized new eigenvalues for $\tilde{\Lambda}$ to construct the fused data representation $\tilde{\mathbf{K}} = \tilde{\mathbf{Q}}\tilde{\Lambda}\tilde{\mathbf{Q}}^T$ and additionally an already trained support vector classifier using optimized $\boldsymbol{\alpha}$ values to classify new unseen data points.

3.2 Vector Output SVM and ASKF

In order to extend the binary ASKF-SVM to an efficient multiclass approach, we make use of the SVM formulation with vector valued output as described in [16]. The extension to vector outputs is achieved using a matrix $\mathbf{W}$ as the linear operator instead of the normal $\mathbf{w}$ of the separating hyperplane. Transferred to the dual formulation, our problem takes the form:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T(\mathbf{K} \odot (\mathbf{Y}^T\mathbf{Y}))\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \alpha_i \geq 0, \quad \alpha_i \leq C \quad \forall i \in \{1, ..., N\} \qquad \mathbf{Y}\boldsymbol{\alpha} = \mathbf{0} \end{aligned} \tag{2}$$

Where $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_n]$ contains the label vectors of the dataset as columns, $\odot$ denotes element-wise multiplication and $\mathbf{K} \in \mathbb{R}^{N \times N}$ is the kernel matrix. The classification function is now $\arg\max_{t \in \{0,...,T-1\}} \sum_{i=1}^{N} \alpha_i \langle \hat{\mathbf{y}}_t, \mathbf{y}_i \rangle \langle \mathbf{x}_i, \mathbf{x} \rangle_\Phi - \langle \hat{\mathbf{y}}_t, \mathbf{b} \rangle$, where the components of the bias vector $\mathbf{b}$ are (analogously to the scalar bias offset of the classical SVM) given by $b_j = \sum_{n=1}^{N} \alpha_n (y_n)_j K_{ni} - (y_i)_j$, for any i with $\alpha_i > 0$ where $(y_i)_j$ denotes the jth component of the label vector of instance i with $\mathbf{b} = [b_1, ..., b_{dim(\mathcal{Y})}]^T$.

For the choice of label vectors, we follow the scheme introduced in [16]. For a classification problem with T classes, our label vectors for each class $t \in \{0, ..., T-1\}$ will be $\hat{\mathbf{y}}_t \in \mathbb{R}^T$, with $dim(\mathcal{Y}) = T$. Integrating the voSVM problem from Eq. (2) into the binary ASKF Eq. (1) yields the formulation of the vectorized ASKF-SVM (**voASKF**):

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \tilde{\Lambda}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T(\underbrace{(\tilde{\mathbf{Q}}\tilde{\Lambda}\tilde{\mathbf{Q}}^T)}_{\tilde{\mathbf{K}}} \odot (\mathbf{Y}^T\mathbf{Y}))\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} - \underbrace{\beta \cdot \mathbf{1}^T\tilde{\Lambda}\mathbf{1}}_{R_1} + \underbrace{\gamma \cdot ||\tilde{\mathbf{Q}}\Lambda\tilde{\mathbf{Q}}^T - \tilde{\mathbf{Q}}\tilde{\Lambda}\tilde{\mathbf{Q}}^T||_*}_{R_2} \\ \text{subject to} \quad & \alpha_i \geq 0, \quad \alpha_i \leq C \quad \forall i \in \{1, ..., N\} \qquad \mathbf{Y}\boldsymbol{\alpha} = \mathbf{0} \\ & \underbrace{\tilde{\lambda}_i \geq 0}_{C_1} \quad \forall \tilde{\lambda}_i \in \tilde{\Lambda} \quad \text{and} \quad \underbrace{\sum_i |\tilde{\lambda}_i| \leq \delta \cdot \sum_i |\lambda_i|}_{C_2}, \end{aligned} \tag{3}$$

4 Experiments and Results

We implemented the optimization problem of Eq. (3) using the GENO solver[17] and ran experiments using an external GPU[2]. The initial experiments investigate the scaling of training time with respect to the dataset size. We compare a One-vs-Rest wrapping

[2]https://github.com/Pedda1712/ASKFvoSVM contains our implementation of voASKF

around the binary ASKF-SVM with our **voASKF**. For this purpose, we created blob datasets of 10 classes with varying number of samples. Five RBF kernels with differing γ values served as input to the learning models. Table 1 presents the runtimes for each model training on the synthetic blob datasets, Table 2 the respective accuracies. As the dataset size increases, the training time for ASKF One-vs-Rest grows significantly, particularly for the CPU-based implementation, while voASKF (GPU) exhibits a more moderate increase in training time.

Table 1: Training **time** in seconds on blob datasets with 2 features and 10 classes.

Dataset Size	**ASKF OvR (CPU)**	**ASKF OvR (GPU)**	**voASKF (GPU)**
100	82.34 ± 11.73	271.70 ± 48.66	49.67 ± 8.34
200	170.66 ± 22.02	356.12 ± 17.93	55.48 ± 16.80
400	659.15 ± 55.86	465.84 ± 29.89	55.94 ± 3.03
600	$1{,}728.71 \pm 206.10$	610.90 ± 26.55	82.85 ± 14.26
800	$3{,}328.05 \pm 120.54$	813.28 ± 33.91	105.85 ± 20.45
1000	$5{,}218.64 \pm 346.13$	$1{,}036.35 \pm 15.38$	165.68 ± 31.57
1500	$13{,}778.79 \pm 631.05$	$2{,}641.08 \pm 111.34$	569.42 ± 161.05
2000	$26{,}355.60 \pm 1{,}134.13$	$4{,}784.90 \pm 125.53$	$1{,}072.41 \pm 256.96$

Table 2: Test **accuracy** on blob datasets with 2 features and 10 classes.

Dataset Size	**ASKF OvR (CPU)**	**ASKF OvR (GPU)**	**voASKF (GPU)**
100	0.89 ± 0.03	0.89 ± 0.03	0.86 ± 0.04
200	0.86 ± 0.03	0.86 ± 0.03	0.85 ± 0.03
400	0.91 ± 0.05	0.92 ± 0.03	0.90 ± 0.02
600	0.91 ± 0.01	0.91 ± 0.02	0.93 ± 0.02
800	0.93 ± 0.01	0.93 ± 0.02	0.93 ± 0.01
1000	0.94 ± 0.02	0.92 ± 0.01	0.91 ± 0.01
1500	0.81 ± 0.05	0.80 ± 0.05	0.89 ± 0.01
2000	0.64 ± 0.05	0.64 ± 0.03	0.83 ± 0.01

For smaller dataset sizes, all three methods demonstrate comparable performance. However, as the dataset size increases, voASKF (GPU) shows superior performance against ASKF One-vs-Rest implementations, particularly for datasets > 1000 samples. This effect arises from the cumulative effect of individual binary classifiers, which can lead to ambiguity where class distributions overlap. The performance discrepancy between the CPU and GPU OvR implementations arises from known numerical effects associated with GPU processing [3]. Not shown in Table 1 are the training times for the CPU implementation of voASKF, for which we observed roughly 6 times longer training times in this experiment, which mirrors the GPU scaling behaviour of the OvR method.

Our second experiments investigate the scaling effect of training time with respect to the number of classes. Table 3 shows the execution times and prediction accuracies on the test data. Notably, **voASKF** scales to a large number of classes without encountering execution time issues, while maintaining comparable performance.

Finally, to underscore the effectiveness of our method, we evaluated our approach on diverse datasets from the UCI machine learning repository [18] .We intentionally

[3]https://pytorch.org/docs/stable/notes/randomness.html

Table 3: Training time and test accuracy on many-class blob data (2d, 750 samples).

	Training Time (s)		Test Success Rate	
Classes	**ASKF OvR (GPU)**	**voASKF (GPU)**	**ASKF OvR (GPU)**	**voASKF (GPU)**
5	181.23 ± 17.79	121.71 ± 14.56	0.94 ± 0.01	0.92 ± 0.02
10	347.85 ± 8.21	118.03 ± 15.64	0.94 ± 0.02	0.92 ± 0.01
15	550.63 ± 22.24	111.07 ± 13.72	0.90 ± 0.02	0.88 ± 0.02
20	768.82 ± 23.38	119.54 ± 13.01	0.83 ± 0.03	0.78 ± 0.04
25	874.51 ± 28.45	113.06 ± 9.88	0.78 ± 0.03	0.74 ± 0.02
30	$1{,}085.27 \pm 20.99$	115.60 ± 8.90	0.73 ± 0.01	0.70 ± 0.03

generated indefinite *tanh*-kernels in conjunction with some psd RBF kernels. We score the dataset's definiteness by considering the union of the sets of eigenvalues of the M different kernel matrices $S = \sigma(\mathbf{K}^1) \cup \ldots \cup \sigma(\mathbf{K}^M)$. The score is then $\tau = \frac{\sum_{\lambda_{neg} \in \{s \mid s \in S, s<0\}} |\lambda_{neg}|}{\sum_{\lambda \in S} |\lambda|}$ following the experiments from [8].

Our approach was benchmarked against k-Nearest Neighbors (k-NN) with $k = 5$, a method inherently capable of handling indefinite kernels. The results of this experiment are given in Table 4. While performing comparably, a notable distinction between

Table 4: Test accuracy on some vectorial datasets with indefinite *tanh* kernels included.

Dataset	**#classes**	τ	**voASKF**	**ASKF OvR**	**kNN**
Image Segmentation	7	0.18	0.86 ± 0.03	0.85 ± 0.03	0.83 ± 0.03
Wholesale customers	3	0.19	0.71 ± 0.01	0.72 ± 0.01	0.62 ± 0.02
Vertebral Column	3	0.15	0.79 ± 0.03	0.82 ± 0.02	0.76 ± 0.03
Glass Identification	6	0.16	0.68 ± 0.04	0.70 ± 0.04	0.64 ± 0.04
Vehicle Silhouettes	4	0.18	0.74 ± 0.02	0.76 ± 0.02	0.69 ± 0.02
User Knowledge Modeling	4	0.04	0.87 ± 0.02	0.86 ± 0.02	0.80 ± 0.02

voASKF and ASKF OvR emerge in their model sparsity, given by the number of support vectors. In Table 5, we present the number of necessary support vectors which are an indicator for model complexity and sparsity.

Table 5: Support vector count for the ASKF models of Table 4

Dataset	**#classes**	**Instances**	**voASKF**	**ASKF OvR**
Image Segmentation	7	210	72.80 ± 2.04	100.60 ± 5.16
Wholesale customers	3	440	164.80 ± 6.11	219.20 ± 1.60
Vertebral Column	3	310	123.20 ± 4.17	85.40 ± 7.45
Glass Identification	6	214	83.60 ± 4.13	105.80 ± 0.75
Vehicle Silhouettes	4	845	370.80 ± 3.87	401.00 ± 3.22
User Knowledge Modeling	4	403	178.20 ± 11.51	200.80 ± 0.40

As shown, **voASKF** generally yields sparser models than ASKF OvR, improving computational efficiency and potentially enhancing generalization.

5 Conclusion

In this paper, we presented an extension of the Adaptive Subspace Kernel Fusion (ASKF) method to address the challenges of multi-class classification in multi-view data analysis. By introducing a vector-labeled multi-class strategy and leveraging GPU-enabled numerical solvers, our approach significantly enhances the scalability whilst retaining the performance of ASKF. Our approach not only circumvents the limitations of deep learning-based embeddings, which often require extensive computational resources and large datasets, but also provides a robust alternative for integrating non-psd or non-metric, often domain-specific proximity functions into the learning process. In future the approach will be evaluated on more real-life scenarios and could be potentially extended to regression tasks or ordinal classification problems. Due to the computational cost of ASKF, a rewarding avenue of research may be the use of kernel approximation schemes [19] to further reduce the cost of the base approach.

Acknowledgments The authors thank the Center for Artificial Intelligence and Robotics, Würzburg, Germany. Parts of this project where funded in the EU-EFRE project GREEN-INNO (FKZ:2404-003-1.2).

References

[1] Andreas K Triantafyllidis and Athanasios Tsanas. Applications of machine learning in Real-Life digital health interventions: Review of the literature. *J Med Internet Res*, 21(4):e12286, April 2019.

[2] Nikolaos Thomos, Thomas Maugey, and Laura Toni. Machine learning for multimedia communications. *Sensors (Basel)*, 22(3), January 2022.

[3] William W. Hsieh. Evolution of machine learning in environmental science—a perspective. *Environmental Data Science*, 1:e3, 2022.

[4] Frank-Michael Schleif and Peter Tiño. Indefinite proximity learning: A review. *Neural Computation*, 27(10):2039–2096, 2015.

[5] Shiliang Sun, Liang Mao, Ziang Dong, and Lidan Wu. *Multiview Machine Learning*. Springer, 2019.

[6] Elzbieta Pekalska and Robert P. W. Duin. *The Dissimilarity Representation for Pattern Recognition - Foundations and Applications*, volume 64 of *Series in Mach. Perc. and AI*. WorldScientific, 2005.

[7] Philipp Väth, Maximilian Münch, Christoph Raab, and F.-M. Schleif. Proval: A framework for comparison of protein sequence embeddings. *Journal of Comp. Math. and Data Science*, 3:100044, 2022.

[8] Maximilian Münch, Manuel Röder, Simon Heilig, Christoph Raab, and Frank-Michael Schleif. Static and adaptive subspace information fusion for indefinite heterogeneous proximity data. *NC*, 555:126635, 2023.

[9] Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and ML Series. MIT Press, 2002.

[10] Jeffrey Pennington, Felix X Yu, and Sanjiv Kumar. Spherical random features for polynomial kernels. In *Adv. in NIPS*, pages 1837–1845. MIT Press, 2015.

[11] Fanghui Liu, Xiaolin Huang, Yingyi Chen, and Johan Suykens. Fast Learning in Reproducing Kernel Krein Spaces via Signed Measures. In *AISTATS'2021*, pages 388–396. PMLR, 2021.

[12] Maximilian Münch, Christoph Raab, Michael Biehl, and Frank-Michael Schleif. Data-driven supervised learning for life science data. *Frontiers Appl. Math. Stat.*, 6, 2020.

[13] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *JMLR*, 12:2211–2268, 2011.

[14] Fabio Aiolli and Michele Donini. EasyMKL: A scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

[15] Alain Rakotomamonjy, Francis R. Bach, Stéphane Canu, and Yves Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9(83):2491–2521, 2008.

[16] Sandor Szedmak and John Shawe-Taylor. Multiclass learning at one-class complexity. 01 2005.

[17] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. GENO – GENeric Optimization for classical machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[18] Dheeru Dua and Casey Graff. Uci machine learning repository, 2017.

[19] Maximilian Münch, Katrin Sophie Bohnsack, Frank-Michael Schleif, and Thomas Villmann. Data-distribution-informed nyström approximation for structured data using vector quantization-based landmark determination. *Neurocomputing*, 596:128100, 2024.