New Kernel for SVMs

Nabeel Bacchus

PSID: 1584360

University of Houston

Fall 2022 - COSC 6342

1. Survey

A. Introduction

Support Vector Machines(SVM) are a supervised machine learning algorithm that can be used for estimating regression and classification problems [3]. They are able to outperform other machine learning algorithms and are used to solve various real-world problems like text categorizations, image classification, and fraud detection. SVMs are able to pick the optimal hyperplane to separate different classes in that data by finding the largest separating margin of different classes. If the data is not linearly separable, then kernels can be used to map the input space into a linearly separable feature space. The advantages of SVMs: it can handle semi-structured and structured data, it can scale up with high dimensional data, it does not get stuck in local minimum, and it is less prone to overfitting. The disadvantages of SVMs: large datasets reduce performance due to the increase in training time, it can be difficult to find the best kernel, and SVM does not perform well with noisy data [5].

Choosing the proper kernel is very important to the performance of the SVM because mapping data points to a new feature space can improve the margin of the decision boundary. More specifically, it can increase the distance between the support vectors of different classes. In this paper, we seek to create a new kernel that improves the performance of SVMs on classification problems if we make no previous assumptions on the data. We test the kernel on three different datasets and compare the performance to traditional machine learning models and SVM models that use different kernel functions.

B. Kernel Trick

The kernel functions are a crucial part to why SVMs are successful with handling non-linear data. It works well because the complexity of the optimization problem is dependent on the dimensionality of the input space and not the feature space. Thus, we can have a high dimensionality of the feature space to get a linear decision boundary without affecting the complexity of the algorithm. Choosing an appropriate kernel can cause similar data points being clustered together, resulting in data points being more linearly separable [3]. For this reason, kernel functions can be viewed as a similarity function

To show how kernels operate, assume that we have two transformed feature vectors $h(x_i)$ and $h(x_j)$. This would cause the Lagrange dual function to have the form:

$$L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle h(x_i), h(x_j) \rangle$$

Resulting in the discriminant function f(x) being written as:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle h(x_i), h(x_j) \rangle + \beta_0$$

If we were to define the kernel function [2] as

$$K(x, x') = \langle h(x), h(x') \rangle$$

The specific transformations of $h(x)$ no longer become relevant to us since we are directly calculating the dot product of the transformed data points in some feature space. We can plug the kernel into the discriminant function and get the optimal separating hyperplane:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + \beta_0$$

Where α_i is the optimal Lagrange multipliers and β_0 is the optimal perpendicular distance from the origin [7].

C. Other Kernel Functions

As discussed before, picking the appropriate kernel function is important for the performance of SVMs. Some popular kernel functions that exist are listed below [4]:

Linear kernel:

$$K(X, Y) = X^T Y$$

Polynomial kernel:

$$K(X, Y) = (\gamma X^T Y + r)^d$$

Radial Basis Function kernel:

$$K(X, Y) = exp(-\gamma ||X - Y||^2)$$

Sigmoid kernel:

$$K(X, Y) = tanh(\gamma X^T Y + r)$$

Where γ is the gamma term in the kernel function for all kernel types except linear. The d is the polynomial degree in the kernel function for the polynomial kernel. The r is the bias term in the kernel function for the polynomial and sigmoid kernel. All terms are user-controlled parameters that help improve the performance of the SVM model.

I. Multiple Kernel Learning

Multiple Kernel Learning(MKL) is a method where instead of using just one kernel function, we use multiple different kernel functions for a single model [1]. The intuition behind this is similar to why ensemble models are used. We let the algorithm choose which kernel function works best instead of the user trying to determine which kernel function or combination of kernels to use. Another reason to use MKL is because different kernels result in different mapping of the data, combining kernels makes it possible to combine different forms of data [1].

There are six properties of MKL that are used to make proper classifications: learning method, functional form, target function, training method, base learner, and computational complexity. The learning method is about the way kernels are combined together. The functional form refers to ways kernel combination can be done and the parameters involved with the combination.

Target functions are broken down into three categories: similarity-based functions, structural risk functions, and bayesian functions. Similarity-based functions calculate the similarity matrix and pick a combination function that improves the similarity. Structural risk functions use regularization terms and try to minimize complexity and error. Bayesian functions calculate the quality of the kernel using Bayesian formulation.

There are two different methods of training. The one-step method involves calculating the combination function and the parameters of the machine learning algorithm in one pass. The two-step method is an iterative approach where in each iteration, update the combination function parameters while fixing machine learning algorithm parameters; and vice versa. This repeats until parameters converge.

The base learner is the kernel based machine learning algorithm that can use MKL, the most common one being SVMs. Finally, the computational complexity is based on the training method and the complexity of the base learner.

There are many different MKL algorithms that exist, however in this paper we will only be utilizing two forms. The summation and multiplication form are displayed below, respectively [1]:

$$K_{\eta}(X,Y) = \sum_{m=1}^{P} k_m(X_m, Y_m)$$

$$K_{\eta}(X,Y) = \prod_{m=1}^{P} k_m(X_m, Y_m)$$

II. Weighted P-norm t Kernel

Weighted p-norm t kernel was created to improve the applicability and classification prediction of SVM [4]. It combines MKL and obtaining the most optimal weights for the kernel combinations. A t-class kernel is based on the t distribution probability density function. To have better flexibility and get a suitable mapping, the t-class kernel function is extended to the p-norm distance kernel. This leads to getting the optimal kernel weights and parameters for the combinations of different kernel functions. The weighted p-norm t kernel is displayed below:

$$K(X,\ Y)\ =\ \sum_{s=1}^{M} \omega_s \left(\frac{c}{1+||X-Y||_p}\right)^{v_s}$$

Where ω_s represents the weight for each kernel M, c and v_s are scale parameters where both values are greater than 0, and $||X\ -\ Y||_p$ is the p-norm distance of any two samples.

The results of the weighted p-norm t kernel shows that it can improve the classification prediction performance for SVMs compared to just a single kernel function. In order to get the best results, one must choose the best norm distance for the specific dataset being used.

III. Legendre Polynomial Kernel

The Legendre polynomial kernel is used to improve the performance of SVMs on non-linear datasets. It uses the Reproducing Kernel Hilbert Space(RHKS) to map the data. RHKS states that if two functions are close in norm, then they are pointwise close as well [6]. Additionally, RHKS is shown to work with Mercer's theorem and high power kernel theorem. RHKS allows for the Legendre polynomials to be used as a kernel function and properly map the polynomials in the feature space. The Legendre polynomial kernel is displayed below

$$K_n(X,\ Y)\ =\ \sum_{i=0}^{n} L_i(X)L_i(Y)$$

Where $L(x)$ represents the Legendre polynomials.

The results of the Legendre polynomial kernel show that it outperformed the traditional single kernel functions for SVMs. However, it takes significantly more time to build the model with the Legendre polynomial kernel than it does with the single kernel function.

2. Experiment

A. Custom Kernel

The goal of the custom kernel is to have better performance of SVMs for classification problems compared to the pre-existing kernel functions and traditional machine learning models. The custom kernel that will be tested in this paper is displayed below:

$$K(X,\ Y)\ =\ \left(\frac{X^TY}{tanh(\gamma * X^TY + r)}\right)^d$$

Where γ is the gamma term in the kernel function, d is the polynomial degree in the kernel function, and r is the bias term in the kernel function. All terms are user-controlled parameters that help improve the performance of the SVM model.

B. Experiment

To test how well the custom kernel performed for classification problems, a SVM model was built using the custom kernel and tested on three different datasets. Additionally, we compared the custom kernel model to five other SVM models that were built with different kernels: Linear, Polynomial, Radial Based Function (RBF), MKL summation form, and MKL multiplication form. We will also be comparing the custom kernel to traditional machine learning models like logistic regression and decision tree classifier. Both the MKL summation form and multiplication form consists of a linear, polynomial, and RBF kernel. The purpose of using other kernels is to provide a proper understanding of how the custom kernel performs in comparison to pre-existing kernels and machine learning models. Using multiple different dataset shows how robust the kernel is. It is important to know that the kernel performs well on other datasets, and not just a specific one.

For the parameters in the custom kernel, the γ term is equal to 3, d is equal to 3, and r is equal to 2. For the polynomial kernel, d is equal to 3. For the two custom kernels, the parameters we use are γ and d which are both set to 2. These parameters were chosen because they provided the best results. The training size and test size is split 70/30.

Program was implemented with Python 3.9.7 on a Windows 10 64 bit machine with an Intel i5-9600k @ 3.70 GHz CPU and 32 gigabytes of memory. The models and experiments are implemented using Jupyter Notebook and the "SKLearn" library. The custom kernel and the MKL algorithms are implemented by manually creating the kernels and passing them into the SKLearn SVM model. The linear, polynomial, and RBF kernel models used the default kernel function that SKLearn provides. The logistic regression and decision tree model are made using the default setting provided by SKLearn.

The datasets used are the Mushroom Classification, Pima Indian Diabetes, and Iris Species. All three datasets are classification problems and are provided by the UCI Machine Learning Repository. The Mushroom Classification dataset asks to predict if a mushroom is poisonous or not. It has 22 features that are used to predict and 8124 rows. The Pima Indian Diabetes dataset asks to predict if a patient has diabetes or not depending on 8 features and 768 rows. The Iris Species dataset asks to classify what species an Iris flower is. There are three different classes: Iris-setosa, Iris-versicolor, Iris-virginica. We predict this class using four features and 150 rows. We use these datasets since they are very good for testing classification models. The Mushroom Classification and the Pima Indian Diabetes datasets are binary classification problems. The Iris Species dataset has the model predicting three different classes. Using these datasets allow us to test how versatile the custom kernel is in predicting binary class and multi-class problems.

All the data is cleaned before the models are built. Any NULL values and outliers found in the datasets were removed. Label encoders were used to convert string values to numerical values. Pearson correlation coefficient was used to determine what features were important. The features that were dropped were the "Id" feature in the Iris Species dataset and the "veil-type" feature in the Mushroom Classification dataset. The features were scaled using the Standard Scaler function in the SKLearn library.

To compare the performance of the kernels and machine learning models, the execution time, accuracy, precision, recall and F1-score will be used. The execution time will show how fast the model can be built and

predict classes. This is beneficial since the datasets being used are not massive, so it will provide an indication of how it might operate with big data. Accuracy is used to test how many predictions were classified correctly. Precision is used to check how many positive predictions are correct out of all positive predictions. Recall is used to calculate how many true positive predictions out of all the positive classes. F1-score is used to provide an overview of how precision and recall work even for unbalanced datasets. These metrics will provide a proper understanding of how well the models perform. If we only relied on accuracy to determine how well a model performs, then we would not be able to determine if the model performs well with an unbalanced dataset.

C. Results & Analysis

TABLE I

Model/Kernel	Execution Time	Accuracy	Precision	Recall	F1-Score
Linear	0.136	1.000	1.000	1.000	1.000
Polynomial	1.156	1.000	1.000	1.000	1.000
RBF	0.800	0.999	1.000	1.000	1.000
MKL Summation	1.594	1.000	1.000	1.000	1.000
MKL Multiplication	1.927	0.998	1.000	1.000	1.000
Custom	1.910	0.985	0.987	0.987	0.987
Logistic Regression	0.030	1.000	1.000	1.000	1.000
Decision Tree	0.015	1.000	1.000	1.000	1.000

Table 1. Performance of all kernels using SVM as a base learner, Logistic Regression, and Decision Tree Classifier on the Mushroom Classification dataset.

The results displayed in the tables are the average metrics after being run three times with different training and test sets. This is to provide an accurate representation of how the models perform on the datasets.

From the results, the custom kernel does not completely outperform the other models in any of the datasets. It outperformed the MKL multiplication kernel and the decision tree model in the Iris Species dataset on Table 2. It takes longer to build and predict in most of the datasets. The reason for why it takes so long is due to doing a lot of matrix operations. The custom kernel has the worst metrics in the binary classification problems compared to the multi-class problem of the Iris Species dataset. The kernel that has the best overall performance, in comparison to the other kernels, in all three datasets is the RBF kernel. The logistic regression model had the best performance out of all the models used.

TABLE II

Model/Kernel	Execution Time	Accuracy	Precision	Recall	F1-Score
Linear	0.002	0.970	0.970	0.970	0.967
Polynomial	0.006	0.962	0.963	0.960	0.960
RBF	0.002	0.977	0.977	0.977	0.977
MKL Summation	0.011	0.970	0.973	0.970	0.970
MKL Multiplication	0.010	0.933	0.930	0.930	0.930
Custom	0.028	0.948	0.950	0.947	0.943
Logistic Regression	0.023	0.977	0.980	0.980	0.980
Decision Tree	0.002	0.940	0.936	0.930	0.930

Table 2. Performance of all kernels using SVM as a base learner, Logistic Regression, and Decision Tree Classifier on the Iris Species dataset.

TABLE III

Model/Kernel	Execution Time	Accuracy	Precision	Recall	F1-Score
Linear	0.008	0.758	0.753	0.713	0.723
Polynomial	0.007	0.732	0.763	0.657	0.660
RBF	0.008	0.761	0.760	0.713	0.720
MKL Summation	0.061	0.751	0.740	0.713	0.723
MKL Multiplication	0.015	0.643	0.647	0.533	0.477
Custom	0.018	0.555	0.520	0.520	0.520
Logistic Regression	0.005	0.757	0.770	0.730	0.740
Decision Tree	0.002	0.658	0.647	0.643	0.643

Table 3. Performance of all kernels using SVM as a base learner, Logistic Regression, and Decision Tree Classifier on the Pima Indian Diabetes dataset.

In the Mushroom Classification dataset or Table 1, the custom kernel has the worst performance in comparison to the other kernels. However, it still has high performance metrics at 0.985. The linear kernel has the best kernel performance since it is the fastest to build and scores one for all metrics. The decision tree model has the best model performance out of all the models.

In the Iris Species dataset or Table 2, the custom kernel only slightly underperforms against the polynomial kernel for accuracy, precision, recall, and F1-score. However, it takes almost five times longer to build the model. The custom kernel still has very good metrics at around 0.948. The RBF kernel performs the fastest and has the best metrics compared to the other kernels. The logistic regression model has the best performance out of all the other models due to its high prediction rate and fast execution time.

In the Pima Indian Diabetes dataset or Table 3, the custom kernel has the worst performance with all metrics being slightly over 0.5. This is very bad since this means that the custom kernel is only slightly better at correct prediction than flipping a coin. The RBF kernel performed the best out of the other kernels with the highest accuracy and precision values. It also has a very low execution time. The best model for this dataset is a toss-up between the RBF kernel and the logistic regression model. The RBF kernel has higher accuracy, but the logistic regression model has a higher F1-score.

3. Conclusion

Overall, the custom kernel does not outperform the linear, polynomial, RBF, and the MKL summation kernel. It has lower performance metrics and takes longer to build the models. Since it was tested on smaller datasets, the time it would take to build a model with big data would be exponential longer. This is due to using many matrix operations. Ultimately, the custom kernel is not recommended to use for building SVM models that solve classification problems. Based on the experiments, it would be best to build a SVM model with the RBF kernel due to being fast to build and high performance metrics. The MKL summation kernel also performs well for classification problems, if you have time to build the model. One could also just use a traditional machine learning model to build a quick and accurate model. Future work includes testing the kernel on regression problems and optimizing the kernel to reduce the time to build the models.

4. REFERENCES

[1] Gönen, Mehmet, and Ethem Alpaydın. "Multiple kernel learning algorithms." *The Journal of Machine Learning Research* 12 (2011): 2211-2268.

[2] Hastie, T., Tibshirani, R.,, Friedman, J. (2001). The Elements of Statistical Learning. New York, NY, USA: Springer New York Inc..

[3] Hofmann, Martin. "Support vector machines-kernels and the kernel trick." *Notes* 26.3 (2006): 1-16.

[4] Liu, W., Liang, S. & Qin, X. "Weighted *p*-norm distance t kernel SVM classification algorithm based on improved polarization." *Sci Rep* **12**, 6197 (2022). https://doi.org/10.1038/s41598-022-09766-w

[5] Ray, Susmita. "A quick review of machine learning algorithms." *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019.

[6] Rebei, H. and Alharbi, N. (2022) "Legendre Polynomial Kernel: Application in SVM." *Journal of Applied Mathematics and Physics*, **10**, 1732-1747. doi: 10.4236/jamp.2022.105121.