Kernel Fisher's Linear Discriminent and SVM

William Richard

December 12, 2012

1 Introduction

A version of Fisher's Discriminent using Kernels was presented by Mika et. al [1]. It develops an algorithm much like Fisher's Lineary Discriminent, but instead of mapping examples onto a linear vector, it maps them onto a kernelized feature space. This allows for non-linear, more flexible discriminents, which allows Kernelized Fisher's Discriminent (KFD) to classify problems that are not linearly seperable. We attempt to reproduce this algorithm, and compare its results to SVM using the libSVM implimentation [2].

2 Kernel Fisher Discriminent

Kernel Fisher Discrimient is analogeous to Fisher's Linear Discrimient. Let $\boldsymbol{\chi}_1 = \{\boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_{l1}^1\}$ and $\boldsymbol{\chi}_2 = \{\boldsymbol{x}_1^2, \ldots, \boldsymbol{x}_{l_2}^2\}$ be samples from the two classes, and $\boldsymbol{\chi} = \boldsymbol{\chi}_1 \cup \boldsymbol{\chi}_2 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$.

Fisher's linear discriminent is given by the $\boldsymbol{w}$ which maximizes:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_W \boldsymbol{w}} \tag{1}$$

where

$$S_B := (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T \tag{2}$$

$$S_W := \sum_{i=1,2} \sum_{\boldsymbol{x} \in \chi_i} (\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m_i})^T \tag{3}$$

with

$$\boldsymbol{m_i} := \frac{1}{l_i} \sum_{j=1}^{l_i} \boldsymbol{x}_j^i \tag{4}$$

By maximizing (1), one finds the direction which maximizes the difference in the class means (2) while minimizing the class variances (3).

Similarly, KFD attempts to maximize the class means in feature space, while minimizing the class variances, again in feature space. If we take Φ to be a non-lineary mapping into some feature space $\mathcal{F}$, then the analogeous equation to (1) is

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T S_B^\Phi \boldsymbol{w}}{\boldsymbol{w}^T S_W^\Phi \boldsymbol{w}} \tag{5}$$

where

$$S_B^\Phi := (\boldsymbol{m}_1^\Phi - \boldsymbol{m}_2^\Phi)(\boldsymbol{m}_1^\Phi - \boldsymbol{m}_2^\Phi)^T \tag{6}$$

$$S_W^\Phi := \sum_{i=1,2} \sum_{\boldsymbol{x} \in \chi_i} (\boldsymbol{\Phi}(\boldsymbol{x}) - \boldsymbol{m}_i)(\boldsymbol{\Phi}(\boldsymbol{x}) - \boldsymbol{m}_i)^T \tag{7}$$

$$\boldsymbol{m}_i^\Phi := \frac{1}{l_i} \sum_{j=1}^{l_i} \boldsymbol{\Phi}(\boldsymbol{x_j^i}) \tag{8}$$

If we want to use the kernel trick on (5), we need to reformulate it into dot products of input samples in feature space. To start, we know that any solution $w \in \mathcal{F}$ must be in the span of the training samples of $\mathcal{F}$. In other words, since $\mathcal{F}$ is defined by the training samples χ, any vector in that space can be constructed from those training samples. Thus

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x_i}) \tag{9}$$

Using (9) and (8), we can write

$$\boldsymbol{w}^T \boldsymbol{m}_i^\Phi = \frac{1}{l_i} \sum_{j=1}^{l} \sum_{k=1}^{l_i} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k^i) = \boldsymbol{\alpha}^T \boldsymbol{M}_i \tag{10}$$

where

$$(\boldsymbol{M}_i)_j := \frac{1}{l_i} \sum_{k=1}^{l_i} k(\boldsymbol{x}_j, \boldsymbol{x}_k^i) \tag{11}$$

Now, (7) becomes

$$\boldsymbol{w}^T S_B^\Phi \boldsymbol{w} = \boldsymbol{\alpha}^T M \boldsymbol{\alpha} \tag{12}$$

where

$$M := (\boldsymbol{M}_1 - \boldsymbol{M}_2)(\boldsymbol{M}_1 - \boldsymbol{M}_2)^T \tag{13}$$

Similarly, (7) can be transformed into

$$\boldsymbol{w}^T S_W^\Phi \boldsymbol{w} = \boldsymbol{\alpha}^T N \boldsymbol{\alpha} \tag{14}$$

where

$$N := \sum_{j=1,2} K_j (I - 1_{l_j}) K_j^T \tag{15}$$

K_j is an l x l_j matrix with $(K_J)_{nm} := k(\boldsymbol{x}_n, \boldsymbol{x}_m^j)$, or the kernel matrix for class j, I is the identity matrix, and $\mathbf{1}_{l_j}$ is the matrix with all entries $1/l_j$.

Combining (12) and (14), we can find a the discriminent in $\mathcal{F}$ by maximizing

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^T M \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T N \boldsymbol{\alpha}} \tag{16}$$

which is equivilent to finding the leading eignvector of $N^{-1}M$.

To project new examples into the fetaure space, one must simply do the following

$$\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \tag{17}$$

This projection is simply a number, denoting the projected value of the example into $\mathcal{F}$. To classify this value, other classifiers are used. In [1], a linear SVM optimized by gradient descent was used for this final classification step. In this case, a sigmoid was fitted to the data, minimzing the error rate of the projected training set values. The minimization was computed using a rough brute force algorithm, then a truncated Newton's method to get more finely tuned parameters. [3, 4]

3 Experimental Setup

There were two experiments that I wanted to address. The first was how KFD and SVM compare on the same kernel. The second was how they compare with optimal values for a specific kernel algorithm.

In both cases, datasets from the UCI databes were used, and split into a train seciton and a test section. If any dataset had more than 2 classes, classes were combined to create a two class problem. Also, the number of examples from each class were equalized, to avoid overfitting to one class. Then, the data sets were split into test and train divisions, with about one third of each set used for testing.

Three kernels were tested - linear (18), RBF (19) and polynomial (20).

$$k(x, y) = x \cdot y \tag{18}$$

$$k(x, y) = \exp -\gamma \parallel x - y \parallel^2 \tag{19}$$

$$kx, y = (\gamma x y^T + c)^d \tag{20}$$

In order to compare the algorithms on the same kernel, a large variety of kernels were computer, each with different parameters. I computed train kernels, with each entry being a kernel for two train examples, and test kernels, where each entry is the kernel function applied to a test example and a train example. KFD and SVM were trained using the train kernels, and tested on the corresponding test kernel.

To compare the algorithms with optimal kernel parameters, I minimized the mean error rate of the test set using 3 fold cross validation. In other words, each cross validation fold computed a kernel, trained the model, predicted on that fold's test set. The mean was taken of all error rates of the folds, and a minization algorithm was used to attempt to minimize that mean error rate. Once optimial kernel parameters were found, they were tested against the test set, as it was created above. As with the sigmoid fitting, this mean error rate was minimized first using a rough brute force minimization, then a truncated Newton's method [3, 4]

4 Results

4.1 Performance on unoptimized, same kernels

For these experiments, KFD and SVM were trained, without cross validation, on the training set, using a wide range of kernel parameters. They were then tested on the test set. The top three results of each algorithm, on each kernel function, of each dataset appear in tables 1, 2 and 3. In some cases, the best parameters were the same for both algorithms, which is why there are fewer than 6 entries for some datasets.

dataset	KFD acc	SVM acc
ionosphere	76.190	**78.571**
iris	**96.970**	**96.970**
wine	48.936	**91.489**

Table 1: Linear kernel results for KFD and SVM on all datasets, with the best performance bolded. Both entries are bolded in the case of a tie.

dataset	gamma	KFD acc	SVM acc
ionosphere	0.01	**88.095**	73.809
ionosphere	0.1	**92.857**	91.666
ionosphere	1	72.619	**83.333**
ionosphere	10	**90.476**	58.333
iris	0.01	**87.879**	**87.879**
iris	0.1	87.879	**96.970**
iris	1	**93.939**	**93.939**
iris	10	**90.909**	84.848
iris	100	**90.909**	60.606
wine	0.0001	**80.851**	**80.851**
wine	1e-05	**80.851**	**80.851**
wine	0.001	**78.723**	**78.723**

Table 2: Top three RBF kernel parameters for KFD and SVM on all datasets, with the best performance bolded. Both entries are bolded in the case of a tie.

For the RBF kernel, table 2, KFD either beat or tied SVM in almost every case when given the best kernel parameters found. There is one case where SVM has a

dataset	gamma	coef	degree	KFD acc	SVM acc
ionosphere	10	-10	2	**91.667**	75.000
ionosphere	1	0	2	**90.476**	86.905
ionosphere	0.0001	0	2	**89.286**	46.429
ionosphere	0.1	0	2	**89.286**	86.905
ionosphere	0.1	0	3	**89.286**	86.905
iris	0.01	3	2	96.970	**100.000**
iris	0.001	1	3	**96.970**	48.485
iris	0.001	1	4	**96.970**	63.636
iris	0.001	2	3	**96.970**	90.909
iris	0.001	5	3	48.485	**100.000**
iris	0.0001	6	4	48.485	**100.000**
wine	0.1	-10	3	48.936	**95.745**
wine	*	*	*	48.936	**93.617**

Table 3: Top three polynomial kernel parameters for KFD and SVM on all datasets, with the best performance bolded. Both entries are bolded in the case of a tie. The '*' for the wine dataset denote that, for all kernel parameters, the results were the same.

better performance than KFD. Just based on this tabel, KFD appears to be a very strong algorithm, especially consideng that these are the kernel parameters with the best performance.

This conclusion is less resounding when using a polynomial kernel (table 3) or linear kernel (table 1). With a polynomial kernel, KFD had a strong performance in the ionosphere dataset, with a better performance than SVM in all cases. On the iris dataset, SVM was able to achieve perfect accuracy in three cases, and achieved accuracies greater than 75% in almost other cases (including cases not shown here). Finally, the wine dataset does not appear to be easily seperable using a polynomial kernel, as the 48.936% accuracy of KFD implies that the sigmoid fitting was unsucessful. With a linear kernel, SVM outperformed KFD in almost all cases, though not always by a significant margin.

Based on these limited results, it appears that SVM outperforms KFD almost regardless of kernel. Furthermore, KFD appears to be more sensitive than SVM, as we can see from its failures in the wine dataset.

This poor performance may be the fault of KFD, or it may be the result of bad sigmoid fitting. While every attempt was made to exhaustively fit the sigmoid, time

constraints did not allow for this optimization step to be fully tested. Clearly, it worked in many cases, but it is possible that faults in it are responsible for KFD's poor performance in some of the tests.

4.2 Optimized parameters of the RBF kernel

dataset	algorithm	optimal gamma	accuracy
ionosphere	kfd	0.500	71.429
ionosphere	svm	0.285	**89.286**
iris	kfd	0.500	87.879
iris	svm	0.400	**93.939**
wine	kfd	0.200	**72.340**
wine	svm	0.009	70.213

Table 4: Optimal parameters found with 3-fold cross validation on the train set, with accuracy results from predictions on the test set. The better result of each dataset is bolded.

Table 4 shows the results of attempting to discover the optimal parameters for an RBF kernel on both algorithms, as described in section 3. As you can see, the accuracies here are much lower than that found in table 2. This implies that my approach to determining optimal parameters is flawed, though due to lack of time I was not able to sucessfully find those problems. Also due to lack of time, I was not able to attempt to calculate optimal parameters on any other kernels.

References

[1] S. Mika, G. Rätsch, J. Weston, B. Schölkoph, and K. Müller *"Fisher Discriminent Analysis with Kernels"*

[2] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[3] Wright S., Nocedal J. (2006), Numerical Optimization

[4] Nash S.G. (1984), Newton-Type Minimization Via the Lanczos Method, SIAM Journal of Numerical Analysis 21, pp. 770-778