The Mathematica® Journal

Fisher Discrimination with Kernels

Hugh Murrell
Kazuo Hashimoto
Daichi Takatori

Fisher first introduced the Fisher linear discriminant back in 1938. After the popularization of the support vector machine (SVM) and the kernel trick it became inevitable that the Fisher linear discriminant would be kernelized. Sebastian Mika accomplished this task as part of his Ph.D. in 2002 and the kernelized Fisher discriminant (KFD) now forms part of the large-scale machine-learning tool *Shogun*. In this article we introduce the package *MathKFD*. We apply *MathKFD* to synthetic datasets to demonstrate nonlinear classification via kernels. We also test performance on datasets from the machine-learning literature. The construction of *MathKFD* follows closely in style the construction of *MathSVM* by Nilsson and colleagues. We hope these two packages and others of the same ilk will eventually be integrated to form a kernel-based machine-learning environment for *Mathematica*.

■ Introduction

A two-class machine-learning problem requires learning how to discriminate between data points x_i in sample space X belonging to classes $y_i \in \{+1, -1\}$, when given only a set of examples $\{x_i, y_i\}$ from each class. The currently popular support vector machine [1] solves this problem through the construction of a hyperplane $\omega.x + \beta$ that separates the data points x_i, in the sense that all the x_i of a given class are on the same side of the plane. In SVMs the separating plane is chosen to maximize the distances from it to the closest data points.

The original multidimensional machine-learning algorithm [2] solves the same problem by maximizing between-class to within-class scatter ratio. In this article we describe Fisher's technique and how the introduction of a kernel allows nonlinear classifiers. We build a Kernelized Fisher Discriminant package, *MathKFD*, and explore its classification capabilities using synthetic and real datasets.

■ Linear Fisher Discrimination

We follow [3] and [4] in our construction of a Fisher linear discriminant as the vector ω that maximizes:

$$J(\omega) = \frac{\omega^T S_B \omega}{\omega^T S_W \omega}, \tag{1}$$

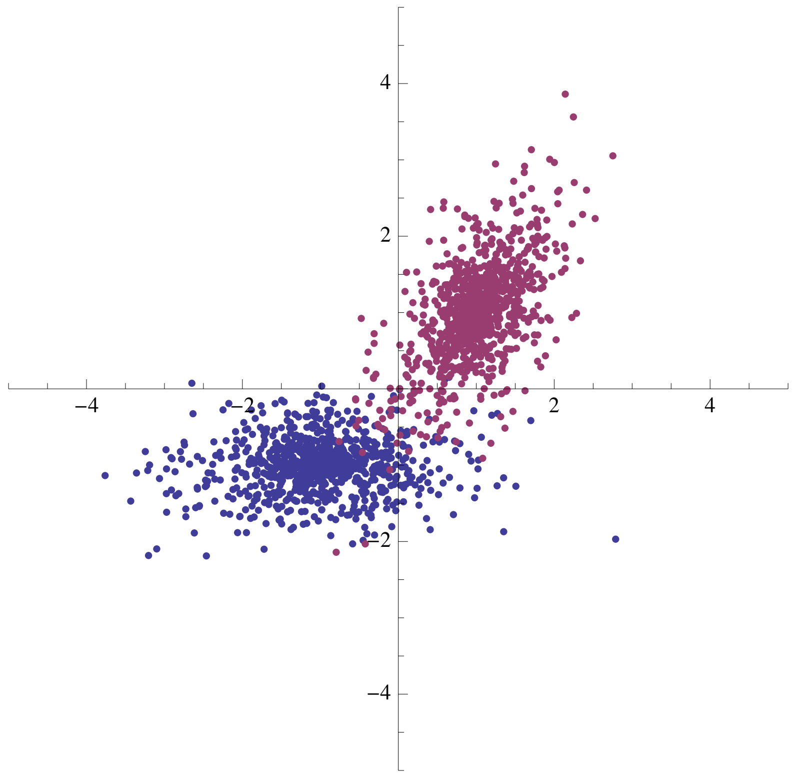
where the between-class and within-class scatter matrices are defined by:

$$S_B = \sum_c N_c (\mu_c - \mu)(\mu_c - \mu)^T \text{ and } S_W = \sum_c \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T, \tag{2}$$

where μ is the mean of the x_i and μ_c is the mean of the x_i within class c.

To understand the meaning of a scatter matrix, we need a test dataset; the usual choice is two sets of normally distributed points with an elliptical shape. The two elliptical sets are rotated and translated away from each other and then adjusted to have zero combined mean. These commands generate two elliptical datasets. The positive (blue) and negative (purple) datasets are used for training a Fisher discriminant.

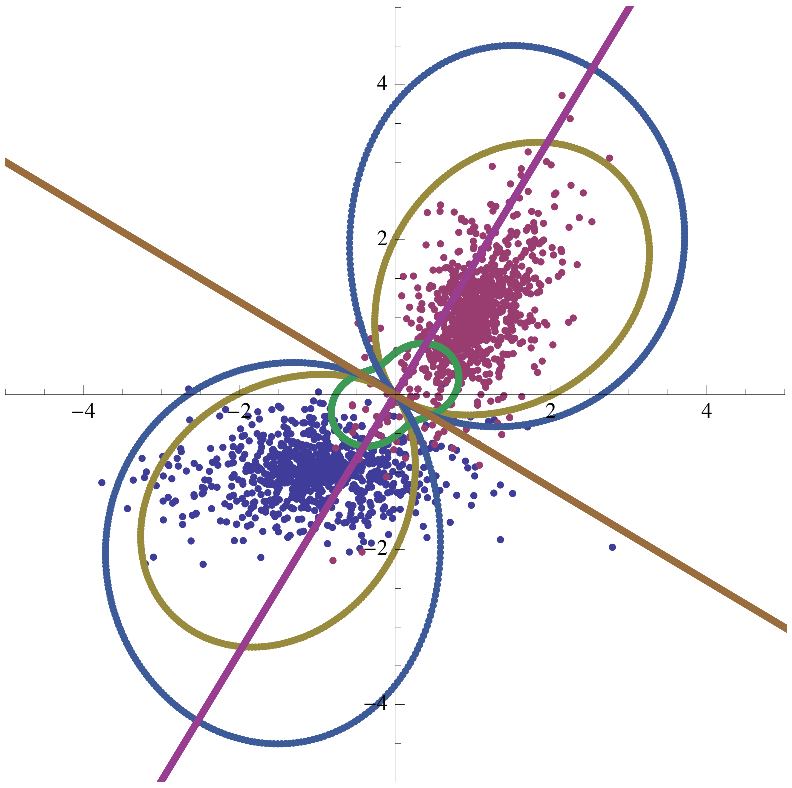
```
EllipsePoint[] := Module[{}, t = RandomReal[{0, Pi}];
   r = RandomReal[NormalDistribution[0, 0.5]];
   {2 r Cos[t], r Sin[t]}];
Xn = Table[EllipsePoint[] - {1, 1}, {1000}];
rm = RotationMatrix[Pi / 3];
Xp = Table[rm.EllipsePoint[] + {1, 1}, {1000}];
mu = Mean[Join[Xn, Xp]];
Xn = Map[# - mu &, Xn];
Xp = Map[# - mu &, Xp];
X = Join[Xn, Xp];
ListPlot[{Xn, Xp}, PlotRange → {{-5, 5}, {-5, 5}},
 AspectRatio → 1]
```



Now we can compute the between-class and within-class scatter matrices for these two-dimensional datasets and plot their action on a potential projection discriminant. The Fisher linear discriminant is the vector that maximizes the scatter ratio and the Fisher separating plane is perpendicular to the Fisher discriminant. These commands generate scatter matrices and plot their action on all unit vectors.

```
Mv[v_] := Outer[Times, v, v];
Action[m_, v_] := (2 / Length[X]) v.m.v;
u[t_] := {Cos[t], Sin[t]};
Sb = Length[Xn] Mv[Mean[Xn] - Mean[X]] +
   Length[Xp] Mv[Mean[Xp] - Mean[X]];
Sw = Apply[Plus, Map[Mv, Map[# - Mean[Xn] &, Xn]]] +
   Apply[Plus, Map[Mv, Map[# - Mean[Xp] &, Xp]]];
PolarPoints[m_] := Table[Action[m, u[t]] u[t],
   {t, 0, 2 Pi, 0.01}];
SbPoints = Table[Action[Sb, u[t]] u[t], {t, 0, 2 Pi, 0.01}];
SwPoints = Table[Action[Sw, u[t]] u[t], {t, 0, 2 Pi, 0.01}];
SrPoints = Table[(Action[Sb, u[t]] / Action[Sw, u[t]]) u[t],
   {t, 0, 2 Pi, 0.01}];
{mr, mt} =
  FindMaximum[Action[Sb, u[theta]] / Action[Sw, u[theta]],
   {theta, 0}];
mt = theta /. mt;
BestProjector = Table[r u[mt], {r, -2 mr, 2 mr, 0.01}];
```

```
BestSeparator = Table[r u[mt + Pi / 2], {r, -2 mr, 2 mr, 0.01}];
ListPlot[{Xn, Xp, SbPoints, SwPoints, SrPoints,
   BestProjector, BestSeparator},
  PlotRange → {{-5, 5}, {-5, 5}}, AspectRatio → 1]
```



▲ **Listing 1.** A *Mathematica* script for generating scatter matrices and plotting their action on all unit vectors.

Curves show the action of the scatter matrices on unit vectors (gold: between, green: within, blue: between or within). The purple line is the Fisher projection vector and the brown line is the Fisher linear discriminator.

■ Fisher Discrimination with Kernels

Often, in the real world, a linear discriminant is not complex enough to separate datasets effectively. To deal with nonlinear separations, we consider a mapping Φ from sample space X into a feature space F. Assuming that the Fisher linear discriminant w in F can be expressed as a linear combination of sample points in F, we require:

$$w = \sum_{i=1}^{l} \alpha_i \Phi(x_i). \tag{3}$$

In terms of α, the objective function $J(\alpha)$ now reads

$$J(\alpha) = \frac{\alpha^T S_B^\Phi \alpha}{\alpha^T S_W^\Phi \alpha}. \tag{4}$$

The between-class scatter is now given by:

$$S_B^\Phi = (M_1 - M_2)(M_1 - M_2)^T \text{ with } (M_i)_j = \frac{1}{l_i} \sum_{k=1}^{l_i} < \Phi(x_j), \Phi(x_k^i) >, \tag{5}$$

where M_i is a vector of length $l = l_1 + l_2$ and $< \Phi(x_j), \Phi(x_k) >$ represents the inner product between data points in the new feature space F.

The within-class scatter is given by:

$$S_W^\Phi = K_1(I - 1_{l_1}) K_1^T + K_2(I - 1_{l_2}) K_2^T \text{ with } K_i = \left[< \Phi(x_j), \Phi(x_k^i) > \right], \tag{6}$$

where 1_{l_i} is a matrix with all entries set to $\frac{1}{l_i}$ and K_i is a matrix of inner products in feature space of dimensions $l \times l_i$. Derivations for S_B^Φ and S_W^Φ can be found in [3, 4, 5], but the important point to note is that the vector notation now applies in the space spanned by the data vectors in R^l and an explicit form for Φ is not required. The scatter matrices can be computed through the inner products $K_{jk} = < \Phi(x_j), \Phi(x_k) >$, and a new test data point x from X can be projected onto ω in F (for future classification) via the computation

$$< w, \Phi(x) > = \sum_{i=1}^{l} \alpha_i < \Phi(x_i), \Phi(x) > . \tag{7}$$

The α_i projection coefficients are computed from training sets by maximizing $J(\alpha)$ in (4). However, the scatter matrices now have dimensions $l \times l$, so the naive technique employed in the 2×2 case of the previous section will not work. To maximize $J(\alpha)$, we must now find the leading eigenvector of

$$A = \left(S_W^\Phi + \lambda I\right)^{-1} S_B^\Phi . \tag{8}$$

λI is a regularizing diagonal term introduced to improve the numerical stability of the inverse computation. See [3, 4, 5] for details.

Fisher discrimination is now cast into a setting whereby the nature of the classification (linear or nonlinear) is entirely governed through the specification of $K(x, y) = < \Phi(x), \Phi(y) >$. The mapping $K : X \times X \to R$ is called the kernel and can be constructed to suit the problem at hand without specifying Φ.

A training algorithm for data X with class labels y is implemented in the *MathKFD* package that accompanies this article. It is available from www.mathematica-journal.com/data/uploads/2011/07/Murrell.zip.

```
Needs["MathKFD`"];
? TrainKFD
```

$\{\alpha,\beta\}$=TrainKFD[K,X,y] trains a Fisher discriminant in feature space (kernel K, training data X, training labels y). The multiplier vector α is computed by maximizing the ratio $(\alpha^T B\ \alpha)/(\alpha^T W\ \alpha)$ where B is the between−class scatter and W is the within−class scatter in the feature space induced by the kernel K. A Mahalonobis bias $\beta = (\mu_+ \sigma_- + \mu_- \sigma_+)/(\sigma_+ + \sigma_-)$ is also calculated. Returns the multiplier vector and the bias as $\{\alpha,\beta\}$ so that subsequent classification of a single data point x can be achieved through α.x − β.

Projection onto the Fisher discriminating vector and various scoring and visualization functions are also provided in the *MathKFD* package. In the next section we make use of the package to create nonlinear Fisher discriminants for synthetic datasets.

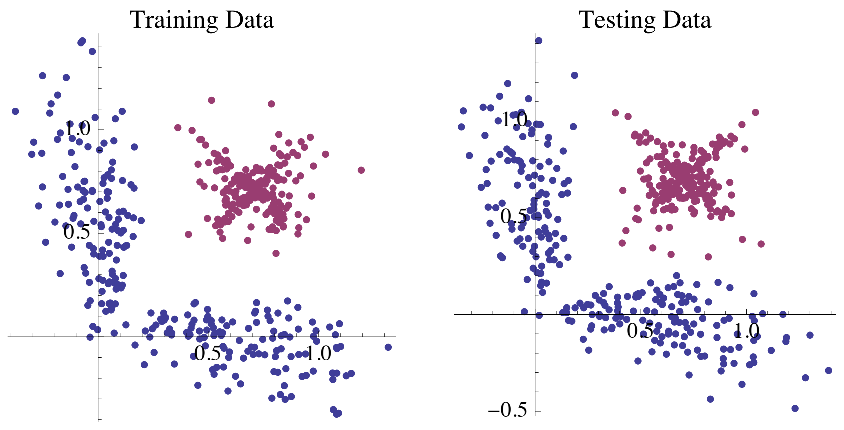
Using Kernels on Synthetic Datasets

In many pattern-recognition problems, the training data requires a nonlinear separating surface [6]. For each specific problem, we could devise some appropriate transformation $\Phi(x)$ from input space X (the domain of the original data) to feature space F. The function Φ must be chosen so that a hyperplane in F corresponds to some desirable class of surfaces in X. How does the analyst choose Φ? The Fisher formulation in the previous section tells us that we need not construct Φ explicitly, but only require an inner product or kernel, $K(x_i, x_j)$. The traditional inner product given by $K(x_i, x_j) = x_i.x_j$ delivers the linear Fisher discriminant. If X itself happens to be a dot product space, then two popular nonlinear kernels are the degree-d polynomial $K(x_i, x_j) = (1 + x_i.x_j)^d$ and the radial basis function $K(x_i, x_j) = e^{-\gamma (x_i - x_j).(x_i - x_j)}$. These two nonlinear kernels and the standard linear kernel are provided in the *MathSVM* package by [1] and are available in *MathKFD* in exactly the same format. To find out more about machine learning with kernels see [7].

A Nonlinear Example in R^2 Using a Polynomial Kernel

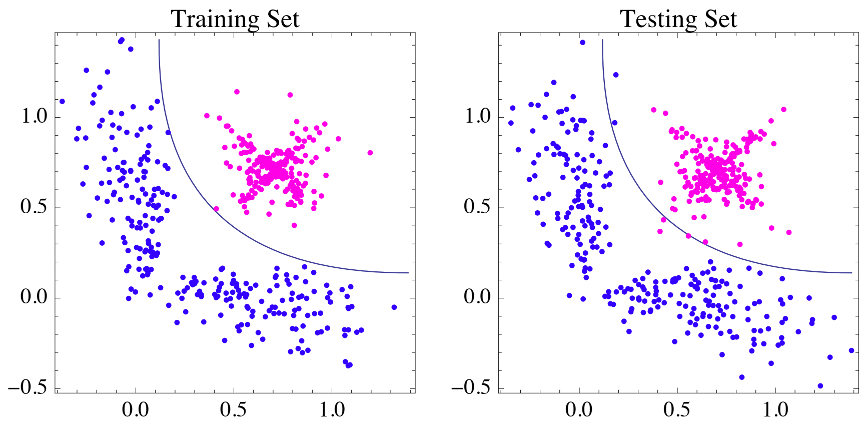
Let us now use *MathKFD* to solve a nonlinear classification problem. We use the polar shapes, a folium and an astroid, to construct two classes of observations. Each observation is characterized by a two-dimensional feature vector and a class assignment. The classes have been selected so that they are not linearly trainable, but may be trainable via a nonlinear kernel. In all the plots that follow, the positive samples are in blue while the negative samples are rendered in purple.

```
FoliumPoint[] :=
  Module[{t, r},
   t = (2 RandomInteger[] - 1)
     RandomReal[NormalDistribution[Pi / 4, 0.2]];
   r = RandomReal[NormalDistribution[0.5, 0.2]]
     (4 Cos[t] Sin[t] Sin[t]);
   {r Cos[t], r Sin[t]}];
AstroidPoint[] :=
  Module[{t, r},
   t = RandomReal[UniformDistribution[{0, 2 Pi}]];
   r = RandomReal[NormalDistribution[0.0, 0.2]];
   {r Sin[t] Abs[Sin[t]], r Cos[t] Abs[Cos[t]]}];
n = 500;
rm = RotationMatrix[-Pi / 4];
X = Join[Table[FoliumPoint[].rm, {n / 2}],
   Table[(AstroidPoint[] + {1, 0}).rm, {n / 2}]];
y = Join[Table[1, {n / 2}], Table[-1, {n / 2}]];
XTest = Join[Table[FoliumPoint[].rm, {n / 2}],
   Table[(AstroidPoint[] + {1, 0}).rm, {n / 2}]];
yTest = Join[Table[1, {n / 2}], Table[-1, {n / 2}]];
DataPlotKFD[X, y, XTest, yTest]
```



The task at hand is then to construct a Fisher discriminant from the training set and use it to classify the test set. We select a nonlinear kernel and train a kernelized Fisher discriminant. Then, because our data originated in R^2, we are able to view the Fisher discriminating curve.

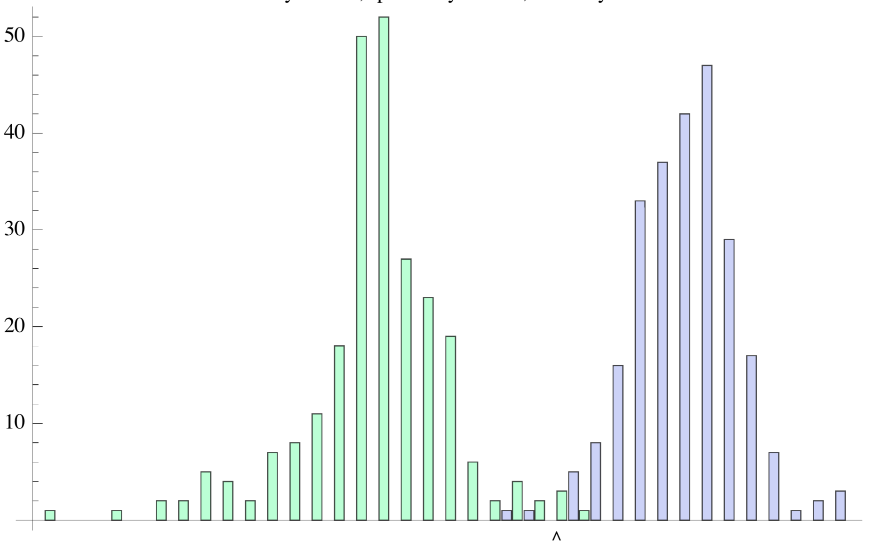
```
kf = PolynomialKernel[#1, #2, 3] &;
{α, β} = TrainKFD[kf, X, y];
ContourPlotKFD[kf, X, y, α, β, XTest, yTest]
```



In general, our datasets are not in R^2 and we will not be able to view separation boundaries in sample space. However, we can always view the performance of the classifier by generating a histogram of projections onto the Fisher discriminator in feature space. *MathKFD* provides a bar chart function showing Fisher classification histograms on a testing dataset. The Mahalonobis classification boundary is marked with an up arrow. In addition to the histograms, `BarChartKFD` reports on the number of features per sample, the kernel used, the number of positive and negative samples in the training and testing data, and three simple success statistics achieved by the classification. Sensitivity measures the classification success rate for positive test samples, specificity measures the success rate for negative test samples, and accuracy measures the success rate for all test samples.

```
BarChartKFD[kf, X, y, α, β, XTest, yTest,
 "KDF Classification"]
```

KDF Classification
features = 2
$K(x_i, x_j) = (1 + x_i . x_j)$^3
training set: $p = 250, n = 250$
testing set: $p = 250, n = 250$
sensitivity = 0.99, specificity = 0.99, accuracy = 0.99

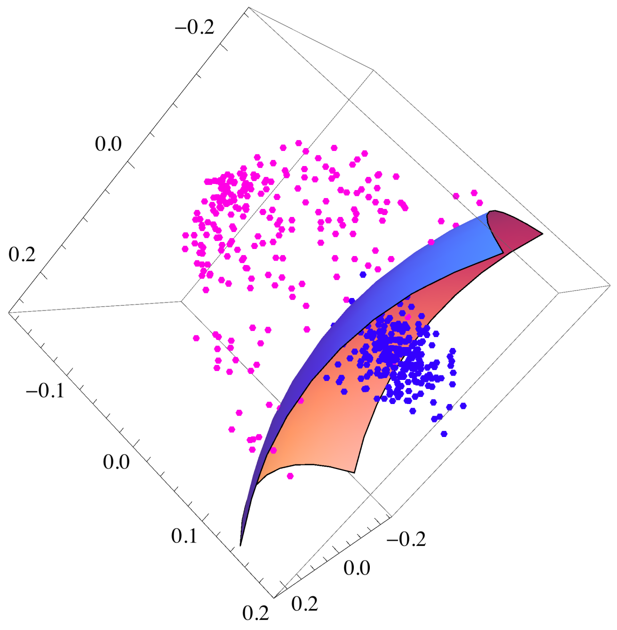


□ A Nonlinear Example in R^3 with Polynomial and RBF Kernel

In the following nonlinear example, borrowed and adapted from [1], the *MathKFD* package requires a polynomial kernel of degree 16 to find a reasonable separating surface for the training data. However, one or two of the positive samples are still classified negative.

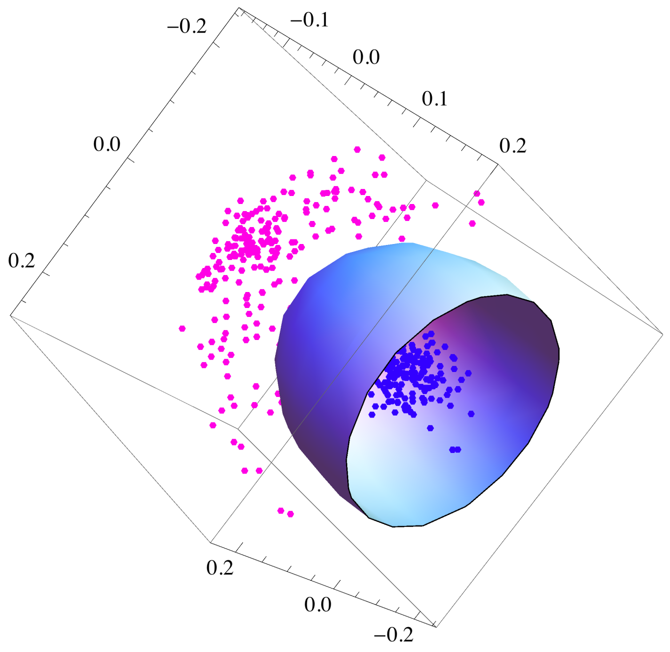
The following commands generate two classes of data in 3D that are separable via a polynomial surface. The positive class is a normal distribution about the origin and shifted up the y axis. The negative class is a quadratic in the x-y plane with each point rotated randomly about the y axis. The data is generated and a Fisher discriminant is trained and plotted together with the training data in 3D.

```
len = 500;
Xp = Map[# + {0, 0.1, 0} &,
   RandomReal[NormalDistribution[0, 0.03], {len / 2, 3}]];
Xn = Table[
   {RandomReal[NormalDistribution[i / len - 1 / 4, 0.02]],
     RandomReal[NormalDistribution[(2 i / len - 1 / 2)^2 - 1 / 6,
       0.01]], 0}.
    RotationMatrix[RandomReal[{0, 2 Pi}], {0, 1, 0}],
   {i, len / 2}];
X = Join[Xp, Xn];
y = Join[Table[1, {len / 2}], Table[-1, {len / 2}]];
kf = PolynomialKernel[#1, #2, 16] &;
{α, β} = TrainKFD[kf, X, y];
ContourPlot3DKFD[kf, X, y, α, β, X, y]
```



The same data is used to train and display a radial basis discriminant. The radial basis kernel with parameter $\gamma = 2$ performs much better, separating the training set completely.

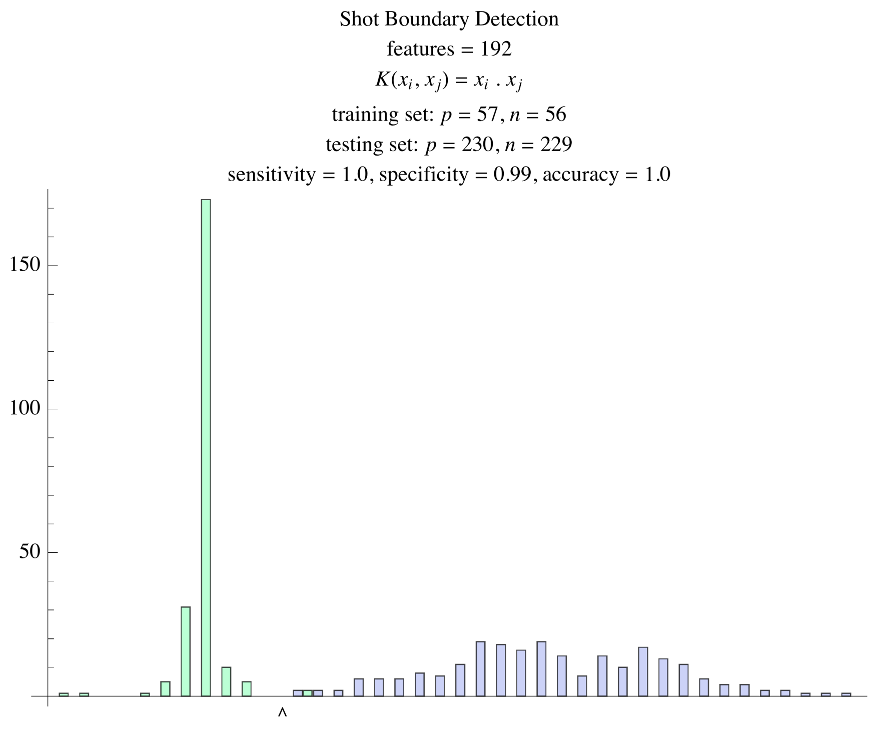
```
kf = RBFKernel[#1, #2, 2] &;
{α, β} = TrainKFD[kf, X, y];
ContourPlot3DKFD[kf, X, y, α, β, X, y]
```



■ Shot Boundary Detection

In 2006, TRECVID [8] completed a second two-year cycle devoted to automatic segmentation of digital video, commonly referred to as the shot boundary detection problem. Researchers around the globe competed to produce the best possible shot boundary detector. See [9] for the prime example. Support vector machines have played their part in this machine-learning contest [10]. In this section we use the *MathKFD* package to create a shot boundary classifier. The training and testing datasets used here have been curated from the digital video sets used to test TRECVID submissions. The result here is the Fisher projection on test data for shot boundary KFD showing the performance of the discriminator.

```
sbTrain = ReadList["sbTrain.txt", Table[Number, {193}],
   WordSeparators → {"\r", " "}];
sbTest = ReadList["sbTest.txt", Table[Number, {193}],
   WordSeparators → {"\r", " "}];
{y, X} = {Map[First, sbTrain], Map[Rest, sbTrain]};
{yt, Xt} = {Map[First, sbTest], Map[Rest, sbTest]};
kf = IdentityKernel[#1, #2] &;
{α, β} = TrainKFD[kf, X, y];
BarChartKFD[kf, X, y, α, β, Xt, yt,
 "Shot Boundary Detection"]
```



▲ **Figure 1.** The Fisher projection on test data for shot boundary KFD showing the performance of the discriminator.

As can be seen, the shot boundary detection data has been well curated. Researchers at KDDI R&D Laboratories were responsible for frame-based features [11]. We have used their data to compute feature differences across known shot boundaries for our positive $+1$ and feature differences at points where shot boundaries are known to be absent for our negative -1 class. The shot boundary training and testing sets used here have been included with the *MathKFD* distribution.

■ Splice Site Recognition

The kernel formulation comes into its own when feature vectors are not Euclidean. Classification problems from bioinformatics usually involve string data. There is no natural ordering of samples. However, the various alignment algorithms that have been developed for bioinformatics provide a similarity measure. In this section we attempt to create a useful kernel for a bioinformatics problem via the `EditDistance` procedure.

Splice sites are locations within a gene that mark exon-intron boundaries. Exons are those sections within the gene that code for protein, whereas introns do not code for protein. Splice sites are either of type donor (if they mark an exon to intron boundary) or of type acceptor (if they mark an intron to exon boundary). Donor sites occur at a `GT` dimer, whereas acceptor sites occur at an `AG` dimer. Not all occurrences of these two dimers enforce a splice. In fact very few of them do. For a splice to occur, certain motifs must be present upstream and downstream from the dimer. A full discussion of the splice site recognition problem is given in [12] and the datasets have been made publicly available.

The dataset for acceptor sites consists of DNA strings of length 60 centered around the `AG` dimer. For each sample there are 30 nucleotides upstream of the `AG` and 28 nucleotides downstream. The samples have been classified $+1$ for a true acceptor site and -1 for a decoy acceptor site. We use a small subset of the acceptor sites to train a Fisher discriminant with a *Mathematica* implementation of the locality-improved polynomial string kernel.

The locality-improved polynomial string kernel is obtained by comparing two strings within a window of length $2\,w+1$ centered at position p.

$$W_p(s, t, d, w) = \left(1 - \frac{1}{2\,w+1}\,\delta\left(s_{p-w..\,p+w},\, t_{p-w..\,p+w}\right)\right)^d, \tag{9}$$

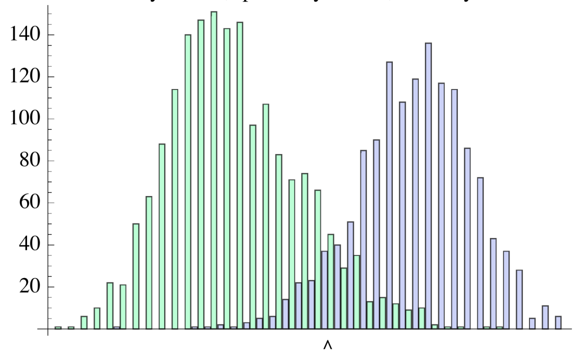
where δ is the edit distance; *Mathematica*'s built in function `EditDistance` is used to count mismatches within the window. To complete the kernel computation, the window now slides along the length of the strings and a weighted contribution from each position is accumulated;

$$K(s, t, d, w) = \sum_{p=w+1}^{n-w} \left(\frac{n}{2} - \left|\frac{n}{2} - p\right|\right) W_p(s, t, d, w), \tag{10}$$

where n is the length of the strings, and window size w and degree d are parameters of the kernel. This locality-improved string kernel (implemented in *MathKFD* as `Local`⋱ `StringKernel`) is used here to train and test an acceptor site classifier.

```
accTrain = ReadList["accTrain.txt", {Number, Word},
   WordSeparators → {"\r", " "}];
{y, X} = {Map[First, accTrain], Map[Rest, accTrain]};
accTest = ReadList["accTest.txt", {Number, Word},
   WordSeparators → {"\r", " "}];
{yt, Xt} = {Map[First, accTest], Map[Rest, accTest]};
kf = LocalStringKernel[#1, #2, 2, 3] &;
{α, β} = TrainKFD[kf, X, y];
BarChartKFD[kf, X, y, α, β, Xt, yt,
 "Acceptor Splice Site Classifier"]
```

Acceptor Splice Site Classifier
features = 1
$K(x_i, x_j) = \text{LocalStringKernel}[x_i, x_j, 2, 3]$
training set: $p = 100, n = 100$
testing set: $p = 1392, n = 1775$
sensitivity = 0.94, specificity = 0.90, accuracy = 0.92



Our Fisher classifier with a localized string kernel achieves a 92% accuracy. In [12] the authors claim a 98.5% accuracy statistic. This is to be expected since we only use a small proportion of their training data here (200 training samples in our experiment as opposed to 5722 training samples in their experiments). *MathKFD* using the `LocalString⋱ Kernel` struggles with large training and testing sets. A fast alignment routine may alleviate matters.

■ Conclusion

In this article, we have demonstrated the utility of the *MathKFD* package for solving pattern-recognition problems, both synthetic and real. This package complements the recently published *MathSVM* package and thus contributes to *Mathematica*'s capabilities as a machine-learning platform. However, there is still a long way to go. Suggested directions for future development include:

- Adaption of the `QPSolve` function based on [13] from *MathSVM* [1] to also solve the optimization problem for the kernel Fisher discriminant.

- Development of a `ReducedKFD` function that must select a subset of the current training set to create a new training set with equal (or nearly equal) discriminating powers. Unlike the SVM, KFD suffers from the fact that all samples in the training set are used to classify, hence the need for an approximate training subset.
- Development of *Mathematica* interfaces to industrial strength machine-learning systems such as *libSVM* [14].

The high-level programming language and symbolic capabilities of *Mathematica* could make it the platform of choice for bioinformatics work. We hope a strong and stable machine-learning environment will go some way to realizing this dream.

■ References

[1] R. Nilsson, J. Bjorkegren, and J. Tegner, “A Flexible Implementation for Support Vector Machines,” *The Mathematica Journal*, **10**(1), 2006 pp. 114–127. www.mathematica-journal.com/issue/v10i1/SupportVectorMachines.html.

[2] R. A. Fisher, “The Statistical Utilization of Multiple Measurements,” *Annals of Eugenics*, **8**(4), 1938 pp. 376–386. onlinelibrary.wiley.com/doi/10.1111/j.1469-1809.1938.tb02189.x/pdf.

[3] S. Mika, G. Ratsch, J. Weston, B. Schölkopf, and K. R. Müller, “Fisher Discriminant Analysis with Kernels,” in *Neural Networks for Signal Processing IX, 1999: Proceedings of the 1999 IEEE Signal Processing Society Workshop*, Madison, WI, IEEE, 1999 pp. 41–48. courses.cs.tamu.edu/rgutier/cpsc689_f08/mika1999kernelLDA.pdf.

[4] M. Welling. “Fisher Linear Discriminant Analysis.” (Jul 5, 2011) www.ics.uci.edu/~welling/classnotes/papers_class/Fisher-LDA.pdf.

[5] S. Mika, “Kernel Fisher Discriminants,” Ph.D. thesis, Elektrotechnik und Informatik der Technischen Universität Berlin, 2002.

[6] V. N. Vapnik, *Statistical Learning Theory*, New York: John Wiley & Sons, 1998.

[7] B. Schölkopf and A. J. Smola, “A Short Introduction to Learning with Kernels,” in *Advanced Lectures on Machine Learning: Lecture Notes in Computer Science*, Vol. 2600, Berlin: Springer-Verlag, 2003 pp. 41–64. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.4786&rep=rep1&type=pdf.

[8] A. F. Smeaton, P. Over, and W. Kraaij, “Evaluation Campaigns and TRECVid,” in *Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*, New York: ACM Press, 2006 pp. 321–330. www.mendeley.com/catalog/evaluation-campaigns-trecvid.

[9] C. Petersohn, “Fraunhofer HHI at TRECVID 2004: Shot Boundary Detection System,” in *Proceedings of the Thirteenth Text REtrieval Conference, (TREC 2004)*, Special Publication 500-261, National Institute of Standards and Technology (NIST), (E. M. Voorhees and L. P. Buckland, eds.), Washington, D.C.: U.S. Government Printing Office, 2004. www-nlpir.nist.gov/projects/tvpubs/tvpapers04/fraunhofer.pdf.

[10] G. Camara Chavez, M. Cord, S. Philipp-Foliguet, F. Precioso, and A. de Albuquerque Araujo, “Robust Scene Cut Detection by Supervised Learning,” in *Proceedings of 14th European Signal Processing Conference (EUSIPCO 2006)*, Florence, Italy. publi-etis.ensea.fr/2006/CCPPD06.

[11] K. Matsumoto, M. Naito, K. Hoashi, and F. Sugaya, "SVM-Based Shot Boundary Detection with a Novel Feature," in *Proceedings of IEEE International Conference on Multimedia & Expo (ICME 2006)*, Toronto, Canada, IEEE, 2006 pp. 1837–1840. www.informatik.uni-trier.de/~ley/db/conf/icmcs/icme2006.html.

[12] G. Rätsch and S. Sonnenburg, "Accurate Splice Site Detection for Caenorhabditis elegans," *Kernel Methods in Computational Biology* (B. Schölkopf, K. Tsuda, and J. P. Vert, eds.), Cambridge, MA: MIT Press, 2004 pp. 277-298.

[13] S. S. Keerthi and E. G. Gilbert, "Convergence of a Generalized SMO Algorithm for SVM Classifier Design," *Machine Learning*, **46**, 2002 pp. 351–360. deepblue.lib.umich.edu/bitstream/2027.42/46957/1/10994_ 2004_Article _ 380512.pdf.

[14] C.-C.Chang and C.-J. Lin. "LIBSVM: A Library for Support Vector Machines." (Jul 6, 2011) doi>10.1145/1961189.1961199.

H. Murrell, K. Hashimoto, and D. Takatori, "Fisher Discrimination with Kernels," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.13–13.

About the Authors

Hugh Murrell teaches Computer Science on the Pietermaritzburg campus of the University of KwaZulu-Natal, South Africa. This work was undertaken while on sabbatical, visiting the Hashimoto Lab of Tohoku University, Sendai, Japan.

Kazuo Hashimoto is the KDDI professor in charge of the Advanced Information Exchange Technology Laboratory (Hashimoto Lab) in the Graduate School of Information Science of Tohoku University, Sendai, Japan.

Daichi Takatori is a student in the Graduate School of Information Science of Tohoku University, Sendai, Japan. He plans to use support vector machines to solve the shot boundary detection problem.

Hugh Murrell
School of Computer Science
Faculty of Science and Agriculture
University of KwaZulu-Natal
Pietermaritzburg, South Africa
murrellh@ukzn.ac.za

Kazuo Hashimoto
Advanced Information Exchange Technology Laboratory
Graduate School of Information Science
University of Tohoku, Sendai, Japan.
kh@aiet.ecei.tohoku.ac.jp

Daichi Takatori
Graduate School of Information Science
University of Tohoku, Sendai, Japan.
takatori@aiet.ecei.tohoku.ac.jp

Acknowledgments

This work was undertaken during a sabbatical awarded by the University of KwaZulu-Natal to Murrell while visiting Hashimoto's Networking Lab at Tohoku University in Sendai, Japan. Takatori undertakes collaborative research funded by KDDI R&D Laboratories and Tohoku University. The visit and the work were directly funded by the South African National Bioinformatics Network. The authors are grateful to the NBN, UKZN, Tohoku University, and KDDI R&D Laboratories for supporting this work.

The authors are also grateful to Roland Nilsson, who made the *MathSVM* package available, to KDDI R&D Laboratories, who supplied the shot boundary detection data, and to the Friedrich Miescher Laboratory (Ratsch group), who made the splice site data available through their web site. This work could not have been carried out without the open access attitude of the established researchers in the machine-learning community.