Comparing Kernel Linear Discriminant Analysis and Support Vector Machines

Ellis Wright

May 2022

Abstract

Support Vector Machines (SVM) and Linear Discriminant Analysis (LDA) are two powerful tools for classification. In their simplest implementation they rely on the underlying data being linearly separable. Applying kernels to the algorithms increases their robustness with regard to nonlinear data. The SVM algorithm finds a hyperplane that maximizes the distance between each class. The LDA algorithm projects the data onto a direction that maximizes the distance between each class and minimizes the distance for data within the same class. Intuitively, one would expect the resulting directions to be similar. We show how the two algorithms compare on a made up non-linear dataset and then on a dataset to classify normal and abnormal electrocardiogram (ECG) readings. On the ECG data, the algorithms differed with the same kernel. Overall, both algorithms performed well on the data with SVM taking a slight edge.

1 Introduction

The goal of this paper is to investigate how the support vector machine (SVM) decision boundary compares to the direction found by linear discriminant analysis (LDA). To reach a larger audience, this paper incorporates and technical and nontechnical subsection within each section. LDA and SVM are two mathematically intensive methods but the reasoning behind their solutions are easily interpretable.

1.1 Nontechnical

For people without a mathematical background we have created an artificial dataset depicting a game of dodgeball (figure 1). The data consists of two teams, red and blue, standing in a gym. Each player has two features, an x and y coordinate. The goal is two draw a line between the players so that the teams are on opposing sides. If we see a new player, and only know their x and y coordinate, we use the line the algorithm found to decide which team they are on. The process of classifying new players onto one of two teams is a type of binary classification. LDA and SVM have two different methods to determine which team a new player is on. The procedure behind SVM is harder to understand so we demonstrate LDA can lead to similar results.

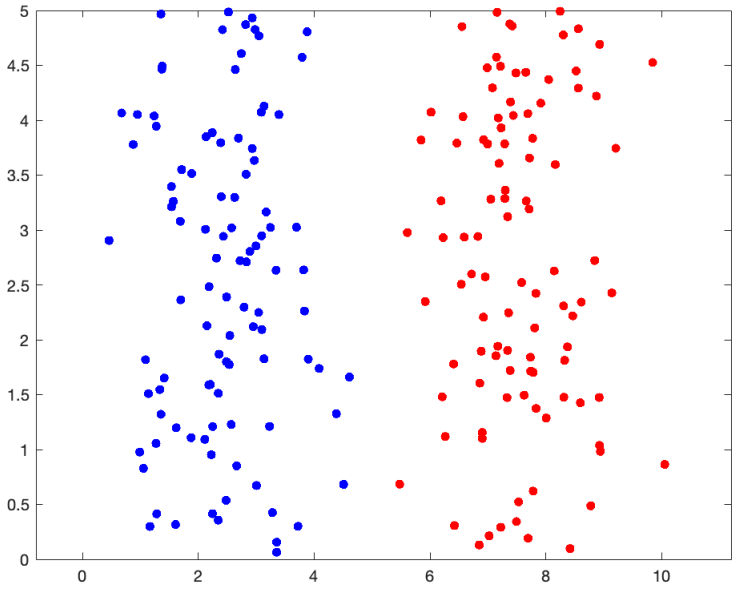

Figure 1: Blue and Red Team

1.2 Technical

A dataset $\mathbb{D}$ is assumed to consist of p data points with each having n features. Each feature is a real quantity, thus $\mathbb{D} \in \mathbb{R}^{nxp}$. Each data point, $x^{(j)}$ for $1 \leq j \leq q$, belongs to one of two classes, $c^{(j)} = \{-1,+1\}$.

2 LDA Review

2.1 Nontechnical

In LDA, the goal is to find a line such that when we flatten the points onto this line we can easily tell which player is on which team. When we take these data points and flatten them onto the bottom axis in figure 1, we see two different distributions as shown in figure 2.

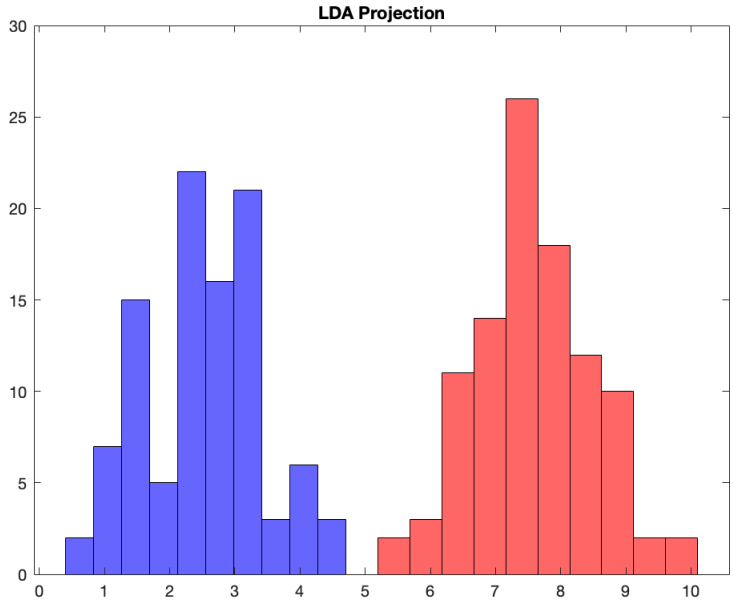

Figure 2: Blue and Red Team projected

It is easy to see the separation of the two teams in figure 1 but, it is even easier in figure 2 because the data has been reduced to one number. The goal of LDA is to find the line.

LDA does this by defining two useful quantities, each team's center. We want each player to be as close to their team center as possible. We also want the centers of different teams to be as far away as possible from each other. LDA tracks the squared distance every player is away from their team center. Then it finds the center of everyone (global center), and keeps track of the squared distance each team center is from the global center.

The best line will separate teams the most while keeping teammates close.

2.2 Technical

In general, for classification problems there may be many classes. Suppose each point is classified into one of r classes. Binary classification is the special case where r = 2. There are two goals; 1) maximize the scatter between each class so that members of different classes are spread out, and 2) minimize the scatter within each class so members of the same class are close to each other. LDA operates under the assumption that within-class data is centered (the mean is 0). First, define two scatter matrices: between class scatter S_b, and within class scatter S_w.

$$S_w = \sum_{i=1}^{r} \sum_{x^{(j)} \in c_i} (x^{(j)} - \overline{c_i})(x^{(j)} - \overline{c_i})^T$$

$$S_b = \sum_{i=1}^{r} (\overline{c_i} - \overline{X})(\overline{c_i} - \overline{X})^T$$

Here, $\overline{X}$ is the mean of the data ($\frac{1}{p}\sum_j^p x^{(j)}$), c_i is the ith class and $\overline{c_i}$ is the mean of the ith class.
To reach the goals, a direction q that maximizes $q^T S_b q$ and minimizes $q^T S_w q$, must be found. Therefore, we

can create the objective function:

$$\mathrm{F(q)} = \frac{q^T S_b q}{q^T S_w q}$$

To maximize F(q) the numerator is maximized and the denominator is minimized. Note, F(q) = F(αq) for any real scalar α. The vector q can always be scaled to ensure $|q^T S_w q| = 1$. Thus, the problem can be transformed into a constrained optimization problem:

$$\text{argmax}\{\mathrm{F(q)}\} \text{ given}$$

$$|q^T S_w q| = 1$$

The algorithm proceeds under the assumption S_w is symmetric positive semi definite (spsd).

$$\text{Matrix A is spsd} \iff A = A^T \text{ and } M \geq 0$$

Therefore, with a cholesky decomposition ($S_w = M^T M$) the problem becomes

$$\text{argmax}\{q^T S_b q\} \text{ given } q^T M^T M q = 1$$

$$\implies \text{argmax}\{q^T S_b q\} \text{ given } (Mq)^T (Mq) = 1$$

With the substitution w = Mq we have

$$\text{argmax}\{w^T M^{-T} S_b M^{-1} w\} \text{ given } |w|^2 = 1$$

This is the form of solving for the lagrange multipliers. We use the lagrange function

$$\mathrm{L(w,}\lambda) = w^T M^{-T} S_b M^{-1} w - \lambda(|w|^2 - 1)$$

We solve for the critical points,

$$0 = \nabla L(w, \lambda) = 2(M^{-T} S_b M^{-1} w - \lambda w)$$
$$M^{-T} S_b M^{-1} w = \lambda w$$

Notice that this is the form of eigenpairs. The lagrange multipliers are the eigenvalues of $M^{-T} S_b M^{-1}$ and the direction w are the corresponding eigenvectors. The maximizer is the eigenpair with the largest magnitude eigenvalue. Finally q, the desired direction, is obtained $q = M^{-1} w$. To find more directions, $q^{(1)}, ..., q^{(k)}$ (for different projections or subspaces), the first k eigenpairs of $M^{-T} S_b M^{-1}$ must be calculated.

3 SVM Review

3.1 Nontechnical

Similarly to LDA, SVM tries to find a line that separates the two teams. However SVM does not flatten the data. Instead, it finds two lines that bound the teams and then it chooses the line in between these two (Figure 3).

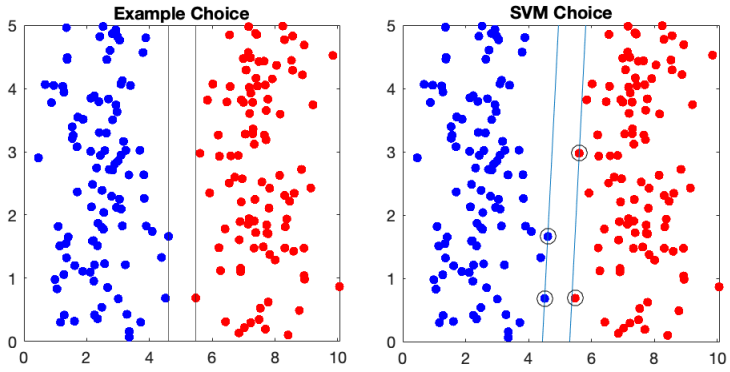

Figure 3: Different choices to draw the team boundaries

In our dodgeball example, we could draw several pairs of lines as shown in figure 3, but how can we tell which is the best pair? We assume that the best pair of lines are the ones that are the furthest apart. This is because we want to maximize the distance between two lines that separate the teams. In figure 3, the circled players are the ones that the SVM classifier uses to make up the boundary between the teams.

3.2 Technical

SVM is a binary classification algorithm. SVM tries to find a decision boundary that separates the two classes (± 1) from each other. If the data exists in $\mathbb{R}^n$ then a decision boundary, or hyperplane, exists in $\mathbb{R}^{n-1}$. The goal is to find a hyperplane H $\in \mathbb{R}^{n-1}$ that separates the classes with the best generalization. Generalization is some measure of how well one can classify unseen data. SVM optimizes a constrained system of linear equations to find the hyperplane.

Let H(w,s) $=\{x|\langle w, x\rangle = s\}$ be a hyperplane. For now, let the inner product be the standard dot product, $(w^T x)$. The distance a point is to the hyperplane is the magnitude of the orthogonal projection onto the plane. Therefore, the distance between two hyperplanes, $H(w, s_1)$ and $H(w, s_2)$ equals $|s_1 - s_2|$. The intuition of this step is to create two parallel hyperplanes that bound each class respectively. The gap between two hyperplanes is called the margin. We know that there exists a hyperplane perfectly in between these two hyperplanes i.e. $H(w, s \pm h) = \{x|w^T x = s \pm h\} = H(w, s_1) \cup H(w, s_2)$ for some value s. Thus, the margin has magnitude 2h. If H separates the data, then for positively classified data $w^T x > s + h$ (it lies above the boundary) and for negative data is $w^T x \leq s - h$ (it lies below the boundary). After moving s to the left and scaling everything by $\frac{1}{h}$ we can write the conditions:

$$\text{if } c^{(j)} = +1,\ \langle \tfrac{1}{h} w, x^{(j)} \rangle - \tfrac{s}{h} \geq 1$$
$$\text{if } c^{(j)} = -1,\ \langle \tfrac{1}{h} w, x^{(j)} \rangle - \tfrac{s}{h} \leq -1$$

Let q $= \frac{1}{h} w$ and b $= \frac{s}{h}$. Notice that we can write these conditions compactly as

$$c^{(j)}(q^T x^{(j)} - b) \geq 1 \text{ for } 1 \leq j \leq p$$

This is because when multiplying both sides by $c^{(j)} = +1$ the value does not change, but when $c^{(j)} = -1$, the '$\leq$' symbol flips.

Theoretically, the hyperplane that has the best generalizability is the one that maximizes the margin. The margin has length 2h which means from our substitution we are trying to maximize $\frac{2}{||q||}$. To maximize this quantity we need to minimize the denominator. Therefore, we create the constrained optimization:

$$\text{minimize } \tfrac{1}{2}||q||^2 \text{ subject to } c^{(j)}(q^T x^{(j)} - b) \geq 1 \text{ for } 1 \leq j \leq p$$

Unfortunately, this only works when the constraints are met meaning the underlying data is truly linearly separable. In most real world cases the data is not completely separable, it is more nuanced. To measure misclassified data we add slack variables (ζ). Notice if a data is misclassified then $c^{(j)}(q^T x^{(j)} - b) < 1$ and so $0 < 1 - c^{(j)}(q^T x^{(j)} - b)$. Let I be an indicator vector where $I_j = 1$ if the constraint is met. We define the slack variables as

$$\zeta_j = \begin{cases} 0 & I_j = 1 \\ 1 - c^{(j)}(q^T x^{(j)} - b) & I_j = 0 \end{cases}$$

When the slack variable is 0, it is correctly classified. When it is not, we know the condition is not satisfied so $c^{(j)}(q^T x^{(j)} - b) < 1$ which means $\zeta_j > 0$. Ideally, all the slack variables are 0, however if this is not possible we want to minimize the number of nonzero entries. To promote sparsity we want to minimize $||\zeta||_1$, this is sometimes called l_1 magic or lasso regression. Finally, we create the objective function:

$$\begin{gathered} \text{minimize } \tfrac{1}{2}||q||^2 + \lambda \textstyle\sum_{i=1}^{p} \zeta_i \\ \text{subject to } c^{(j)}(q^T x^{(j)} - b) - 1 \geq 0 \text{ and } \zeta_j \geq 0 \text{ for } 1 \leq j \leq p \end{gathered}$$

Here, λ is a hyperparameter that effects how important the slack variables are- it is a trade-off between generalizability and error. This optimization technique has known solutions when representing this as a primal-dual optimization. In its general form, we can express the problem as

$$\begin{gathered} \text{let f(q,b,}\zeta) = \tfrac{1}{2}||q||^2 + \lambda \textstyle\sum_{i=1}^{p} \zeta_i \\ \text{let } g_j(q, b, \zeta) = -\zeta_j \text{ and} \\ \text{let } g_{j+p}(q, b, \zeta) = 1 - c^{(j)}(q^T x^{(j)} - b) - \zeta_j \text{ for } 1 \leq j \leq p \end{gathered}$$

We then create the lagrange function

$$L(q, b, \zeta, \alpha) = \tfrac{1}{2}||q||^2 + \lambda \textstyle\sum_{i=1}^{p} \zeta_i - \sum_{i=1}^{p} \alpha_i \zeta_i - \sum_{i=1}^{p} \alpha_{i+p}(c^{(j)}(q^T x^{(j)} - b) + \zeta_i - 1)$$

Substituting $y^{(j)} = c^{(j)} x^{(j)}$, $\beta_j = \alpha_{j+p}$, and defining G where $G_{ij} = y^{(i)T} y^{(j)}$, we get the problem

$$\begin{gathered} \text{argmax}\{\beta^T G \beta + e^T \beta\} \text{ constrained to} \\ 0 \leq \beta_i \leq C \text{ and } \textstyle\sum_{i=1}^{p} \beta_i c^{(i)} = 0. \end{gathered}$$

Where e is a vector of ones in $\mathbb{R}^p$ and C is a 'box constraint' to ensure α_j are positive. To solve this equation we apply the sequential minimal optimization (SMO) algorithm. Details are left to the reader in [1]. Finally we find the q and b by

$$\begin{gathered} \text{q} = \textstyle\sum_{j=1}^{p} \beta_j y^{(j)} \text{ and} \\ \text{b} = \tfrac{1}{m} \textstyle\sum_{j=1}^{m} (q^T x^{(i_j)} - c^{(i_j)}) \text{ where } i_j \text{ to } i_m \text{ correspond to the k's where } \beta_k \leq C \end{gathered}$$

4 Nonlinear Extension using Kernels

4.1 Nontechnical

In the dodgeball example, the teams are set up in a way such that a straight line (linear) can be found from their coordinates. But, what if there was mixing of the teams? Or, what if a curved line (nonlinear) was needed as depicted in figure 4. It is impossible to draw a straight line to separate the two teams. The semicircle has been added to show that there exists a separating boundary. But how can we find it and do we need to?

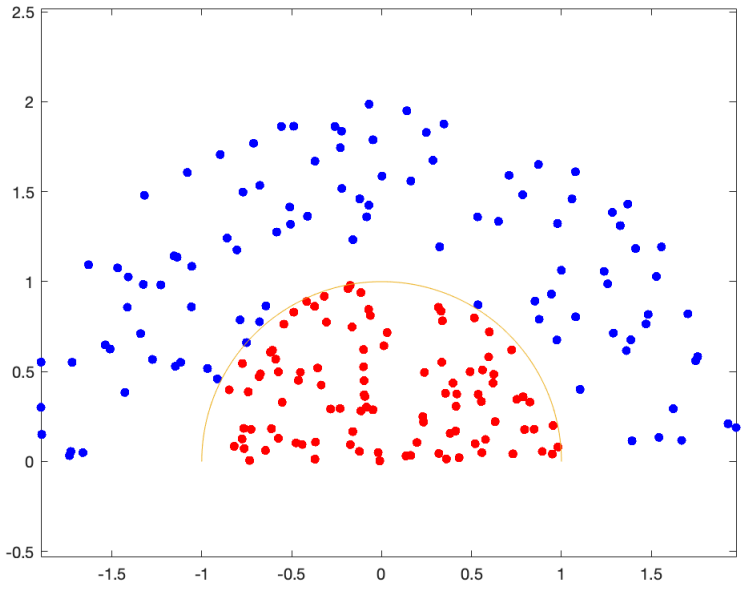

Figure 4: New dodgeball setup

All the players have two features, an x and y coordinate. Mathematically, this is two-dimensional. What if there was a way to add more dimensions (extension)? What if instead of relying on just their x and y coordinate, we create a third coordinate that depicts how far away from point (0,0)? This is how we add the third feature: (x,y) $\rightarrow$ (x,y,z) where z $= x^2 + y^2$. The 3D plot is shown in figure 5. If you were to look from above, it would look like figure 4.

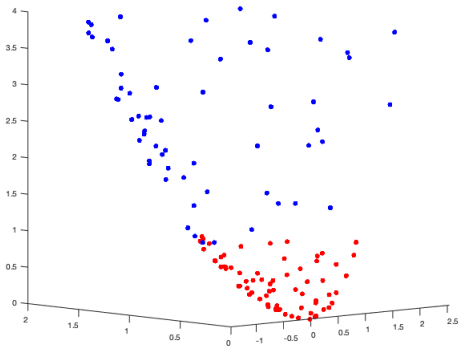

Figure 5: 3D image of teams with 3rd dimension added

Notice that now that is very easy to see the line that separates the teams. By transforming the data into data with more features, we make nonlinear data, linear. We can only view data that is 3-dimensional or less but we can use mathematical tricks with data in much higher dimensions
Sometimes we only care about the distance between players. In that situation we could use the squared distance, or any function to use as our metric, though some work much better than others. The choice of

distance metric is determined by the kernel. If there are 10 players, we use a 10x10 grid to store the distance between players. For example, the number in row 4 column 5 is the distance that player 4 is away from player 5. Anytime we calculate the distance between players in our algorithms we replace the distance with the value of the kernel between two players. Even if the numbers are not intuitive, a solution will be found.

4.2 Technical

To understand kernels, we will give a background on their development. This involves feature maps, inner products, and their applications to LDA and SVM. Given a dataset in $\mathbb{R}^{n*p}$ we can transform it to be in $\mathbb{R}^{m*p}$ by taking each data point and adding dimensions to it. For example suppose $\phi : (x, y) \to (x, y, x^2+y^2)$. ϕ is a feature map. There are infinite possibilities for feature maps and some make classification easier.
The LDA and SVM algorithms work the same way independently of the underlying data. Using feature maps, transform the original data $\mathbb{R}^n * p \xrightarrow{\phi} \mathbb{R}^{m*p}$ and then run LDA and SVM on this data to find the separating hyperplane (assuming feature map is still centered). Throughout both algorithms we use the dot product as the metric to compare data points. We measure the pairwise distance from any one point from another point using a distance matrix where $D_{ij} = \langle x_i, x_j \rangle$. Now we can find a distance matrix for our feature mapped data too, where $D_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$.
We really are only interested in the similarity between points. The intuition behind this is that similar points should have similar classifications. We use kernel functions to create a kernel matrix, which is just like the distance matrix. Some common choices for kernels are

$$\begin{gathered} \text{RBF, k(x,y)} = e^{\frac{||x-y||^2}{\alpha}} \\ \text{Polynomial, k(x,y)} = (x^T y + b)^d \\ \text{basic, k(x,y)} = x^T y \end{gathered}$$

These kernel functions are symmetric and non-negative which makes the kernel matrices spsd. Mercer's theorem tells us that any kernel matrix can be represented as an infinite sum of converging inner products.

$$\text{if K(x,y) is spsd, then } \exists \phi \text{ where K(x,y)} = \sum_{i=1}^{\infty} \phi(x)^T \phi(y)$$

Using a kernel matrix generated from a dataset allows us to calculate these feature map dot products without explicitly calculating the feature maps.

4.3 Kernel LDA

We are given a set of data, $\mathbb{D}$. We know that by applying a feature map, we increase the complexity of the problem ($\Phi : \mathbb{R}^{n*p} \to \mathbb{R}^{m*n}$ for $m >> n$). The feature space is $\mathbb{R}^{m*n}$. Theoretically, the feature maps could be infinite dimensional or extremely large, clearly an intractable problem for a modern computer to solve. First, we choose a kernel and create the kernel matrix K. Recall, for LDA the goal is to find the direction q that maximizes $\text{F(q)} = \frac{q^T S_b q}{q^T S_w q}$. If there exists a direction to project the data in the feature space then it must be some linear combination of the feature vectors i.e

$$q = \sum_{j=1}^{p} \alpha_j \phi(x^{(j)})$$

To classify a new point, one would calculate the feature map, project it onto q, and check the sign after selecting the split. This looks like

$$q^T \phi(x) = \sum_{j=1}^{p} \alpha_j \phi(x^{(j)})^T \phi(x)$$

Notice that this only relies on the inner products of $\phi(x)$. We replace any $\phi(x_1)^T \phi(x_2)$ with $k(x_1, x_2)$ by Mercer's theorem, and further simplify to

$$q^T \phi(x) = \sum_{j=1}^{p} \alpha_j k(x^{(j)}, x)$$

Notice that for any input x, we do not need to explicitly calculate the feature map as long as we define a kernel matrix. Thus, we only need to learn the parameters α_j. We create matrices M_1, M_2, and N to solve for the parameters. Let C_i and $|C_i|$ be the indices and the number of vectors in class i, respectively.

$$M_i = \{m_i^{(1)}, ..., m_i^{(p)}\} \text{ with } m_i^{(j)} = \sum_{l \in C_i} k(x^{(j)}, x^{(l)})$$
$$N = \sum_{i=1}^{2} K_i(\mathbb{I} - \frac{1}{|C_i|}\mathbb{J})K_i^T$$

K_i is the columns of K in class i and $\mathbb{J}$ is a matrix of ones. Let class 2 represent class -1. We solve for $\alpha = N^{-1}(M_2 - M_1)$.

4.4 Kernel SVM

Just like LDA, SVM can run on a dataset that has been transformed to a higher dimension. Recall our definition of G, with $G_{ij} = y^{(i)T}y^{(j)} = (c^{(i)}x^{(i)})^T(c^{(j)}x^{(j)}) = (x^{(i)T}c^{(i)})(c^{(j)x^{(j)}}) = c^{(i)}c^{(j)}x^{(i)T}x^{(j)}$. With the feature maps applied we reduce this to

$$G_{ij} = c^{(i)}c^{(j)}\phi(x^{(i)})^T\phi(x^{(j)}) = c^{(i)}c^{(j)}k(x^{(i)}, x^{(j)})$$

Again, notice there is no need to solve for the feature maps if we use the kernel k. Continuing with the algorithm we need to solve for q. Consider $q^T\phi(x)$

$$\text{q} = \sum_{j=1}^{p} \beta_j y^{(j)} \text{ so}$$
$$q^T\phi(x) = \sum_{j=1}^{p} \beta_j c^{(j)}\phi(x^{(j)})^T\phi(x) = \sum_{j=1}^{p} \beta_j c^{(j)}k(x^{(j)}, x) =$$
$$\text{k(q,x)}$$

q is not solved for, rather we know how to use it's projection for any new point. Plugging this into our formula for b we get

$$\text{b} = \frac{1}{m}\sum_{j=1}^{m}(k(q, x^{(i_j)}) - c^{(i_j)})$$

Finally, we classify a new instance, x, as $sign(k(q, x) - b)$.

5 LDA Extension

After finding the LDA direction (w), we find an orthogonal hyperplane that maximizes the classification accuracy. Using the training data we find the value that splits the projection the best.

$$\text{s} = \text{argmax}\{\sum_{j=1}^{p}(sign(w^Tx^{(j)} - s) == c^{(j)})\}$$

With the kernel projection

$$\text{s} = \text{argmax}\{\sum_{i=1}^{p}(sign(\sum_{j=1}^{p}\alpha_j k(x^{(j)}, x^{(i)}) - s) == c^{(j)})\}$$

Assuming that the optimal split lies in between the means of the two classes, we test how well the training set would be classified for each value. In the histograms below, the gray line shows the optimal split value found.

6 Experiments

We tested the algorithms on two different data sets. The first is the nonlinear dodgeball example. The dodgeball dataset consists of two teams with 100 players each. Each player is represented by a 2d vector, thus, the whole dataset $\mathbb{D} \in \mathbb{R}^{2*200}$. The algorithms were trained on 70% of the data and tested on the remaining 30%. The results for the kernel based LDA are shown below in figure 6.

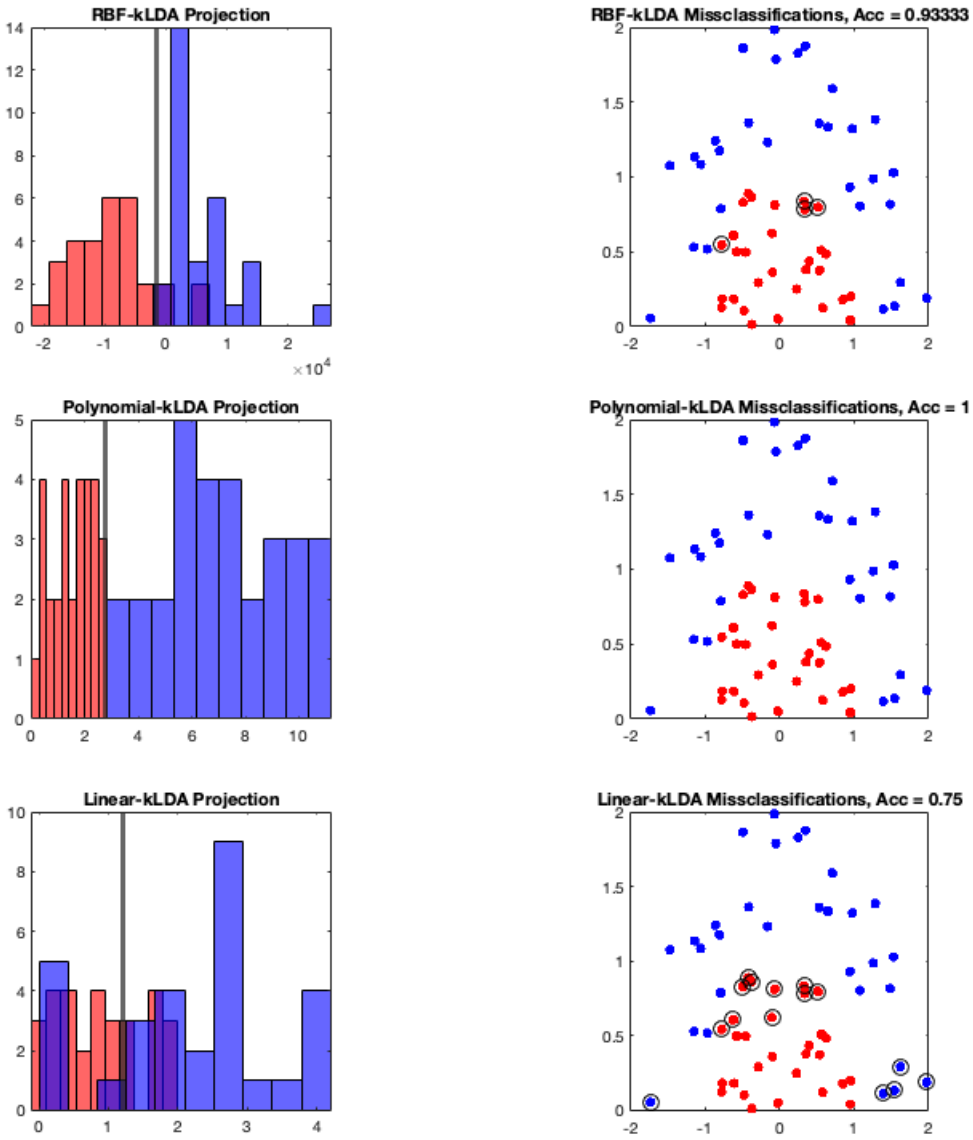

Figure 6: kLDA results on the test set

The histograms on the left show the algorithms projecting the test players onto the direction found in the feature map space. The vertical gray line shows the optimal split. The plots on the right side show the test set. The circled dots are the players that were misclassified by the algorithm. The linear kLDA shows how the algorithm would classify with just a line. The red circles are above the line and the blue circles are the players that fell below the line. Figure 7 shows the support vectors the SVM algorithm found.

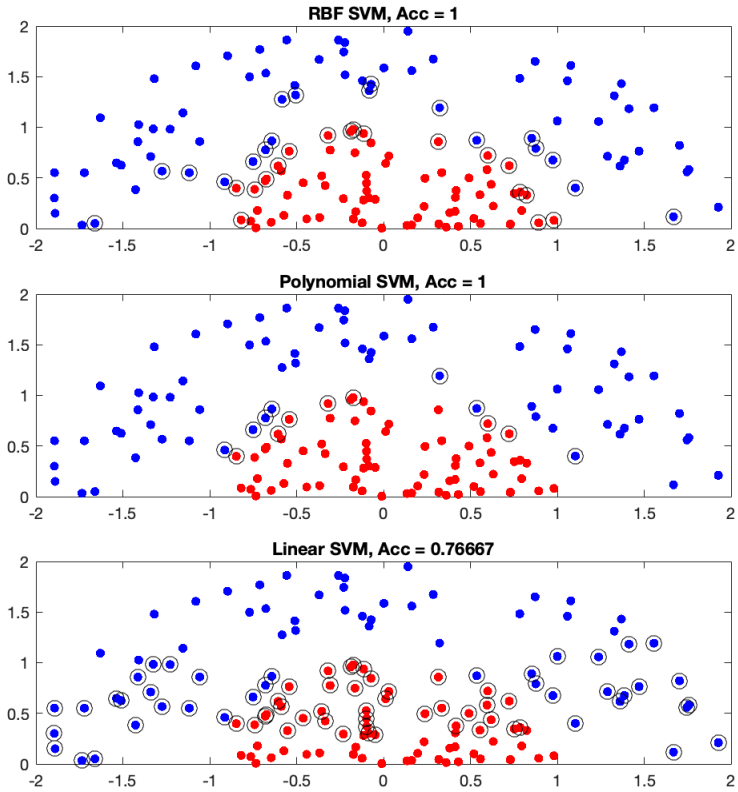

Figure 7: SVM support vectors

The circled points in figure 7 show which players were picked to be on the boundary (unlike in figure 6). The decision boundary is a line that threads between the red and blue support vectors. The linear-SVM shows what would happen without kernels- it finds a line straight through all the points.

The next dataset is of electrocardiogram readings of a heartbeat. Each reading is a timeseries of 96 data points resulting in a training set $\mathbb{D} \in \mathbb{R}^{96*100}$. The abnormal readings represent someone with a supraventricular premature heartbeat. The algorithms were trained on 100 instances and then tested on 100 new readings. Below shows an example of the readings to be classified.

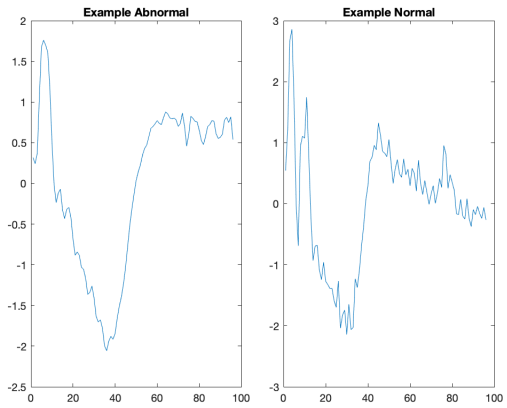

Figure 8: Example ECG readings

Each data point is its own dimension for the algorithms. We tried reducing the dimensionality of the data using principal component analysis (PCA). The performance on this data is shown below.

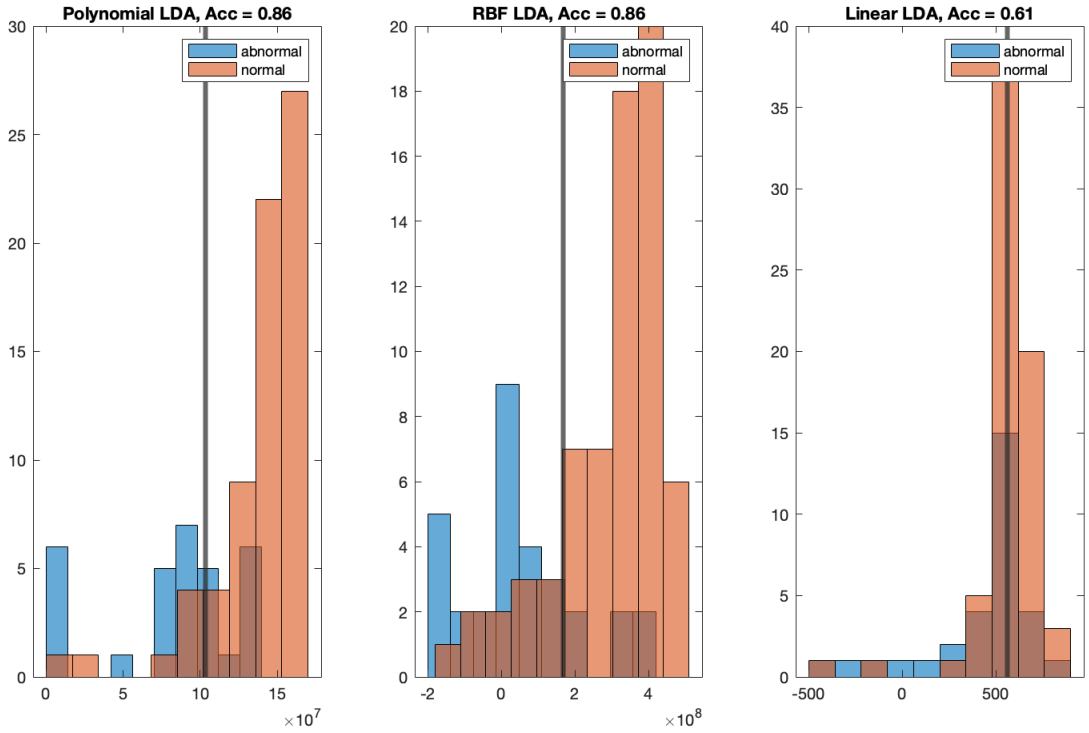

Figure 9: ECG Data projected

The rightmost graph shows that the LDA has a difficult time separating the two classes. The polynomial and RBF kernel's have much better separation even though the accuracy never reaches 90%. Figure 10 shows what happens with reduced the data from PCA.

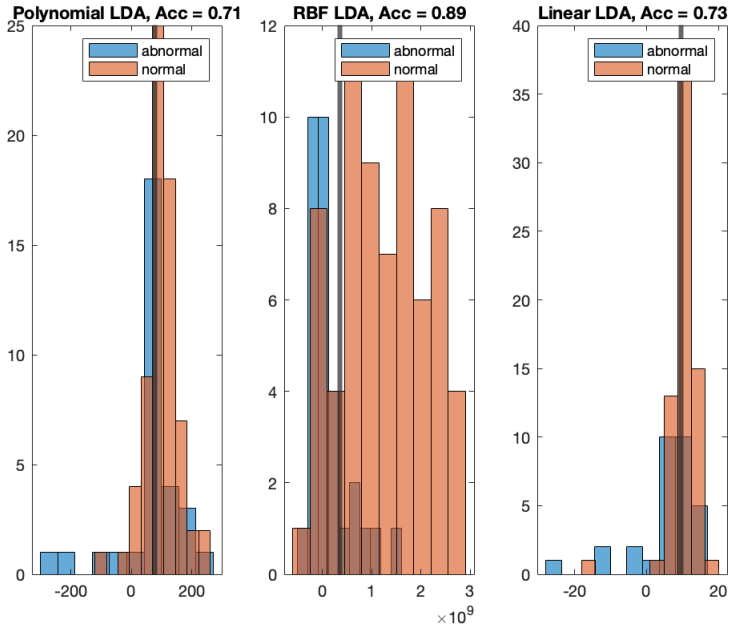

Figure 10: ECG Data projected with PCA dimensionality reduction

PCA finds a direction that maximizes the spread between the classes. We then reduce the noise from many of the dimensions taking only the relevant features. We reduced the data to 20 dimensions from 96 before classification. This made the polynomial kernel perform much worse and the RBF to perform slightly better. Below is a summary of the learning algorithms' performances.

	Accuracy (%)		
Algorithm	ECG	ECG-PCA	Dodgeball
RBF-LDA	86	**89**	93.3
RBF-SVM	64	72	**100**
POLY-LDA	86	71	**100**
POLY-SVM	**92**	85	**100**
LIN-LDA	61	73	75
LIN-SVM	82	80	76.7

7 Discussion

Neither LDA nor SVM is consistently the best choice. Both can perform well. Applying the dimensionality reduction may lead to better performance, but this will vary depending on the hyperparameters and data set. The use of kernels drastically increases performance and should be used in real data where linear separability is not guaranteed. Without kernels, LDA and SVM perform similarly.

8 References

[**1**] Calvetti, Daniela, and Erkki Somersalo. Mathematics of Data Science a Computational Approach to Clustering and Classification. Siam, Society for Industrial and Applied Mathematics, 2021.
[**2**] Roth, Volker, and Volker Steinhage. 1999. Advances in Neural Information Processing Systems, 12. "Nonlinear Discriminant Analysis Using Kernel Functions".

[**3**] S. Mika, G. Ratsch, J. Weston, B. Scholkopf and K. R. Mullers, "Fisher discriminant analysis with kernels," Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No.98TH8468), 1999, pp. 41-48, doi: 10.1109/NNSP.1999.788121.