Birla Institute of Technology and Science Pilani, Hyderabad Campus

2nd Semester 2023-24

27.02.2024

BITS F464: Machine Learning

LINEAR DISCRIMINANT FUNCTIONS FOR CLASSIFICATION

Chittaranjan Hota, Sr. Professor
Dept. of Computer Sc. and Information Systems
hota@hyderabad.bits-pilani.ac.in

(Acknowledgement: Few Images from Bishop’s Text)

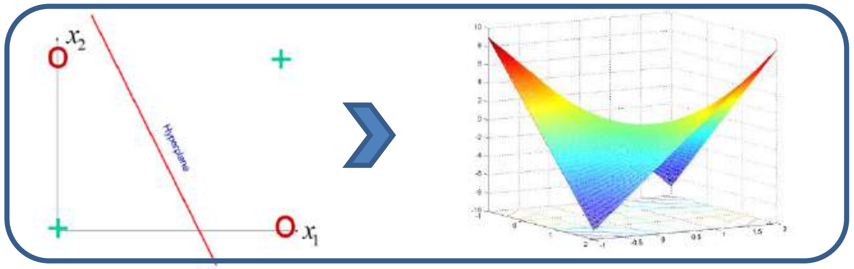
Recap

- Regression with Scalar Input (Univariate)
  - weight = 2 + 1.5 * height
- Regression with Vector Input (Multi-variate)
  - BMI = 18 + 1.5 (diet score) + 1.6 (male) + 4.2 (age > 20)
- Least squares method (Loss function)

$$S = \sum_{i=1}^{n} r_i^2$$

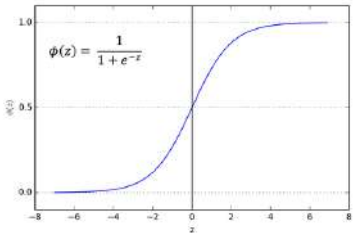
- Basis functions for handling non-linearity in input

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$



- Gradient descent

$$\theta_1 = \theta_1 - \alpha \frac{\mathbf{d}\, \mathbf{J}(\theta_1)}{\mathbf{d}\, \theta_j}$$



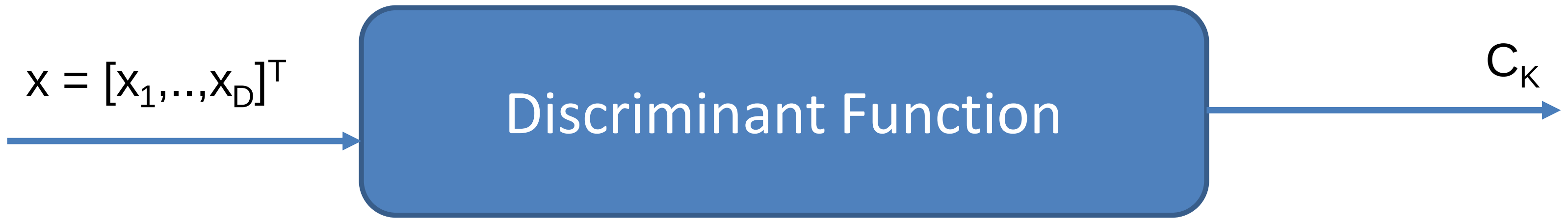
Logistic Regression

Linear Discriminant Functions: Applications

- Fisher’s Linear Discriminant Analysis for reducing the number of features required for Face Recognition.
- Classifying patient’s disease state as Mild, Moderate or Severe.
- Identifying the type of customers who might buy a particular product.

Linear Discriminant Functions

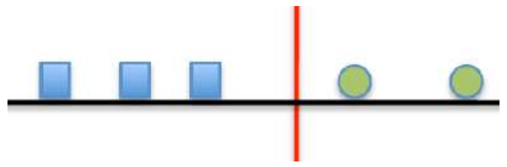
- Used to discriminate between two or more classes based on a set of predictor variables.

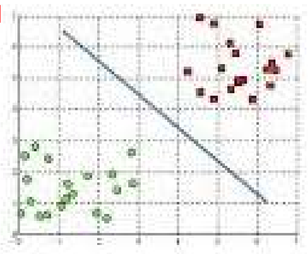


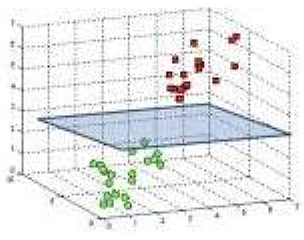
- Learns the mapping between feature vector and class labels.

- Does it not create a decision boundary?

Hyperplanes


(1-D, a point/ threshold)


(2-D, a line)

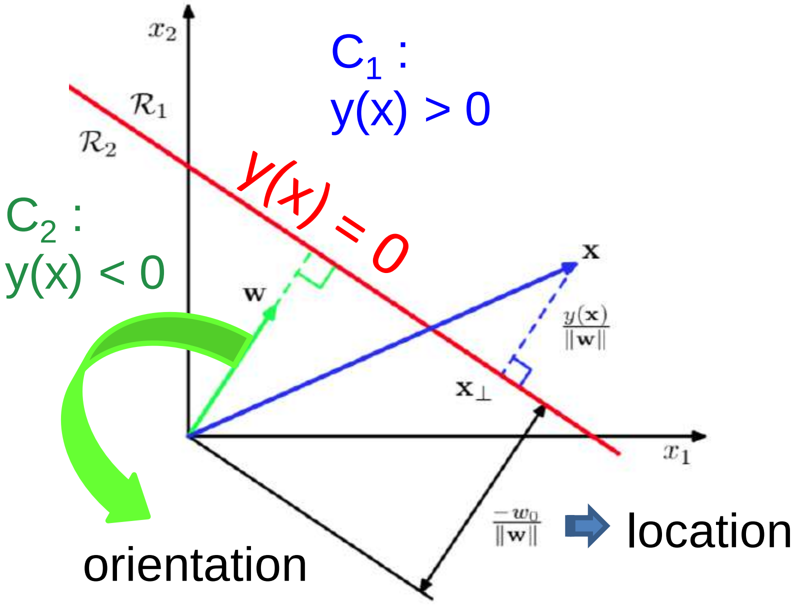

(3-D, a plane)

Logistic Regression may be unstable for well separated classes and few examples. Why?

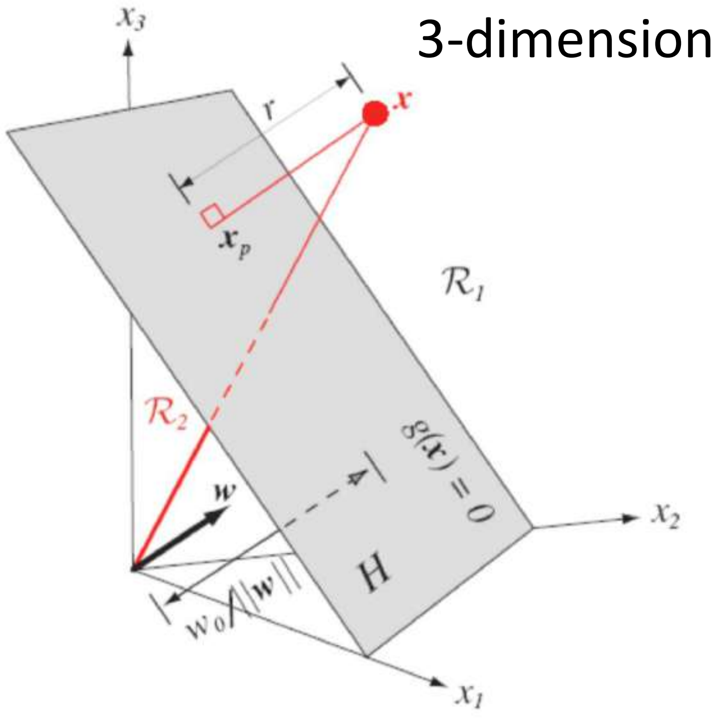
Two-class Linear Discriminant Functions (K=2)

- $y(x) = w^T x + w_0 = \sum_{i=1}^{d} w_i x_i + w_0$

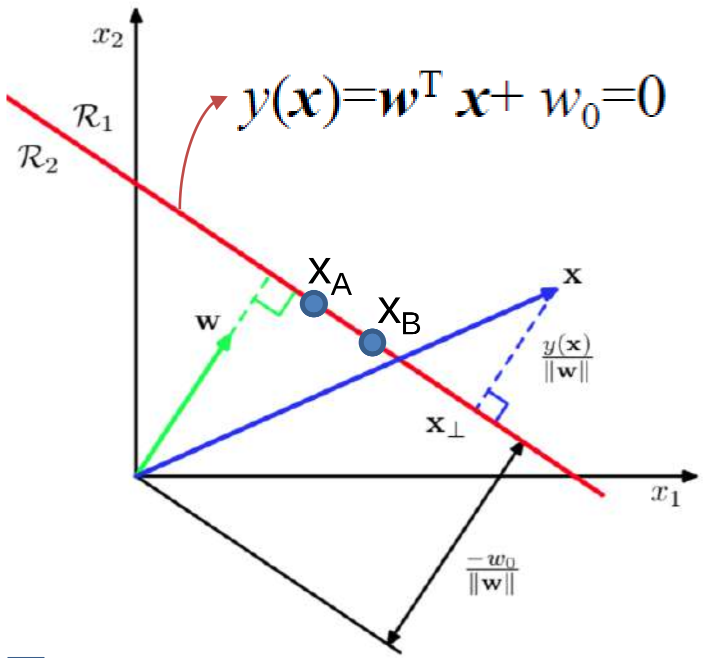
Where, w^T is the weight vector and w_0 is the bias. The negative of bias (i.e. $-w_0$) sometimes is called as threshold.



(geometry of LDF in 2-dimension)



Distance of Origin to Decision Surface (Bias: w_0)



x_2
$\mathcal{R}_1$
$\mathcal{R}_2$
$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$
x_A
x_B
$\mathbf{x}$
$\mathbf{w}$
$\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$
$\mathbf{x}_\perp$
x_1
$\frac{-w_0}{\|\mathbf{w}\|}$

- Let x_A and x_B be points that lie on the decision surface.

$y(x_A) = y(x_B) = 0$

$w^T x_A + w_0 = w^T x_B + w_0 = 0$

$w^T (x_A - x_B) = 0$

$x_A - x_B$ is an arbitrary vector parallel to the line.

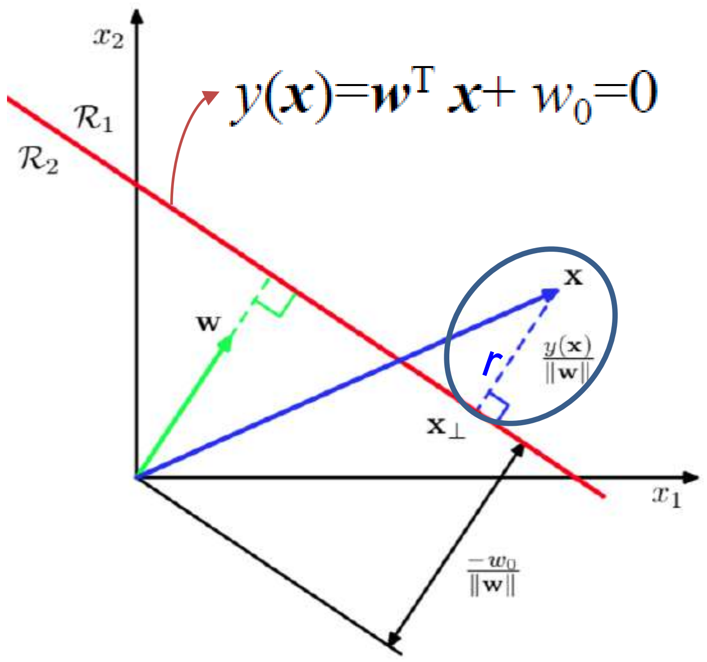
Hence, w is orthogonal to every vector lying on the decision surface. orientation

If x is a point on the decision surface, $y(x) = 0 \rightarrow w^T x = -w_0$

So, the normal distance from origin to decision surface:

$$\frac{\mathrm{w}^T \mathrm{x}}{\|\mathrm{w}\|} = -\frac{w_0}{\boxed{\|\mathrm{w}\|}} \Rightarrow \sqrt{w_1^2 + .. + w_{M-1}^2}$$

Distance of a point ‘x’ to the Decision surface (r)



Let ‘x’ be an arbitrary point and $\mathbf{x}_\perp$ be it’s orthogonal projection on the decision surface.

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$ by vector addition

Second term is a normalized vector to the decision surface, which is collinear with ‘w’.

As $\frac{\mathrm{w}}{\|\mathrm{w}\|} = 1$, we need to scale it by r.

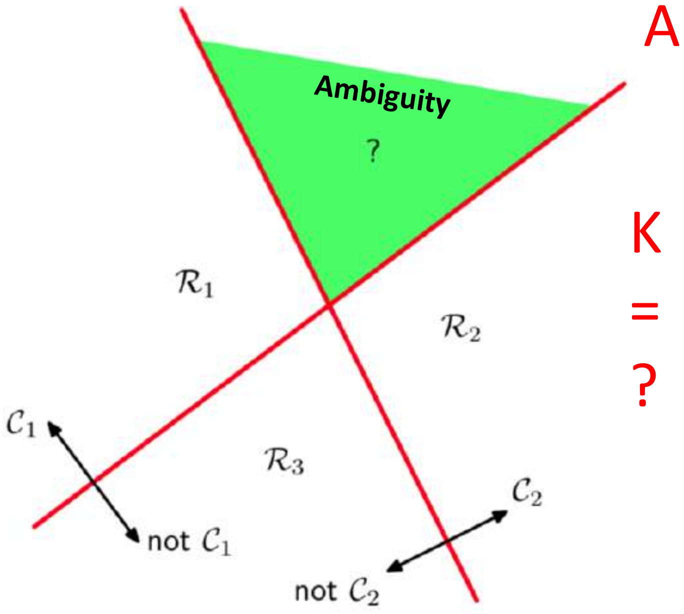
As $y(\mathbf{x}_\perp) = 0$ and $w^T w = ||w||^2$ ➡ $y(x) = w^T x + w_0 = w^T (\mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + w_0$

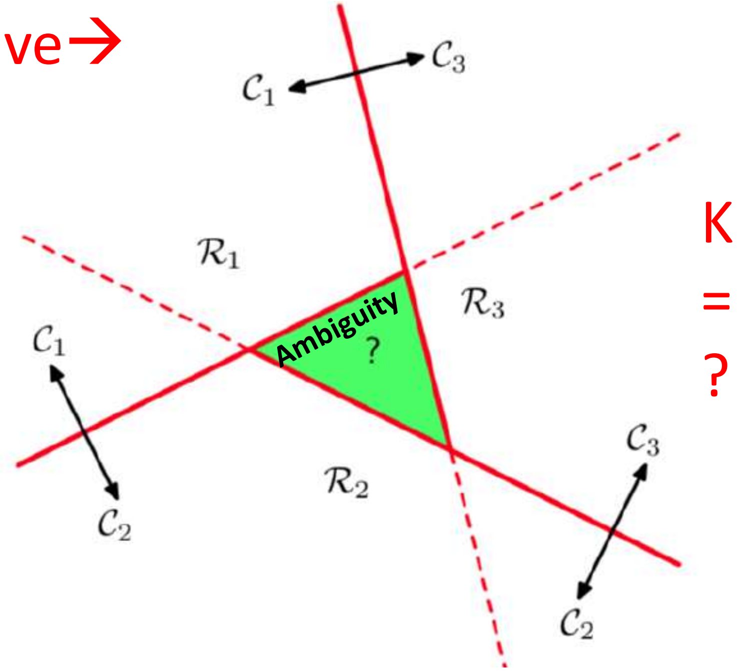
➡ $\underline{w^T \mathbf{x}_\perp + w_0} + r\frac{w^T w}{\|\mathbf{w}\|} = 0 + r\frac{||w||^2}{\|\mathbf{w}\|} = r.||w||$ ➡ $r = y(x) / ||w||$

Multi-class Linear Discriminant Functions (K>2)

Approach 1: By combining a number of two-class discriminant functions.

Alternative→

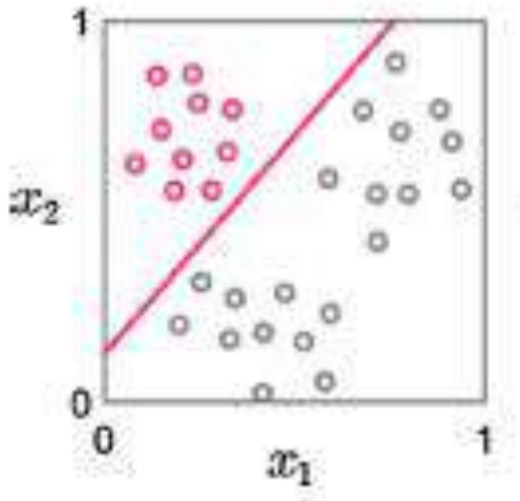


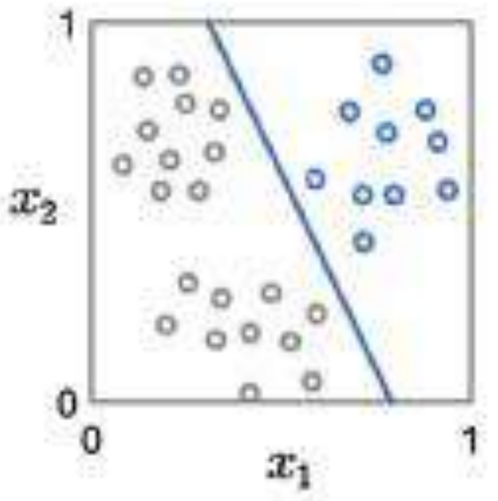


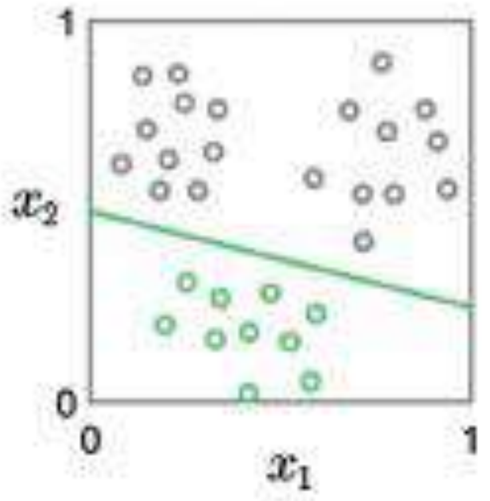
(K-1 classifiers with each one separating points in a particular class C_k from points not in that class) one-versus-the-rest

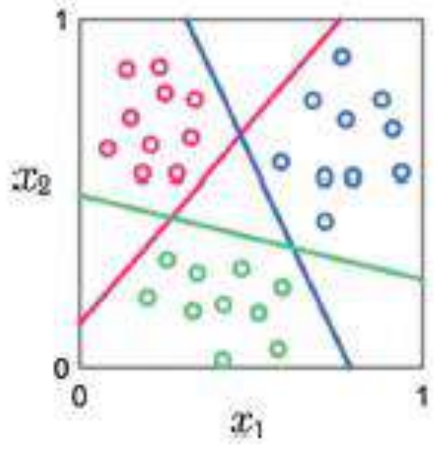
(K (K-1)/2) classifiers with one for every possible pair of classes. Each point is classified according to a majority vote. one-versus-one

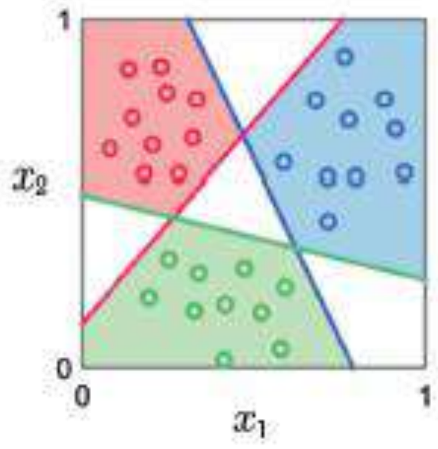
Another Example…











$$b_c + x_p^T w_c > 0$$

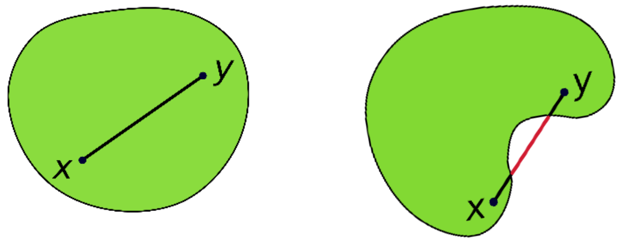
$$b_j + x_p^T w_j < 0,\ j = 1 \ldots c,\ j \mathrel{!}= c$$

$$y = \boxed{\operatorname*{argmax}_{j = 1, \ldots c}}\ b_j + x^T w_j$$

global maximum

Solution: Using K-discriminant functions

- Building a single K-class discriminant comprising K-linear functions of the form:
  - $y_k(x) = w_k^T x + w_{k0}$

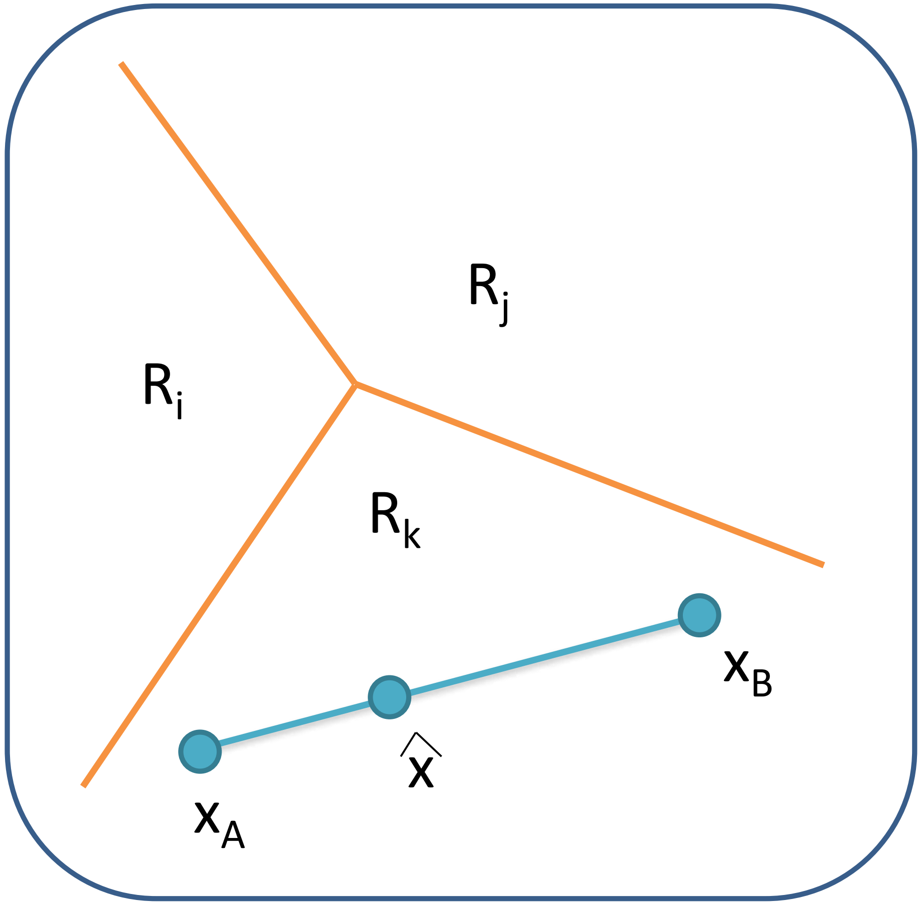


- Then, assigning a point x to class C_k if $y_k(x) > y_j(x), \forall j \neq k$.
- The decision boundary between C_k and C_j : $y_k(x) = y_j(x)$
- Defined by: $(w_k - w_j)^T x + (w_{k0} - w_{j0}) = 0 \longrightarrow$ Same as 2-class
- Hence, same geometrical properties apply.

Decision regions of such discriminants are always singly connected and convex.

Proof of Convexity Next…

Proof of Convexity of Decision Region



$$\hat{x} = \lambda x_A + (1-\lambda) x_B$$

Where, $0 \leq \lambda \leq 1$

From the Linearity of Discriminant functions:

$$y_k(\hat{\mathbf{x}}) = \lambda y_k(\mathbf{x}_A) + (1-\lambda) y_k(\mathbf{x}_B)$$

As X_A and X_B lie inside R_K :

$y_k(x_A) > y_j(x_A)$ and $y_k(x_B) > y_j(x_B)$

$\forall\, j \neq k$ ➤ $y_k(\hat{\mathbf{x}}) > y_j(\hat{\mathbf{x}})$ ➤ $\hat{x}$ Lies in R_K

$\hat{x}$ must also lie in R_K $\longrightarrow$ R_K is Singly Connected and Convex

Multi-class Classification using LDA (sklearn)

```
from sklearn.datasets import load_wine
dt = load_wine()
X = dt.data
y = dt.target
```

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)

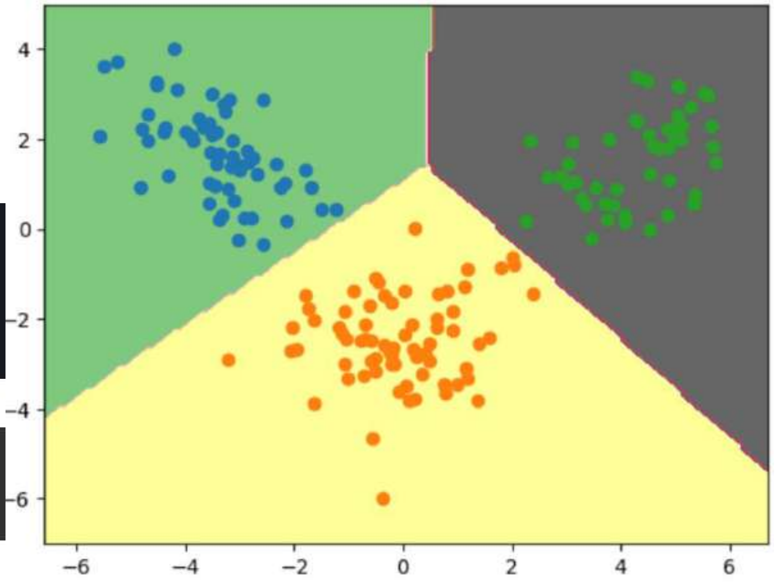
lda.fit(X_train,y_train)

```
y_pred = lda.predict(X_test)
print(accuracy_score(y_test,y_pred))
```

```
0.9814814814814815
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[13,  0,  0],
       [ 0, 26,  0],
       [ 0,  1, 14]])
```



```
num_records = wine_data.data.shape[0]
print(num_records)
```

Alcohol, magnesium, hue, proline, ...

Least Squares for Classification

- Straightforward way to adapt regression techniques for classification tasks.

- How do we compute y(x), and w_0, w_1, w_2, …w_d ?
- Each class C_k is described by its own linear model:
    - $y_k(x) = w_k^T x + w_{k0}$ (where x and w have D dimensions each)
- We can group these together using a vector notation:

$$\mathbf{y}(\mathbf{x}) = \underbrace{\widetilde{\mathbf{W}}^{\mathrm{T}}}\boxed{\widetilde{\mathbf{x}}} \longrightarrow$$

Augmented input vector $(1, \mathbf{x}^{\mathrm{T}})^{\mathrm{T}}$

A Parameter matrix whose kth column is a D+1 dimensional vector: $\widetilde{\mathbf{w}}_k = (w_{k0}, \mathbf{w}_k^{\mathrm{T}})^{\mathrm{T}}$

A new input x is then assigned to a class for which the output Is largest. $y_k = \widetilde{\mathbf{w}}_k^{\mathrm{T}} \widetilde{\mathbf{x}}$ Get $\tilde{\mathbf{w}}$ by minimizing the Sum-of-squares.

Why do we need to augment the input?

- To better capture the relationship between input features and classes being classified.
- We introduce an additional feature that always has a constant value (usually 1), which effectively allows the linear function to have an intercept term, or bias.
- The bias allows the decision boundary to be offset from the origin, making the model more flexible and better able to fit the data.
- Without the bias term, the decision boundary would always pass through the origin, which might not be appropriate for all datasets.

Original input vector: [0.2, 0.4, 0.5] $\rightarrow$ Augmented to: [1, 0.2, 0.4, 0.5]

Minimizing sum-of-squares error func

Let there be a training dataset $\{x_n, t_n\}$ where n = 1, …N

Define a matrix T whose nth row is the vector t_n^T and matrix $\widetilde{\mathbf{X}}$ whose nth row is: $\widetilde{\mathbf{x}}_n^{\mathrm{T}}$.

Then, the sum-of-squares error function can be written as:

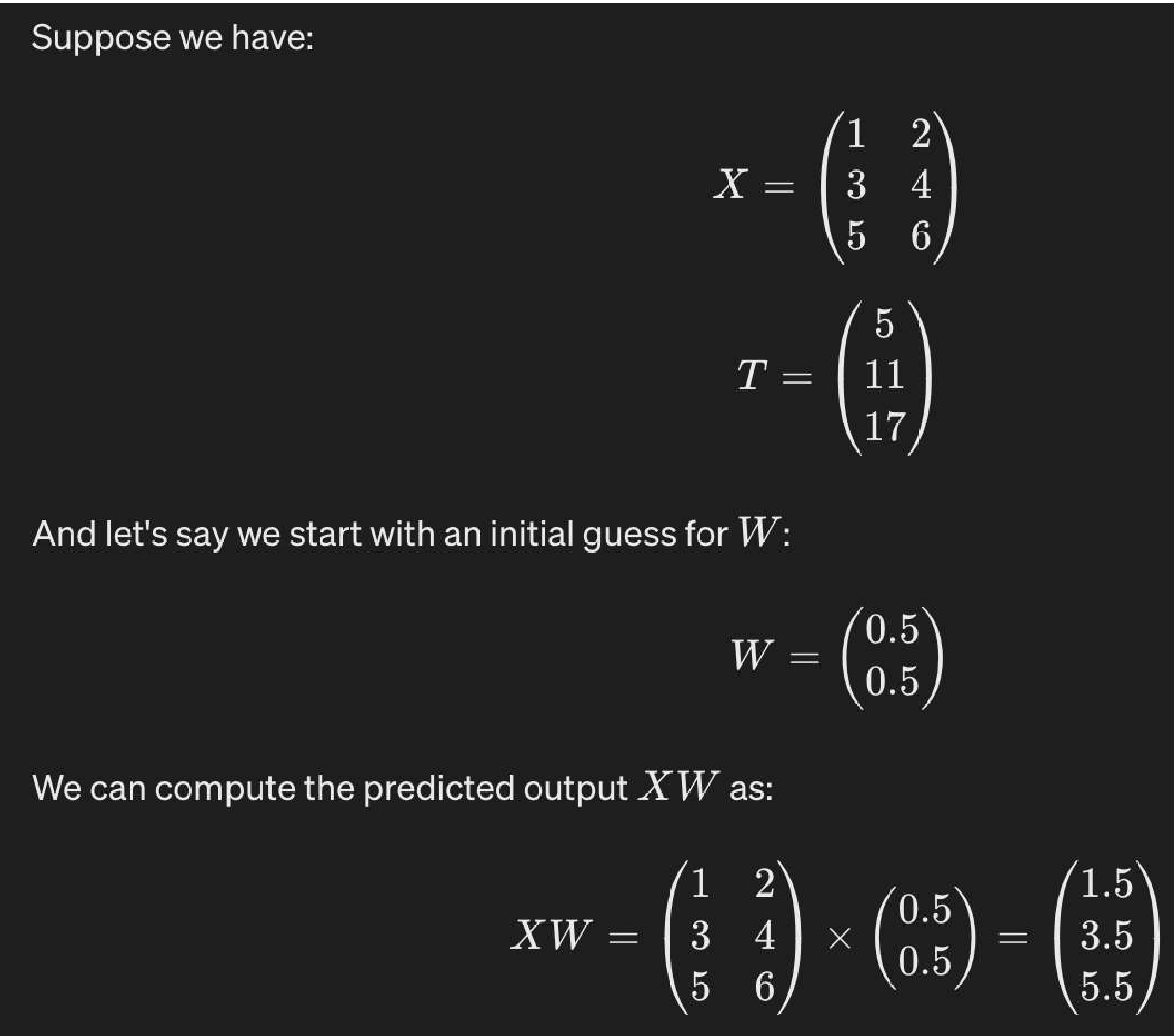
$$E_D(\widetilde{\mathbf{W}}) = \frac{1}{2}\mathrm{Tr}\left\{\underset{\big\downarrow}{\boxed{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})^{\mathrm{T}}(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})}}\right\}$$

Multiplying a matrix with its's transpose results in a square matrix.

Taking a Trace of this square matrix (sum of the elements on the main diagonal)

½ is a scaling factor. Squaring the errors naturally results in terms that are proportional to the squares of the individual errors, and the factor of 1/2 helps cancel out the coefficient that arises when you differentiate with respect to the parameters you're trying to optimize.

An Example LSE Computation...



Suppose we have:

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

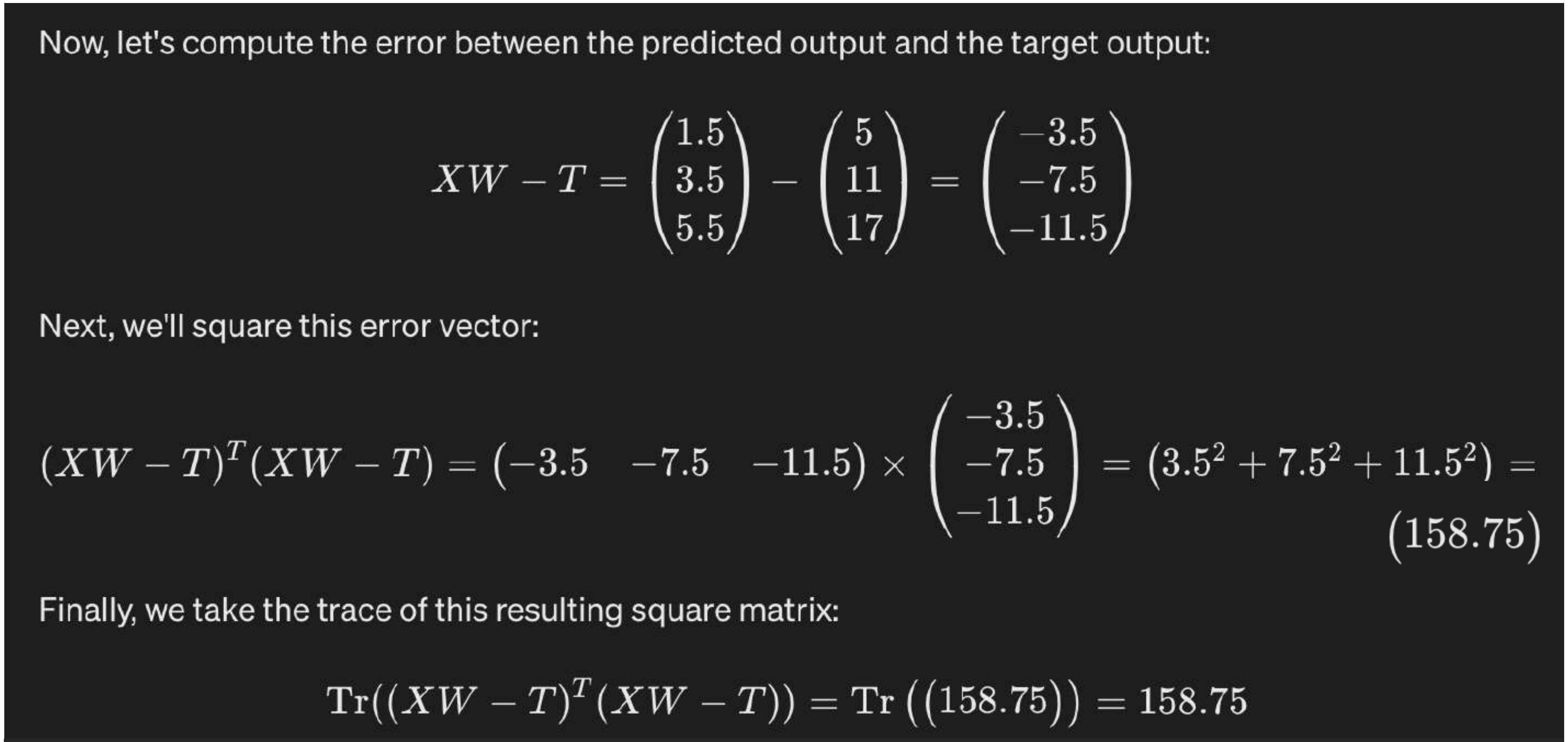
$$T = \begin{pmatrix} 5 \\ 11 \\ 17 \end{pmatrix}$$

And let's say we start with an initial guess for W:

$$W = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

We can compute the predicted output XW as:

$$XW = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \times \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 3.5 \\ 5.5 \end{pmatrix}$$



Now, let's compute the error between the predicted output and the target output:

$$XW - T = \begin{pmatrix} 1.5 \\ 3.5 \\ 5.5 \end{pmatrix} - \begin{pmatrix} 5 \\ 11 \\ 17 \end{pmatrix} = \begin{pmatrix} -3.5 \\ -7.5 \\ -11.5 \end{pmatrix}$$

Next, we'll square this error vector:

$$(XW - T)^T(XW - T) = \begin{pmatrix} -3.5 & -7.5 & -11.5 \end{pmatrix} \times \begin{pmatrix} -3.5 \\ -7.5 \\ -11.5 \end{pmatrix} = (3.5^2 + 7.5^2 + 11.5^2) = (158.75)$$

Finally, we take the trace of this resulting square matrix:

$$\mathrm{Tr}((XW - T)^T(XW - T)) = \mathrm{Tr}\left((158.75)\right) = 158.75$$

So, here the least-square-error (LSE) is given as:

(½) 158.75 = 79.375

→ Quantifies the total squared error between the predicted and target outputs. Here, the goal is to minimize LSE by adjusting W.

Dividing by 2 is for Mathematical convenience. Derivative of a function will be simpler if we have ½ in the front.

Acknowledgement: ChatGPT

Minimizing Sum-of-Squares

$$E_D(\widetilde{\mathbf{W}}) = \frac{1}{2}\mathrm{Tr}\left\{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})^{\mathrm{T}}(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})\right\}$$

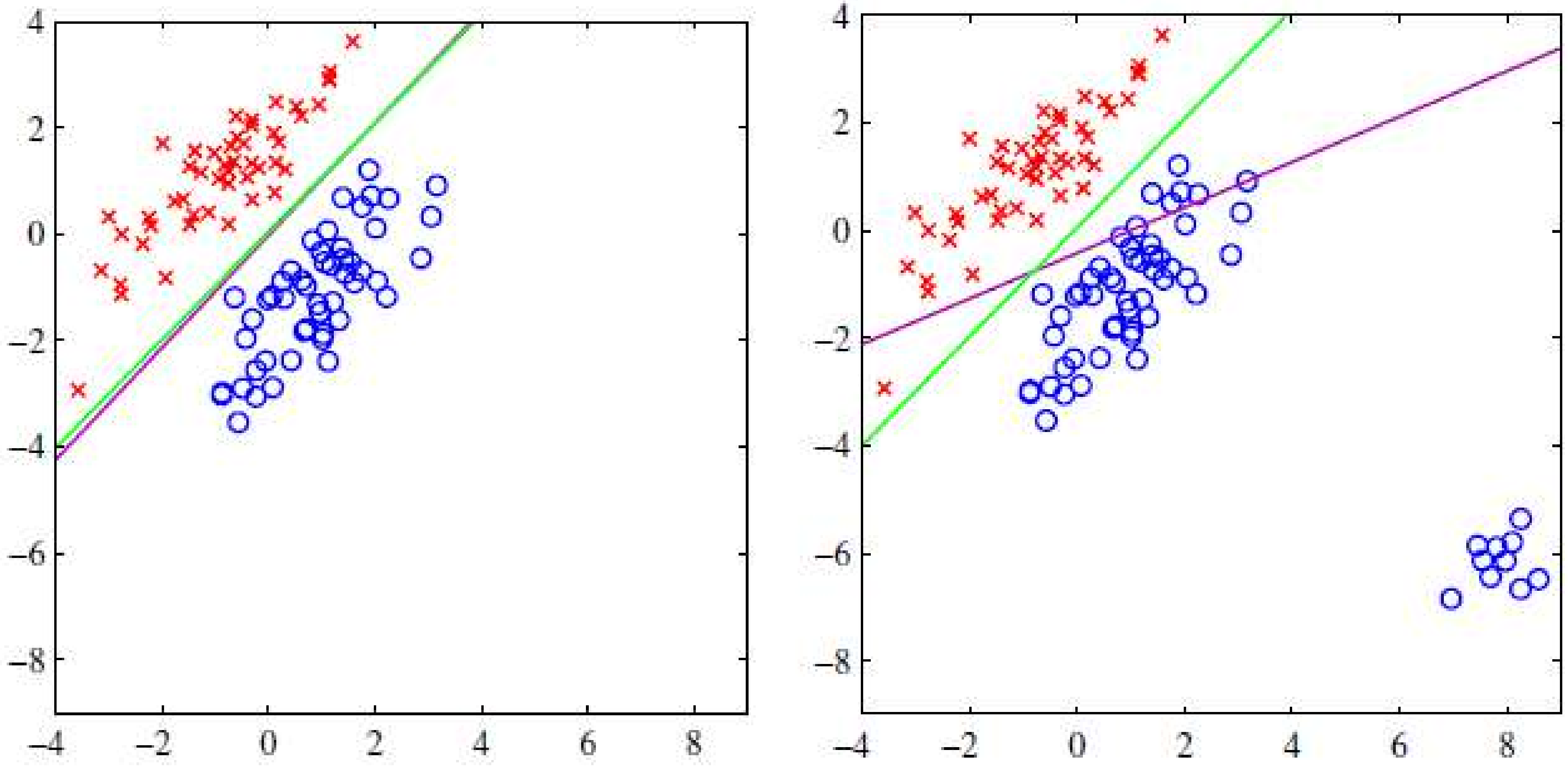
Setting the derivative with respect to $\widetilde{\mathbf{w}}$ to 0 and rearranging, we obtain the solution for $\widetilde{\mathbf{w}}$ as:

$$\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T} = \widetilde{\mathbf{X}}^{\dagger}\mathbf{T}$$ Where, $\widetilde{\mathbf{X}}^{\dagger}$ is pseudo inverse of $\widetilde{\mathbf{X}}$

We then obtain the discriminant function in the form:

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{x}} = \mathbf{T}^{\mathrm{T}}\left(\widetilde{\mathbf{X}}^{\dagger}\right)^{\mathrm{T}}\widetilde{\mathbf{x}}$$

Least-squares: highly sensitive to outliers



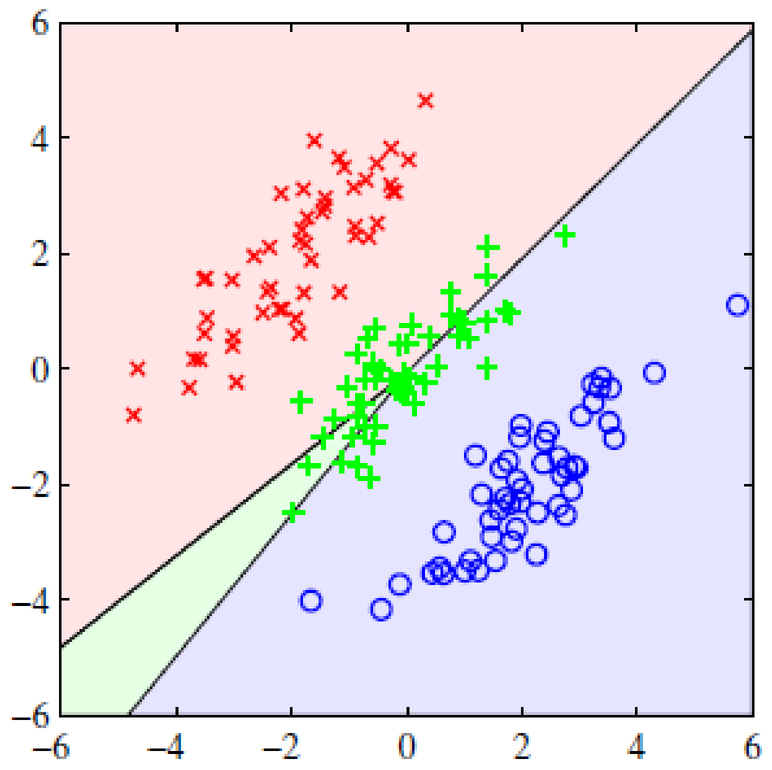
Magenta: Least squares, Green: Logistic regression   Robustness ⇩

The sum-of-squares error function penalizes predictions that are ‘too correct’ in that they lie a long way on the correct side of the decision boundary.

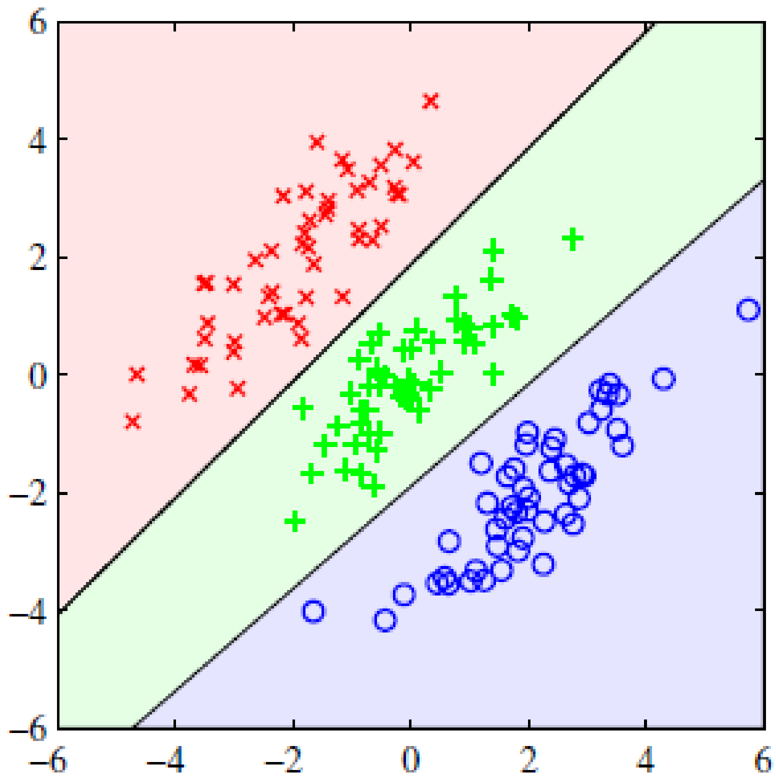
Least squares: more severe problems

Certain datasets: Unsuitable

3-classes, 2-D space, synthetic data



(Least-squares classification)

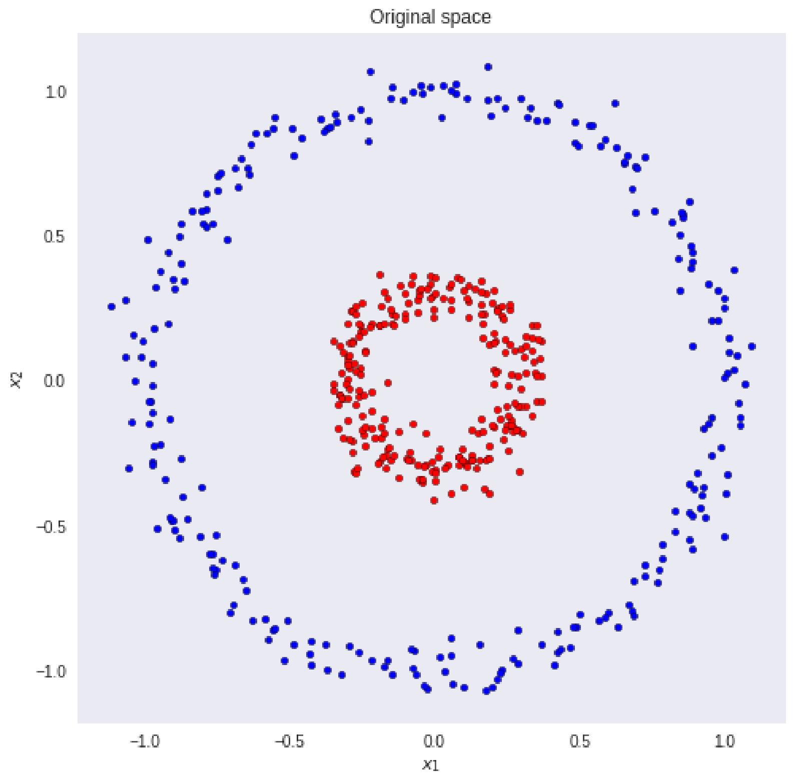


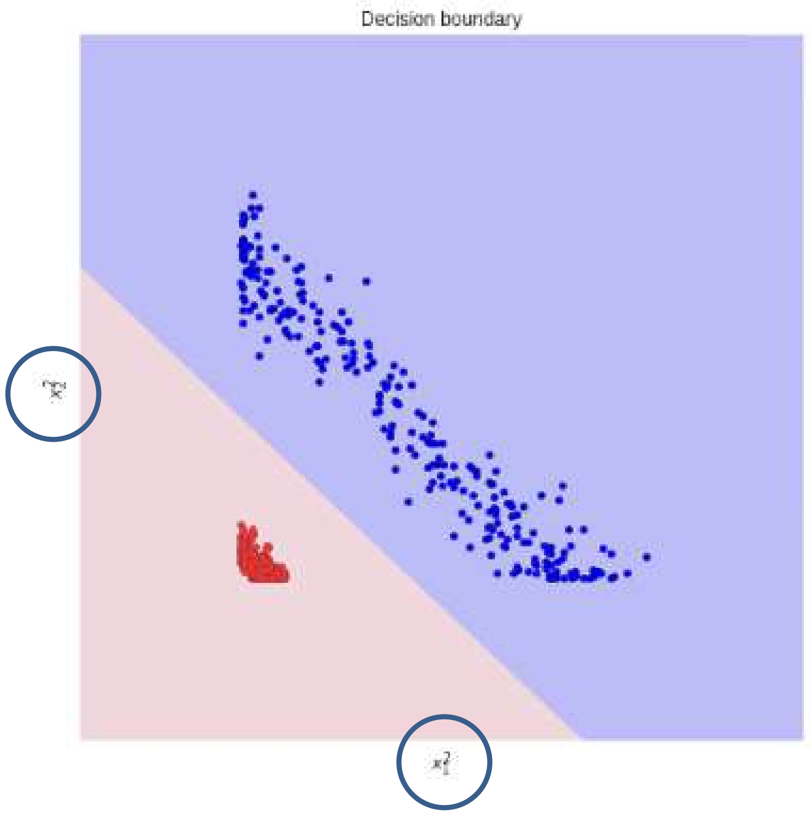
(Logistic Regression classification)

The region of input space assigned to the green class is too small and so most of the points from this class are misclassified.

Fisher's Linear Discriminant: Motivation

- Why do we need it?





Question: How difficult are these transformations to figure out?

Image source: https://sthalles.github.io/

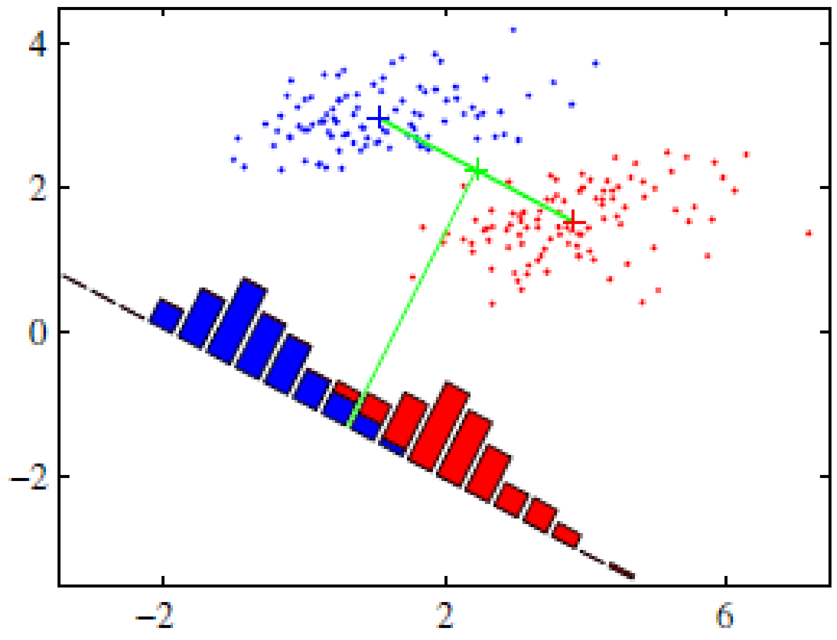
Fisher's Linear Discriminant

Ronald A. Fisher

- View classification in terms of dimensionality reduction
  - Project D-dimensional input vector x into one dimension using: $y = w^T x$
  - Place threshold on y to classify $y >= -w_0$ as class C_1 else class C_2
  - We get a standard linear classifier
- Classes well-separated in D-dimension space may strongly overlap in 1-dimension
  - Adjust component of the weight vector w
  - Select projection to maximize class-separation

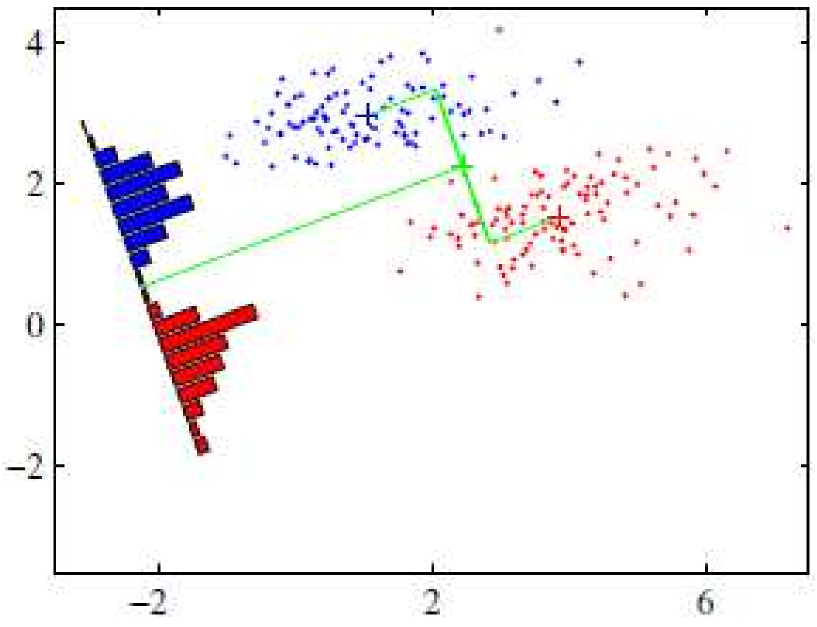
FLD seeks to maximize the ratio of between-class variance to within-class variance, thus maximizing class discrimination.

An illustration of Fisher's LDF



(Projection onto the line joining the class means)

What is the degree of class overlap?



(Projection based on Fisher's Linear discriminant function)

Is the class separation improved?

Maximizing Mean Separation

- Let us consider a two-class problem with N_1 points of C_1 class and N_2 points of C_2 class
- Mean vectors: $m_1 = \frac{1}{N_1} \sum_{n \varepsilon C_1} x_n$ $m_2 = \frac{1}{N_1} \sum_{n \varepsilon C_2} x_n$
- Choose w to best separate class means:
- Maximize $m_2 - m_1 = w^T(m_2 - m_1)$, where $m_k = w^T m_k$ is the mean of the projected data from class C_k
- Can be made arbitrarily large by increasing the magnitude of w:
  - We could have w to be of unit length i.e. $\sum_i w_i^2 = 1.$
  - Using a Lagrange multiplier, maximize

$\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$ There is still a problem with this approach…

Illustration of the problem

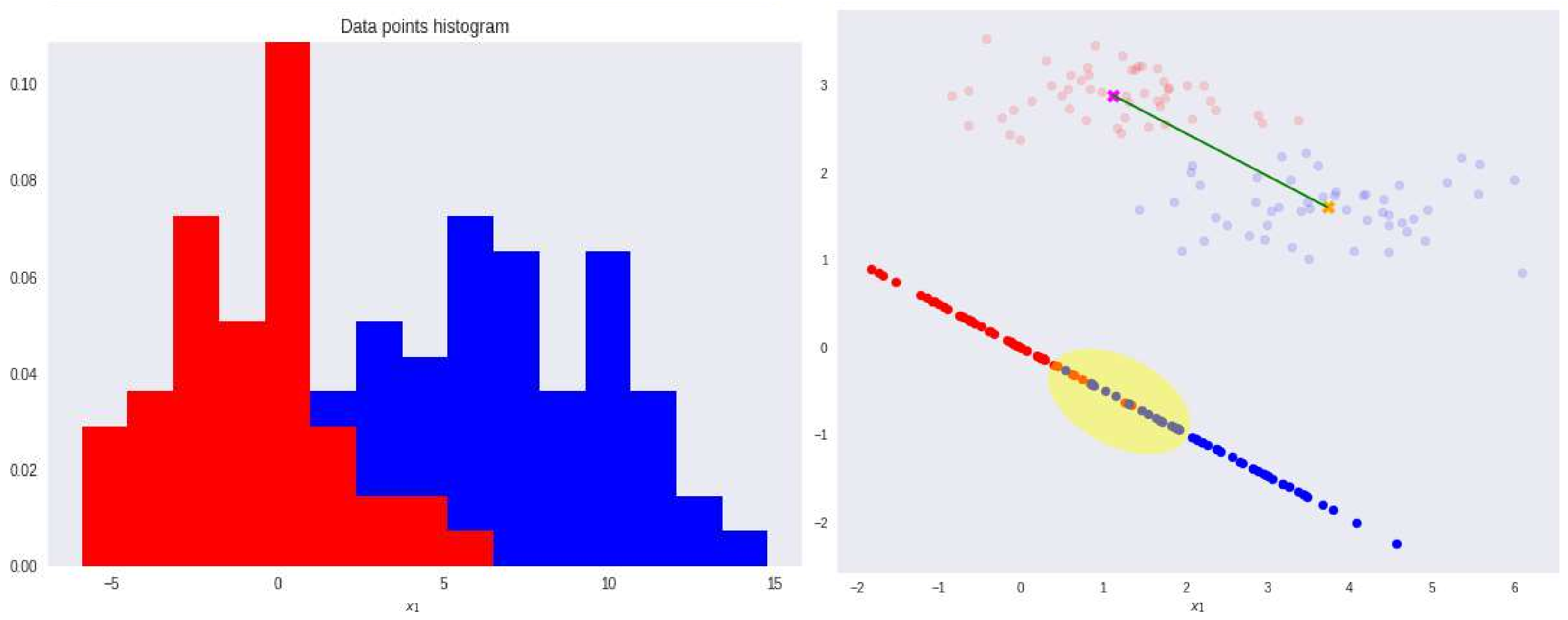


Image source: https://sthalles.github.io/

After re-projection, the data exhibit some sort of class overlapping - shown by the yellow ellipse on the plot.

This difficulty arises from the strongly non-diagonal co-variances of the class distributions.

Minimizing Variance and Optimizing

- Project D-dimensional input vector x into one dimension using: $y_n = w^T x_n$
- The within-class variance of the transformed data from class C_k is given by:

$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$$

- Total within-class variance for the whole dataset is: $s_1^2 + s_2^2$
- Fisher's criterion: $J(\mathbf{w}) = (m_2 - m_1)^2 / s_1^2 + s_2^2$

Rewriting (to make the dependence on w explicit: $J(\mathbf{w}) = \mathbf{w}^T S_B \mathbf{w} \,/\, \mathbf{w}^T S_W \mathbf{w}$

Where, $S_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$ is the between-class covariance matrix &

$S_W = \Sigma_{n \in C1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \Sigma_{n \in C2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$ the within-class covariance matrix.

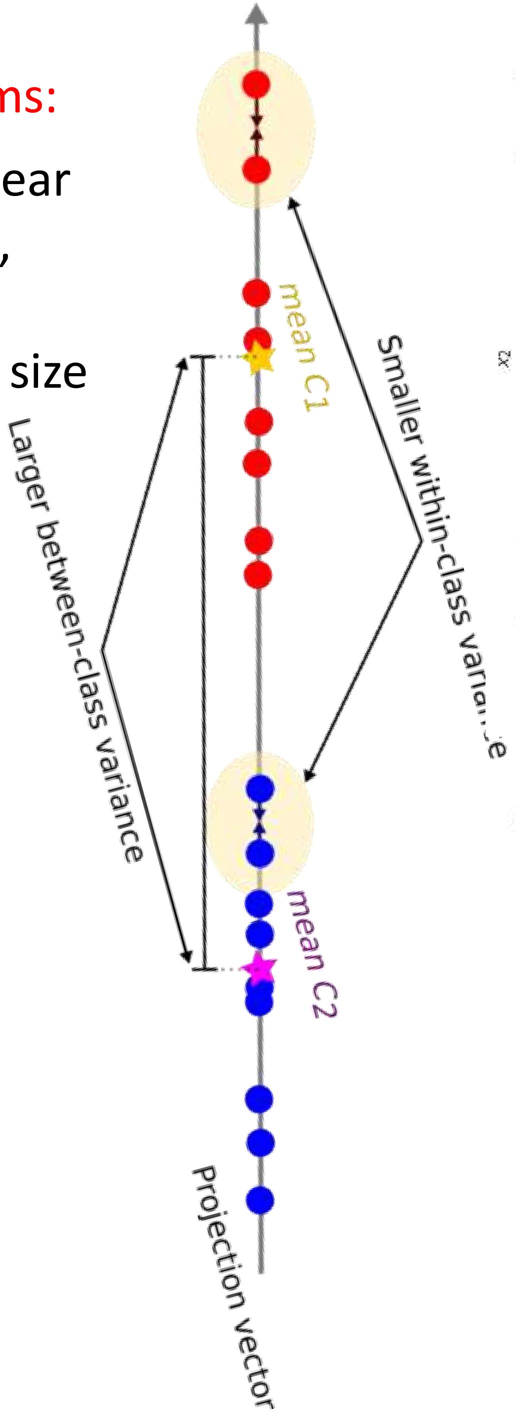
Differentiating with respect to w, J(w) is maximized when: $(\mathbf{w}^T S_B \mathbf{w}) S_W \mathbf{w} = (\mathbf{w}^T S_W \mathbf{w}) S_B \mathbf{w}$

Dropping scalar factors, and noting S_B is in the same direction as $m_2 - m_1$ and multiplying both the sides by S_W^{-1} : $\boxed{w \;\alpha\; S_W^{-1}(m_2 - m_1)}$ $\gg$ Fisher's LD

Illustration with Fisher's LDF

Problems:

Non-linear models, Small sample size



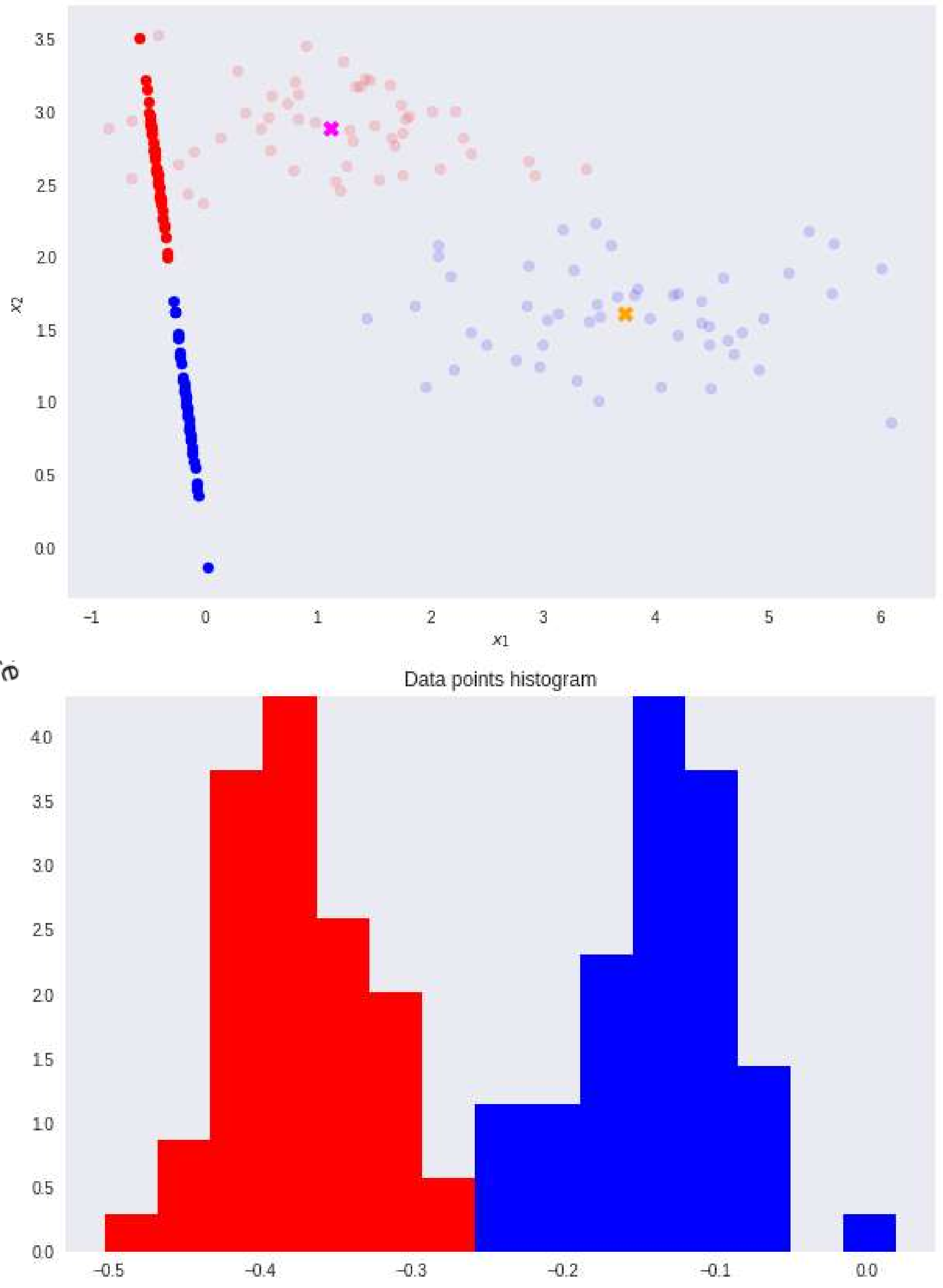
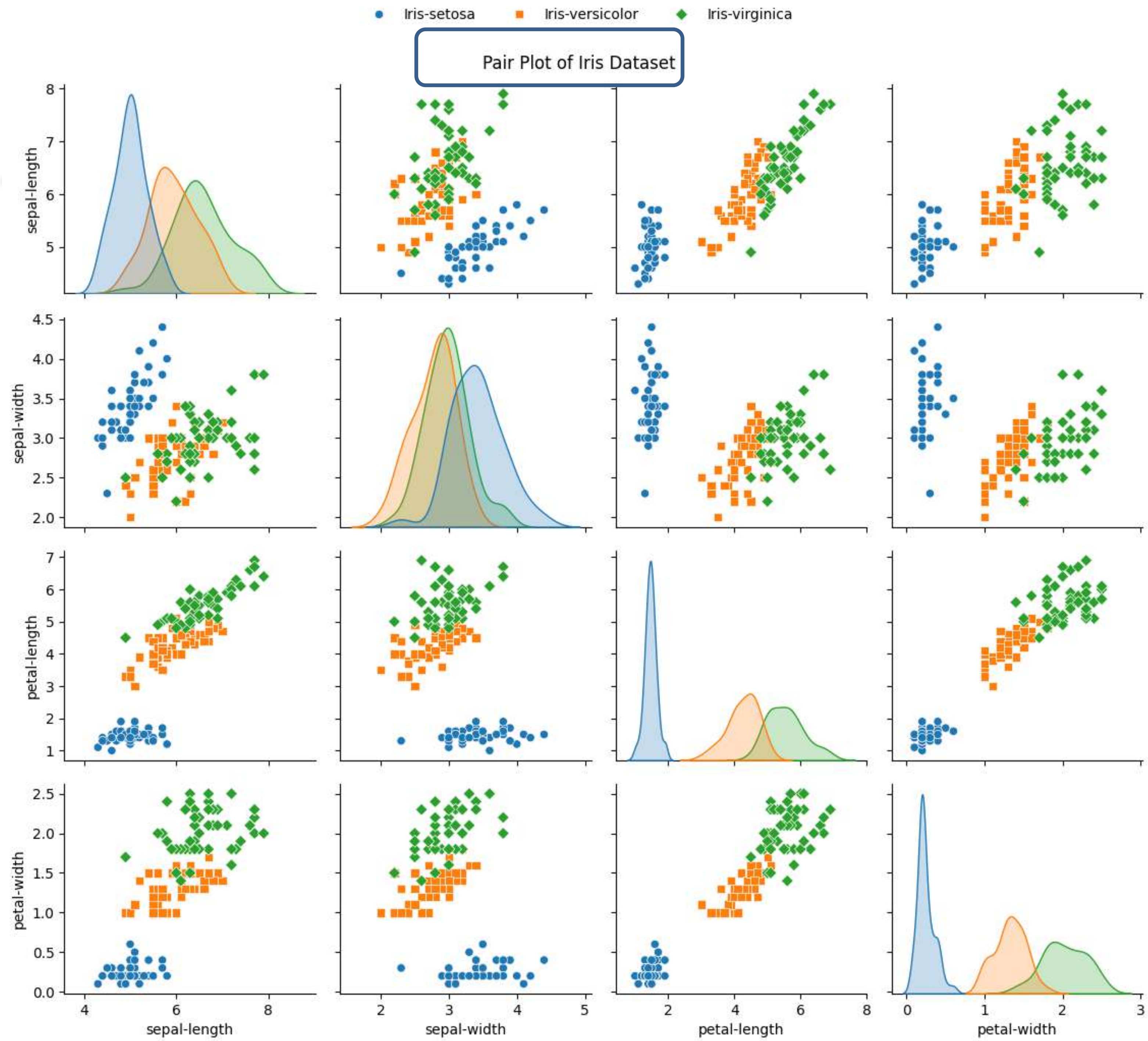


Image source: https://sthalles.github.io/



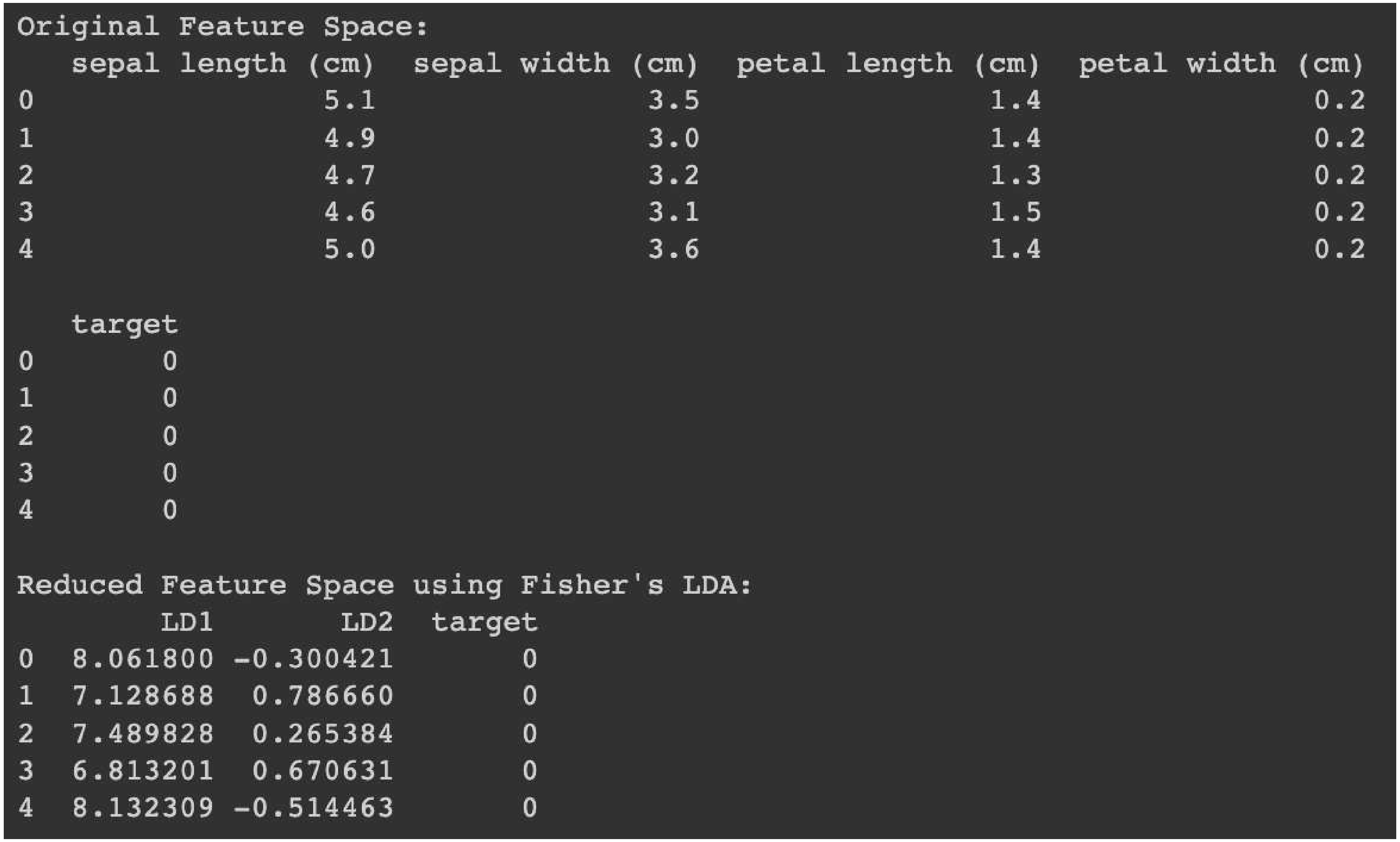
L
D
A

O
N

I
R
I
S

D
A
T
A
S
E
T

Fisher’s Linear Discriminant Functions



```
Original Feature Space:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0

Reduced Feature Space using Fisher's LDA:
        LD1       LD2  target
0  8.061800 -0.300421       0
1  7.128688  0.786660       0
2  7.489828  0.265384       0
3  6.813201  0.670631       0
4  8.132309 -0.514463       0
```

L
D
A

O
N

I
R
I
S

D
A
T
A
S
E
T

Ref: https://developer.ibm.com/tutorials/awb-implementing-linear-discriminant-analysis-python/

Thank you!