CS 474/574 Machine Learning
2. Linear Classifiers

Prof. Dr. Forrest Sheng Bao
Dept. of Computer Science
Iowa State University
Ames, IA, USA

January 31, 2023

Vectors

- Imagine a grocery store selling the following items:
  - juice, at 1 dollar per bottle
  - sugar, at 2 dollars per bag
  - tomatoes, at 3 dollars each
- Now customer A wants to buy 2 bottles of juice, 3 bags of sugar, and 5 tomatoes. The total is $1 \times 2 + 2 \times 3 + 3 \times 5 = 23$
- For any customer who wants to buy x bottles of juice, y bags of sugar, and z tomatoes, the total is $1x + 2y + 3z$.
- We see two groups of numbers here: the unit prices and the amounts of items. And there is always a one-to-one correpondence between them when computing the total price.
- For each of the groups, we could use an *ordered* list to represent the numbers.
- Hence, we introduce the concept of *vectors*. For example, the vector $\mathbf{u} = [u_0, u_1, u_2]$ for unit prices and the vector $\mathbf{v} = [v_0, v_1, v_2]$ for respective amounts. A number in a vector is called an *element*.
- Note that we use bold font for vectors. The notation $\vec{u}$ is also used.

Vectors II

- The sum of pairwise products, e.g., the total price in the grocery example, is called the *dot product* denoted as $\mathbf{u} \cdot \mathbf{v}$.
- For example, for customer A, the total is $[1, 2, 3] \cdot [2, 3, 5] = 23$ where $\mathbf{u} = [1, 2, 3]$ (dollars) and $\mathbf{v} = [2, 3, 5]$ (amounts).
- Generalize: any expression of the form

$$\sum_i x_i y_i = x_1 y_1 + x_2 y_2 + \dots$$

  is the dot product $\mathbf{x} \cdot \mathbf{y}$ between two vectors $\mathbf{x} = [x_1, x_2, \dots]$ and $\mathbf{y} = [y_1, y_2, \dots]$.
- It is also called. . . .*the weighted sum*.
- More examples of dot product: taxi (start price, mileage, time, tips), cloud service (storage, instance, badnwidth)
- In contrast to a vector, a *scalar* has only one number.
- A vector resembles a 1-D array in computer programming. Demo.

Dot product

In numpy:

```
In [5]: numpy.array((1,2,3))@numpy.array((4,5,6))
Out[5]: 32

In [6]: numpy.array((1,2,3)).dot(numpy.array((4,5,6)))
Out[6]: 32

In [7]: numpy.matmul(numpy.array((1,2,3)), \
                     numpy.array((4,5,6)))
Out[7]: 32
```

Dot and matmul differ.

In TF: matmul

Why vectors matter in machine learning?

- Earlier we mentioned that each sample is often characterized by a set of factors known as *feature values*, e.g., factors related to house price, sizes of parts for flowers, or just a sequence of raw information unit, e.g., pixels of hardwritten digits
- For an ML model, the input is a vector – order of elements matters.
- The simplest model is a weighted sum of such vector. Hence we need dot products.
  - For example, predicting the fuel efficiency of a car from the number of seats and the price.
- The batched multiplicaton and summation operations can be very predicable and efficient if parallelized or vectorized. Hence, GPU and SIMD are used widely in ML. (See FMA)
- "Computer science is no more about computers than astronomy is about telescopes.' ' – Edsger Dijkastra

Matrixes (matrices)

- In the grocery store example, what if we want to compute the total prices for two customers at once?
- We introduce *matrixes* which can be considered as the stacked vectors.
- For example, Customer A's amount vector is $\mathbf{v_A} = [2, 3, 5]$, and Customer B's amount vector is $\mathbf{v_B} = [4, 2, 1]$. Their totals are $\mathbf{u} \cdot \mathbf{v_A}$ and $\mathbf{u} \cdot \mathbf{v_B}$, respecitvely.
- We could stack $\mathbf{v_A}$ and $\mathbf{v_B}$ into a matrix of two *rows* and three *columns*

$$\mathbf{V} = \begin{pmatrix} 2 & 3 & 5 \\ 4 & 2 & 1 \end{pmatrix}$$

- And then (tentatively!!!)

$$\mathbf{u} \cdot \mathbf{V} = \mathbf{u} \cdot \begin{pmatrix} \mathbf{v_A} \\ \mathbf{v_B} \end{pmatrix} = \begin{pmatrix} [1, 2, 3] \cdot [2, 3, 5] \\ [1, 2, 3] \cdot [4, 2, 1] \end{pmatrix} = \begin{pmatrix} 23 \\ 11 \end{pmatrix}$$

- In principle, yes. In notation, no.

Matrixes II

- Matrixes are derived from systems of linear equations.
- For example, the system of linear equations (x and y are unknowns) $\begin{cases} a_1x + b_1y = & c_1 \\ a_2x + b_2y = & c_2 \end{cases}$ can be written into matrix representation as $\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$
- Caught your eyes? x and y are written vertically.
- In matrix multiplication, the second matrix is sliced vertically, and a vertical slice is dot-produced with rows of the first matrix to populate one column in the resulting matrix.
- The proper way to write the grocery totals:

$$\mathbf{u} \cdot \mathbf{V}^T = \mathbf{u} \cdot \begin{pmatrix} | & | \\ \mathbf{v_A}^T & \mathbf{v_B}^T \\ | & | \end{pmatrix} = \begin{pmatrix} [1,2,3] \cdot [2,3,5]^T \\ [1,2,3] \cdot [4,2,1]^T \end{pmatrix} = \begin{pmatrix} 23 & 11 \end{pmatrix}$$

- What is the superscript T?

Matrixes III

- The superscript T means *transpose*, basically swapping the row and the column.
- Allow us to extend the definition of a vector: a vector is a matrix of only one column (a *column vector*) or one row (*row vector*).
- The vertical bars in previous slide do not mean anything numerical. They simply indicate that v_A or v_B is a column vector, and V is the result of horizontally stacking them (Demo: `hstack` and `vstack`).
- Due to ML convention, any vector is a column vector in this class. And the dot product between any two (column) vectors $\mathbf{u}$ and $\mathbf{v}$ will be written as $\mathbf{u}^T\mathbf{v}$ or $\mathbf{v}^T\mathbf{u}$
- Given two matrixes $\mathbf{A}$ and $\mathbf{B}$, AB is not always the same as BA.

Tensors

- In computers, matrixes and vectors are special cases of tensors.
- row-major vs column-major
- axses
- Hadamard product
- Outer/tensor product
- Broadcast, tensor product
- Squeeze

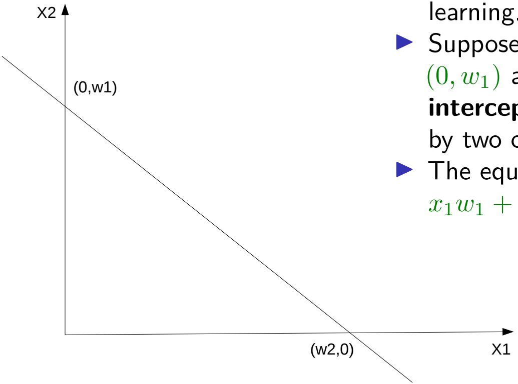
Matrix calculus

Matrix calculus on Wikipedia

Demo: Think “matricsilly’ ’

- scalar multiplication
- no/avoiding for-loops. use matrixes for batch operations.

The hyperplane



- Now, let's begin our journey on supervised learning.
- Suppose we have a line going thru points $(0, w_1)$ and $(w_2, 0)$ (which are the **intercepts**) in a 2-D vector space spanned by two orthogonal bases x_1 and x_2.
- The equation of this line is $x_1 w_1 + x_2 w_2 - w_1 w_2 = 0$.

- In matrix form (By default, all vectors are column vectors):

$$(x_1, x_2, 1) \begin{pmatrix} w_1 \\ w_2 \\ -w_1 w_2 \end{pmatrix} = \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}}_{\mathbf{x}^T} \underbrace{\begin{pmatrix} w_1 \\ w_2 \\ -w_1 w_2 \end{pmatrix}}_{\mathbf{w}} = 0$$

The hyperplane (cond.)

- Let

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

and

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ -w_1 w_2 \end{pmatrix}$$

- x_1 and x_2 are two **feature values** comprising the feature vector. 1 is **augmented** for the bias $-w_1 w_2$.
- Then the equation is rewritten into matrix form: $\mathbf{x}^T \cdot \mathbf{w} = 0$. For space sake, $\mathbf{x}^T \mathbf{w} = \mathbf{x}^T \cdot \mathbf{w}$.
- Further, since both $\mathbf{x}$ and $\mathbf{w}$ are vectors, $\mathbf{x}^T \mathbf{w} = \mathbf{w}^T \mathbf{x}$.

The hyperplane (cond.)

- Expand to D-dimension.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \\ 1 \end{pmatrix}$$

and

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \\ -w_1 w_2 \cdots w_D \end{pmatrix}.$$

Then $\mathbf{x}^T \cdot \mathbf{w} = 0$, denoted as the *hyperplane* in $\mathbb{R}^D$.

Binary Linear Classifier

- A binary linear classier is a function $\hat{f}(\mathbf{x}) = \mathsf{sgn}(\mathbf{w}^T\mathbf{x}) \in \{-1, 0, 1\}$ where sgn is the sign function. Note that the $\mathbf{x}$ has been augmented as mentioned before.
- A properly trained binary linear classifier should hold that

$$\begin{cases} \mathbf{w}^T\mathbf{x} > 0 & \forall \mathbf{x} \in C_1 \\ \mathbf{w}^T\mathbf{x} < 0 & \forall \mathbf{x} \in C_2 \end{cases} \tag{1}$$

  where C_1 and C_2 are the two classes.
- When preparing the training data, we define the label $y = +1$ for every sample $\mathbf{x} \in C_1$, and $y = -1$ for every sample $\mathbf{x} \in C_2$.
- Given a new sample whose augmented feature vector is $\mathbf{x}$, if $\mathbf{w}^T\mathbf{x} > 0$, it is classified to C_1, or equivalently its predicted label $\hat{y} = +1$. Otherwise, class C_2, or $\hat{y} = -1$.
- Example: Let $\mathbf{w} = (2, 4, -8)^T$, what's the class for new sample $\mathbf{x} = (1, 1, 1)^T$ (already augmented)?
- Solution: $\mathbf{w}^T\mathbf{x} = -2 < 0$. Hence the sample of feature value $(1, 1)$ belongs to class C_1.

Normalized feature vector

- Eq. 1 has two directions. Let's unify them into one.
- A correctly classified sample $\mathbf{x_i}$ of label $y_i \in \{+1, -1\}$ shall satisfy the inequality $\mathbf{w}_i^T \mathbf{x} y_i > 0$. (When y_i is negative, it flips the direction of the inequality.)
- $\mathbf{x}_i y_i$ is called the **normalized feature vector** for sample $\mathbf{x_i}$.
- Please note that the term "normalized" could have different meanings in different context of ML.

Solving inequalities: the simplest way to find the $\mathbf{W}$

- Let's look at a case where the feature vector is 1-D.
- Let the training set be $\{(4,+1),(5,+1),(1,-1),(2,-1)\}$. Their augmented feature vectors are: $x_1=(4,1)^T$, $x_2=(5,1)^T$, $x_3=(1,1)^T$, $x_4=(2,1)^T$.
- Let $\mathbf{w}^T=(w_1,w_2)$. In the training process, we can establish 4 inequalities:

$$\begin{cases} 4w_1+w_2 & >0 \\ 5w_1+w_2 & >0 \\ w_1+w_2 & <0 \\ 2w_1+w_2 & <0 \end{cases}$$

- We can find many w_1 and w_2 to satisfy the inequalities. But, how to pick the best?

Math recap: Gradient

- The partial derivative of a multivariate function is a vector called the gradient, representing the derivatives of a function on different directions.
- For example, let $f(\mathbf{x}) = x_1^2 + 4x_1 + 2x_1x_2 + 2x_2^2 + 2x_2 + 14$. f maps a vector $\mathbf{x} = (x_1, x_2)^T$ to a scalar.
- Then we have

$$\nabla f = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1 + 2x_2 + 4 \\ 4x_2 + 2x_1 + 2 \end{pmatrix}$$

- The gradient is a special case of *Jacobian matrix*. (see also: *Hessian matrix* for second-order partial derivatives.)
- A *critical point* or a *stationary point* is reached where the derivative is zero on any direction.
  - a local extremum (a maximum or a minimum)
  - saddle point
- if a function is convex, a local minimum/maxinum is the *global minimum/maximum*.

Finding the linear classifier via zero-gradient

- Two steps here:
  - Define a cost function to be minimized (The learning is the about minimizing the cost function)
  - Choose an algorithm to minimize (e.g., gradient, least squared error etc.)
- One intuitive criterion can be the sum of error square:

$$J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 = \sum_{i=1}^{N} (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

  where $\mathbf{x}_i$ is the i-th sample (we have N samples here), y_i the corresponding label, $\mathbf{w}^T\mathbf{X}$ is the prediction.
- For each sample $\mathbf{x}_i$, the error of the classifier is $\mathbf{w}^T\mathbf{x} - y_i$. The square is to avoid that errors on difference samples cancel out, e.g., $[+1 - (-1)] - [-1 - (+1)] = 0$.

Finding the linear classifier via zero-gradient (cond.)

- Minimizing $J(\mathbf{w})$ means: $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 2\sum_{i=1}^{N} \mathbf{x}_i(\mathbf{x}_i^T \mathbf{w} - y_i) = (0, \ldots, 0)^T$
- Hence, $\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T \mathbf{w} = \sum_{i=1}^{N} \mathbf{x}_i y_i$
- The sum of a column vector multiplied with a row vector produces a matrix.

$$\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \\ | & | & & | \end{pmatrix} \begin{pmatrix} — & \mathbf{x}_1^T & — \\ — & \mathbf{x}_2^T & — \\ & \vdots & \\ — & \mathbf{x}_N^T & — \end{pmatrix} = \mathbb{X}^T \mathbb{X}$$

- Why? Continue on next page.

Finding the linear classifier via zero-gradient (cond.)

$$\mathbf{x}_i \mathbf{x}_i^T = \begin{pmatrix} | \\ \mathbf{x}_i \\ | \end{pmatrix} \begin{pmatrix} — & \mathbf{x}_i & — \end{pmatrix} = \begin{pmatrix} \mathbf{x}_i[1] \cdot \mathbf{x}_i[1] & \mathbf{x}_i[1] \cdot \mathbf{x}_i[2] & \mathbf{x}_i[1] \cdot \mathbf{x}_i[3] & \cdots \\ \mathbf{x}_i[2] \cdot \mathbf{x}_i[1] & \mathbf{x}_i[2] \cdot \mathbf{x}_i[2] & \mathbf{x}_i[2] \cdot \mathbf{x}_i[3] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\begin{aligned} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T &= \begin{pmatrix} \sum_i \mathbf{x}_i[1] \cdot \mathbf{x}_i[1] & \sum_i \mathbf{x}_i[1] \cdot \mathbf{x}_i[2] & \sum_i \mathbf{x}_i[1] \cdot \mathbf{x}_i[3] & \cdots \\ \sum_i \mathbf{x}_i[2] \cdot \mathbf{x}_i[1] & \sum_i \mathbf{x}_i[2] \cdot \mathbf{x}_i[2] & \sum_i \mathbf{x}_i[2] \cdot \mathbf{x}_i[3] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{x}_1[1] & \mathbf{x}_2[1] & \mathbf{x}_3[1] & \cdots \\ \mathbf{x}_1[2] & \mathbf{x}_2[2] & \mathbf{x}_3[2] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} \mathbf{x}_1[1] & \mathbf{x}_2[1] & \mathbf{x}_3[1] & \cdots \\ \mathbf{x}_2[1] & \mathbf{x}_2[2] & \mathbf{x}_3[2] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\ &= \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \\ | & | & & | \end{pmatrix} \begin{pmatrix} — & \mathbf{x}_1^T & — \\ — & \mathbf{x}_2^T & — \\ & \vdots & \\ — & \mathbf{x}_N^T & — \end{pmatrix} = \mathbb{X}^T \mathbb{X} \end{aligned}$$

Finding the linear classifier via zero-gradient (cond.)

- $$\sum_{i=1}^{N} \mathbf{x}_i y_i = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \\ | & | & & | \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$
  $= \mathbb{X}^T \mathbf{y}$
- $\mathbb{X}^T \mathbb{X} \mathbf{w} = \mathbb{X}^T \mathbf{y}$
- $(\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T \mathbb{X} \mathbf{w} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T \mathbf{y}$
- $\mathbf{w} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T \mathbf{y}$

Gradient descent approach

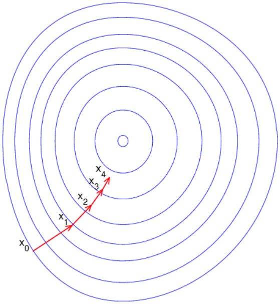
Since we define the target function as $J(\mathbf{w})$, finding $J(\mathbf{w}) = 0$ or minimizing $J(\mathbf{w})$ is intuitively the same as reducing $J(\mathbf{w})$ along the gradient. The algorithm below is a general approach to minimize any multivariate function: changing the input variable proportionally to the gradient.

Algorithm 1: pseudocode for gradient descent approach

Input: an initial $\mathbf{w}$, stop criterion θ, a learning rate function $\rho(\cdot)$, iteration step $k = 0$

```
while ∇J(w) > θ do
    w_{k+1} := w_k − ρ(k)∇J(w)
    k := k + 1
end while
```

$$\textbf{while } \nabla J(\mathbf{w}) > \theta \textbf{ do}$$
$$\quad \mathbf{w}_{k+1} := \mathbf{w}_k - \rho(k)\nabla J(\mathbf{w})$$
$$\quad k := k + 1$$
$$\textbf{end while}$$



Gradient descent approach (cond.)

In many cases, the $\rho(k)$'s amplitude (why amplitude but not the value?) decreases as k increases, e.g., $\rho(k) = \frac{1}{k}$, in order to shrink the adjustment.Also in some cases, the stop condition is $\rho(k)\nabla J(\mathbf{w}) > \theta$. The limit on k can also be included in stop condition – do not run forever. –>

Gradient descent on $J(w)$?

- Recall that $J(\mathbf{w}) = \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 = \sum_{i=1}^{N}(\mathbf{x}_i^T\mathbf{w} - y_i)^2$
- And that $\nabla J_w = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 2\sum_{i=1}^{N}\mathbf{x}_i(\mathbf{x}_i^T\mathbf{w} - y_i)$
- Do it matrixly:

$$\mathbf{A} = \mathbf{x}_i^T\mathbf{w} - y_i = \underbrace{\begin{pmatrix} — & \mathbf{x}_1^T & — \\ — & \mathbf{x}_2^T & — \\ & \vdots & \\ — & \mathbf{x}_N^T & — \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{D+1} \end{pmatrix}}_{N\times 1} - \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \mathbb{X}\mathbf{w} - \mathbf{y}$$

- $\nabla J_w = 2\sum \underbrace{\mathbf{x}_1 A_1}_{D+1\times 1} + \mathbf{x}_2 A_2 + \cdots$
-

```
N = X.shape[0]
X_augmented = numpy.hstack((X, numpy.ones((N,1)) )) # augmen
A = numpy.matmul(X_augmented, w) - y  # this results in a 1-
gradient = X_augmented * A[:, numpy.newaxis]
gradient = numpy.sum(gradient, axis=0)
```

Gradient descent

- Demo notebook
- Videe file

Fisher's linear discriminant

- What really is $\mathbf{w}^T\mathbf{x}$? The vector $\mathbf{x}$ is projected to a 1-D space (actually perpendicular to $\mathbf{w}$) in which the classification decision is done.
- This is what we prefer after the projection:
  - samples of each class distribute tightly around its center (minimized intra-class difference)
  - the distribution centers of two classes are very far from each other (maximized inter-class difference)
- Quantify this goal ($\mathbf{x}$ is **not augmented** because the bias has equal impact on both classes):

$$\max J(\mathbf{w}) = \frac{(\tilde{m}_1 - \tilde{m}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

  where $\tilde{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{w}^T\mathbf{x}$ is the post-projection center of class i and $\tilde{\mathbf{s}}_i^2 = \sum_{\mathbf{x} \in C_i} (\mathbf{w}^T\mathbf{x} - \tilde{m}_i)^2$ is the post-projection, inter-class variance for class i.
- Tails of the distributions of both classes is less likely to overlap. A new sample projected is clearly proximate to one of the two classes.

Fisher's (cond.)

- $(\tilde{m}_1 - \tilde{m}_2)^2 = (\mathbf{w}^T(\mathbf{m_1} - \mathbf{m_2}))^2 = \mathbf{w}^T \overbrace{(\mathbf{m_1} - \mathbf{m_2})(\mathbf{m_1} - \mathbf{m_2})^T}^{\mathbf{S}_B} \mathbf{w}$ where $\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ is the pre-projection center of each class.
- $\tilde{\mathbf{s}}_i^2 = \sum_{\mathbf{x} \in C_i} (\mathbf{w}^T\mathbf{x} - \tilde{m}_i)^2 = \sum_{\mathbf{x} \in C_i} (\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\mathbf{m}_i)^2 =$

  $\mathbf{w}^T \overbrace{[\sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T]}^{\mathbf{S}_i} \mathbf{w} = \mathbf{w}^T \mathbf{S}_i \mathbf{w}$
- Hence $J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T(\mathbf{S}_1+\mathbf{S}_2)\mathbf{w}} = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_w\mathbf{w}}$. This expression is known as *Rayleigh quotient*. To maximize $J(\mathbf{w})$, the $\mathbf{w}$ must satisfy $\mathbf{S}_B\mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}$.
- Finally $\mathbf{w} = \mathbf{S}_w^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$. (Derivation saved.)
- What about bias? $\mathbf{w}^T\mathbf{m} + w_b = 0$ where $\mathbf{m} = (\mathbf{m}_1 + \mathbf{m}_2)/2$ such that the decision hyperplane lies exactly in the middle between the centers of the two classes.
-