Statistical (Machine) Learning

Lecture 11 Classifier Zoo
Part 2 of 2

Pierpaolo Brutti

Many routes to one destination...

...get close to that Bayes strategy!

- Empirical Risk Minimization vs Plug-in methods vs Local Averaging/Smoothing
- Parametric vs Semi-parametric vs Non-parametric
- Linear vs Non-linear
- Discriminative vs Generative
- Unconstrained vs Constrained vs Penalized/Regularized
- etc.

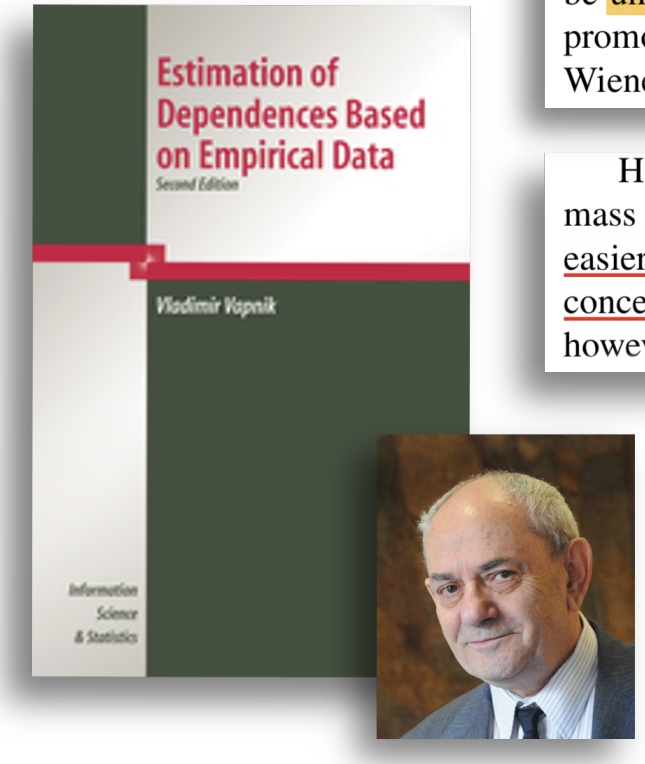
Many routes to one destination...

...get close to that Bayes strategy!

- Empirical Risk Minimization vs Plug-in methods vs Local Averaging/Smoothing
- Parametric vs Semi-parametric vs Non-parametric
- Linear vs Non-linear
- Discriminative vs Generative
- Unconstrained vs Constrained vs Penalized/Regularized
- etc.

Focus more directly on:
Empirical Risk Minimization
(...although,clearly, also plug-in methods can be thought as ERM...)

In the beginning was...Vapnik

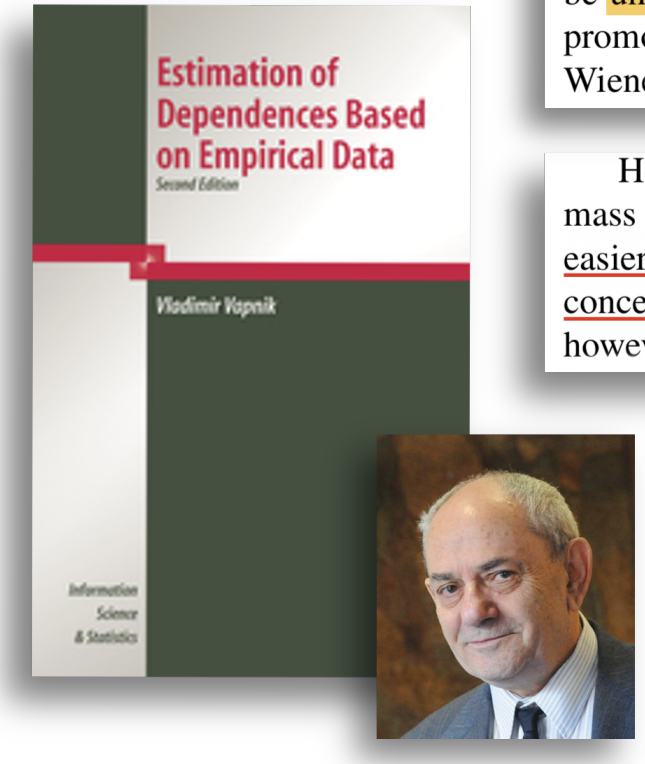


With the increasing role of science in everyday life, the general public began to discuss scientific discoveries in different areas: physical science, computer science (cybernetics), cognitive science (pattern recognition), biology (genomics), and philosophy. The discussions were held using very simplified scientific models that could be understood by the masses. Also scientists tried to appeal to the general public by promoting their philosophy using simplified models (for example, as has been done by Wiener). There is nothing wrong with this.

However, when science becomes a mass profession, the elements of the scientific mass culture in some cases start to substitute for the real scientific culture: It is much easier to learn the slogans of the scientific mass culture than it is to learn many different concepts from the original scientific sources. Science and "scientific mass culture," however, are built on very different principles.

Gloss Over the Essentials and Attract Attention to the Obvious

In the beginning was...Vapnik



With the increasing role of science in everyday life, the general public began to discuss scientific discoveries in different areas: physical science, computer science (cybernetics), cognitive science (pattern recognition), biology (genomics), and philosophy. The discussions were held using very simplified scientific models that could be understood by the masses. Also scientists tried to appeal to the general public by promoting their philosophy using simplified models (for example, as has been done by Wiener). There is nothing wrong with this.

However, when science becomes a mass profession, the elements of the scientific mass culture in some cases start to substitute for the real scientific culture: It is much easier to learn the slogans of the scientific mass culture than it is to learn many different concepts from the original scientific sources. Science and "scientific mass culture," however, are built on very different principles.

Find the Essential in the Nonobvious

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\texttt{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\texttt{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

- Gaussian Discriminant Analyses estimate the **entire** joint $p(y \mid \boldsymbol{x})$...even more extreme!

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\text{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

- Gaussian Discriminant Analyses estimate the **entire** joint $p(y \mid \boldsymbol{x})$...even more extreme!

- In line with Vladimir Vapnik's "reductionism", SVM will completely avoids the first of the two steps and **geometrically** exploits the idea behind ERM with two crucial twists: **penalization** and **surrogate loss**

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\text{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

- Gaussian Discriminant Analyses estimate the **entire** joint $p(y \mid \boldsymbol{x})$...even more extreme!

- In line with Vladimir Vapnik's "reductionism", SVM will completely avoids the first of the two steps and **geometrically** exploits the idea behind ERM with two crucial twists: **penalization** and **surrogate loss**

- In my opinion the *cool* aspect of this class of (linear) models is that all these *analytical* principles naturally flow from "simple" **geometric** arguments!

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\text{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

- Gaussian Discriminant Analyses estimate the **entire** joint $p(y \mid \boldsymbol{x})$...even more extreme!

- In line with Vladimir Vapnik's "reductionism", SVM will completely avoids the first of the two steps and **geometrically** exploits the idea behind ERM with two crucial twists: **penalization** and **surrogate loss**

- In my opinion the *cool* aspect of this class of (linear) models is that all these *analytical* principles naturally flow from "simple" **geometric** arguments!

- ...then there is also the fact that everything can be made non-linear and extremely flexible using the so-called *kernel-trick*

Fast Forward: Toward Kernel-Machines

> *When solving a problem of interest, do not solve a more general problem as an intermediate step*

- Logistic regression, being a *plug-in* method, constructs a (linear) classifier in two steps:

  1. First get an estimate the regression function $f_{\text{opt}}(\boldsymbol{x}) = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
  2. Then place an example in class 1 if its estimated probability exceeds 0.5.

- Gaussian Discriminant Analyses estimate the **entire** joint $p(y \mid \boldsymbol{x})$...even more extreme!

- In line with Vladimir Vapnik's "reductionism", SVM will completely avoids the first of the two steps and **geometrically** exploits the idea behind ERM with two crucial twists: **penalization** and **surrogate loss**

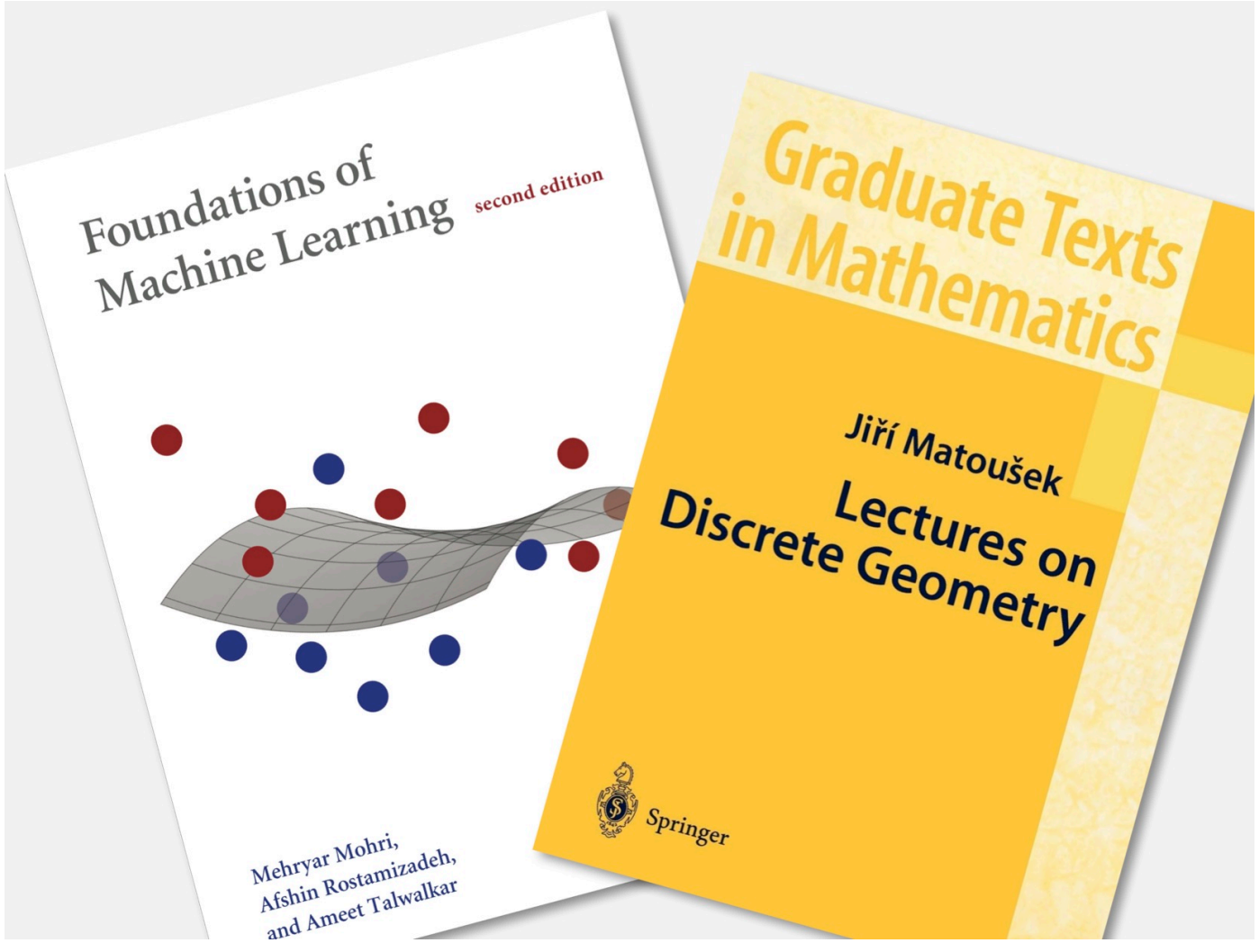
- In my opinion the *cool* aspect of this class of (linear) models is that all these *analytical* principles naturally flow from "simple" **geometric** arguments!

- ...then there is also the fact that everything can be made non-linear and extremely flexible using the so-called *kernel-trick*

- ..finally I should *also* mention that the whole construction comes with solid theoretical foundations: Vapnik–Chervonenkis theory (...zombie-Friday?!?)

...I mean...come on! Can you ask for more?

Fast Forward: Toward Kernel-Machines



Fast Forward: Toward Kernel-Machines

Going Non-Linear: Previously

- **Linear functions in some explicit transformed feature space**
  - Define **explicitly** a feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^d$, we consider $f(\boldsymbol{x}) = \boldsymbol{\beta}^\intercal \phi(\boldsymbol{x})$, with parameters $\boldsymbol{\beta} \in \mathbb{R}^d$ where d is the embedding dimension (possibly pretty large but to be controlled $\rightsquigarrow$ overfit, think *polynomials*).
  - **Pros:** simple to implement, convex optimization with gradient descent algorithms, with running time complexity in $O(nd)$, and theoretical guarantees.
  - **Cons:** only applies to linear functions on **explicit** and **fixed** feature spaces, so they *can* underfit.

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Previously

- **Linear functions in some explicit transformed feature space**
  - Define **explicitly** a feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^d$, we consider $f(\boldsymbol{x}) = \boldsymbol{\beta}^\intercal \phi(\boldsymbol{x})$, with parameters $\boldsymbol{\beta} \in \mathbb{R}^d$ where d is the embedding dimension (possibly pretty large but to be controlled $\rightsquigarrow$ overfit, think *polynomials*).
  - **Pros:** simple to implement, convex optimization with gradient descent algorithms, with running time complexity in $O(nd)$, and theoretical guarantees.
  - **Cons:** only applies to linear functions on **explicit** and **fixed** feature spaces, so they *can* underfit.
- **Neural Network**
  - Introduce non-linearity by chaining together simple layers that will implement a form of progressive non-linear data "distillation".
  - **Pros:** highly flexible and, at this point in time, surprisingly scalable given the available computing infrastructures!
  - **Cons:** this models are associated with non-convex optimization problems where gradient descent algorithms can be applied without guarantees, although several tricks have been observed to lead to better stability and performance.

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$
  - **Pros:** non-linear and flexible, easily adaptable to non-euclidean $\mathcal{X}$-space (just need a *similarity meaure*) $\rightsquigarrow$ be compared with NN-based embeddings (e.g. for text, BERT and similia, FastText, etc.)

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$
  - **Pros:** non-linear and flexible, easily adaptable to non-euclidean $\mathcal{X}$-space (just need a *similarity meaure*) $\rightsquigarrow$ be compared with NN-based embeddings (e.g. for text, BERT and similia, FastText, etc.)
  - **Pros:** simple to implement, convex optimization algorithms with strong guarantees. Easily adapt to the regularity of the target function.

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$
  - **Pros:** non-linear and flexible, easily adaptable to non-euclidean $\mathcal{X}$-space (just need a *similarity meaure*) $\rightsquigarrow$ be compared with NN-based embeddings (e.g. for text, BERT and similia, FastText, etc.)
  - **Pros:** simple to implement, convex optimization algorithms with strong guarantees. Easily adapt to the regularity of the target function.
  - **Cons:** running-time complexity up to $O(n^2)$ ($\rightsquigarrow$ *random sinks* $\rightsquigarrow$ Rahimi vs LeCun) may still suffer from the curse of dimensionality for non-smooth target functions.

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$
  - **Pros:** non-linear and flexible, easily adaptable to non-euclidean $\mathcal{X}$-space (just need a *similarity meaure*) $\rightsquigarrow$ be compared with NN-based embeddings (e.g. for text, BERT and similia, FastText, etc.)
  - **Pros:** simple to implement, convex optimization algorithms with strong guarantees. Easily adapt to the regularity of the target function.
  - **Cons:** running-time complexity up to $O(n^2)$ ($\rightsquigarrow$ *random sinks* $\rightsquigarrow$ Rahimi vs LeCun) may still suffer from the curse of dimensionality for non-smooth target functions.
- Historically speaking, as NN started to gain some traction in the 1990s, a new approach to machine learning rose to fame and quickly sent NN back to oblivion: *kernel methods* wiped out NN!
- Kernel methods are a group of techniques, the best known of which is the support vector machine (SVM), whose modern formulation was developed by *Vladimir Vapnik* and *Corinna Cortes* in the early 1990s at Bell Labs and published in 1995, although an older linear formulation was published by Vapnik and Alexey Chervonenkis as early as 1963.

Fast Forward: Toward Kernel-Machines

Going Non-Linear: Now

- **Linear functions in some implicit feature space through kernel methods**
  - Define the map ϕ **implicitly** via a *kernel* function: a **similarity** measure between $\boldsymbol{x}$'s if you wish.
  - The feature map can have arbitrarily large dimension: ϕ belongs to an Hilbert space $\mathcal{H}$ and we play with it "indirectly" through a *kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \right\rangle_{\mathcal{H}}$
  - **Pros:** non-linear and flexible, easily adaptable to non-euclidean $\mathcal{X}$-space (just need a *similarity meaure*) $\rightsquigarrow$ be compared with NN-based embeddings (e.g. for text, BERT and similia, FastText, etc.)
  - **Pros:** simple to implement, convex optimization algorithms with strong guarantees. Easily adapt to the regularity of the target function.
  - **Cons:** running-time complexity up to $O(n^2)$ ($\rightsquigarrow$ *random sinks* $\rightsquigarrow$ Rahimi vs LeCun) may still suffer from the curse of dimensionality for non-smooth target functions.
- Historically speaking, as NN started to gain some traction in the 1990s, a new approach to machine learning rose to fame and quickly sent NN back to oblivion: *kernel methods* wiped out NN!
- **Progress** generally follows a *sigmoid curve*: it starts with a period of fast progress, which gradually stabilizes as researchers hit hard limitations, and then further improvements become incremental. *Deep learning* seems to be in the first half of that sigmoid now, whereas *kernel methods* are lying flat: we'll see...but there are clear, formal connections between (*shallow*) NN and kernel machines.

Empirical Risk Minimization: Part 1 of 2

Support Vector Machines

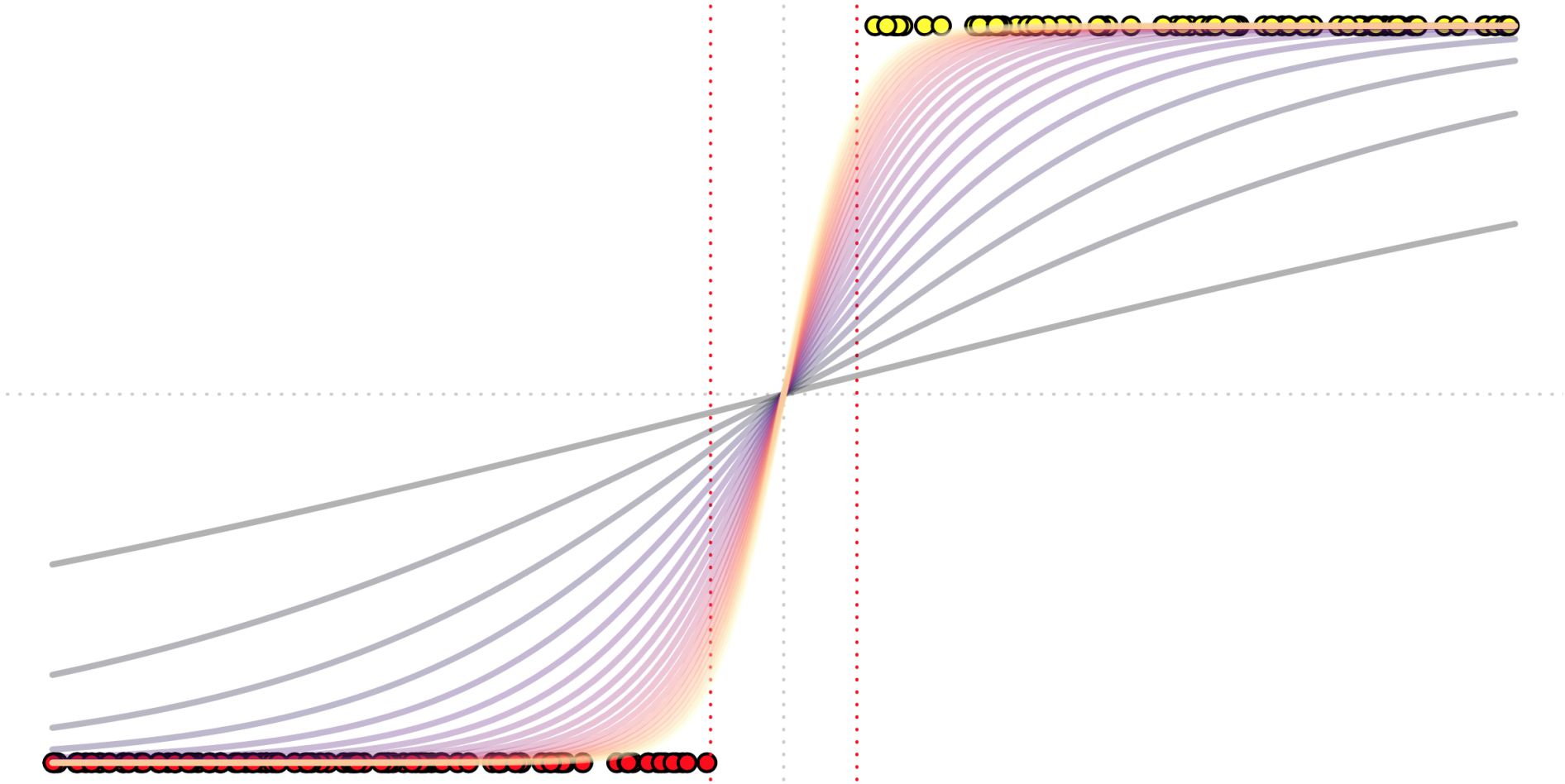
Level 1: Easy

The Linearly Separable Case

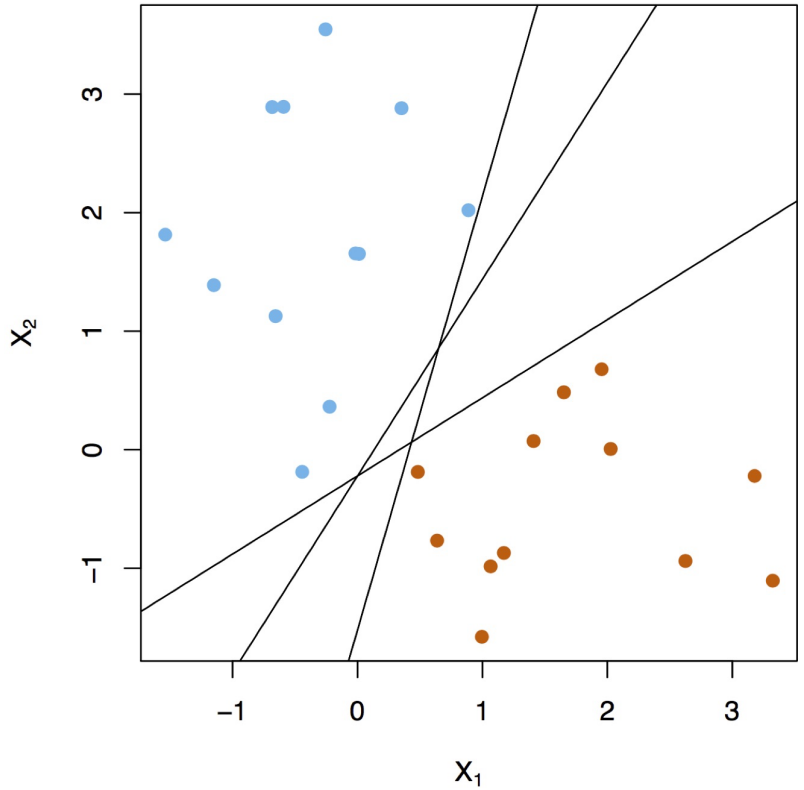
(Hard Margin)

Logistic Regression: The Dark Side

...let's start from an unexpected weakness of logistic regression...

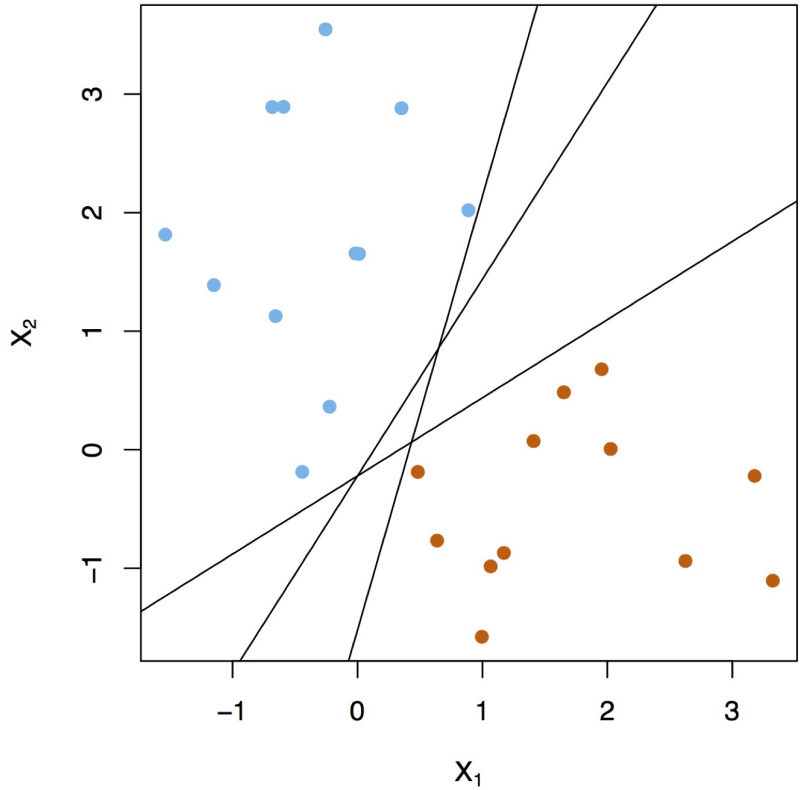


SVM: ...a question of margin!

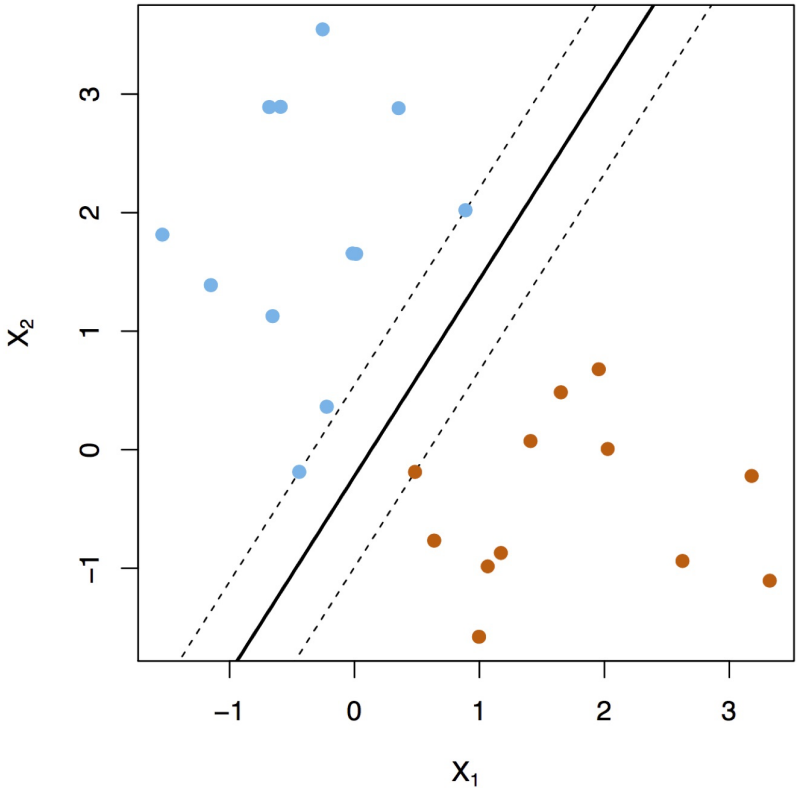


Filter out non-separating hyperplanes
(Hard-margin Constraint)

SVM: ...a question of margin!

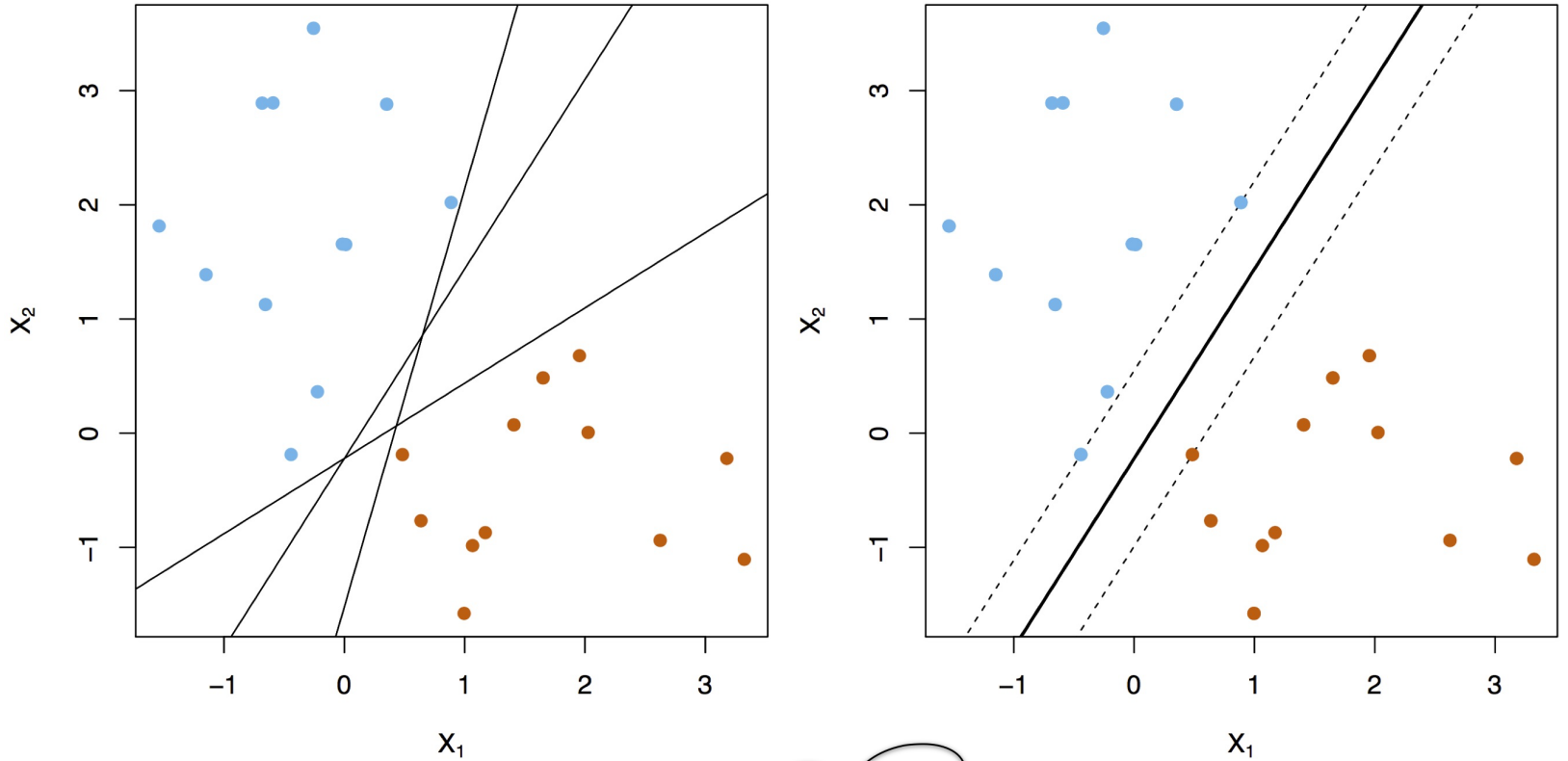


Filter out non-separating hyperplanes
(Hard-margin Constraint)



Pick the one with the largest “margin”
(Objective Function to Optimize)

SVM: ...a question of margin!

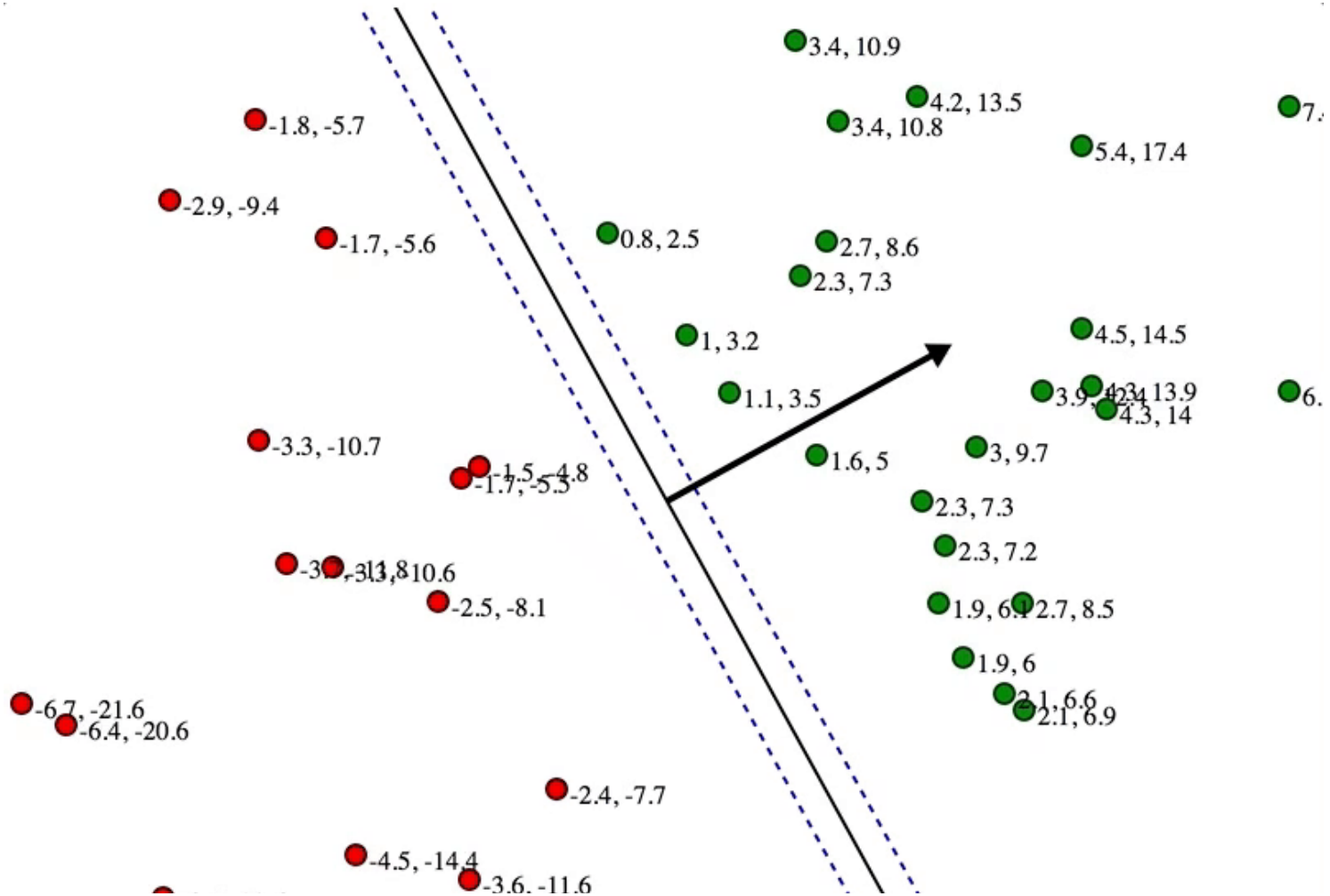


Filter out non-separating hyperplanes
(Hard-margin Constraint)

Pick the one with the largest "margin"
(Objective Function to Optimize)

Constrained (Convex) Optimization Problem

SVM: ...a question of margin!



Previously

- From now on, assume $Y \in \{-1, +1\}$, so a generic classifier can written as

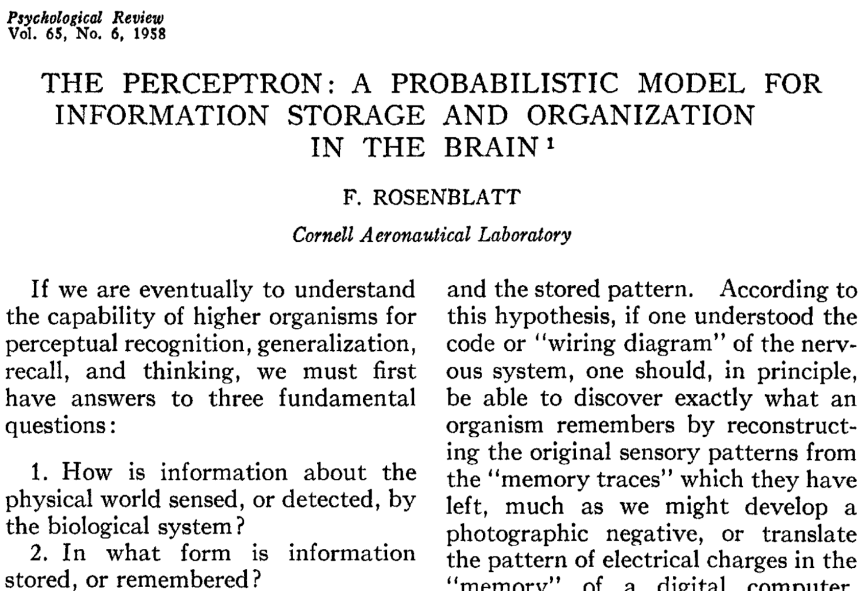
$$h(\boldsymbol{x}) = \operatorname{sign}\big(f(\boldsymbol{x})\big) \quad \text{with} \quad R(h) = \mathbb{E}\, \Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \quad \text{here} \quad f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta} = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle_{\mathbb{R}^p}$$

Previously

- From now on, assume $Y \in \{-1, +1\}$, so a generic classifier can written as

$$h(\boldsymbol{x}) = \operatorname{sign}\big(f(\boldsymbol{x})\big) \quad \text{with} \quad R(h) = \mathbb{E}\,\Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \quad \text{here} \quad f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta} = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle_{\mathbb{R}^p}$$

- Classifiers that compute a linear combination of the input features and return the sign, were called **perceptrons** in the engineering literature in the late 1950s (Rosenblatt, 1958 on *Psychological Review*!).
- Perceptrons + their *online* training are another foundational brick in our modern learning construction



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?

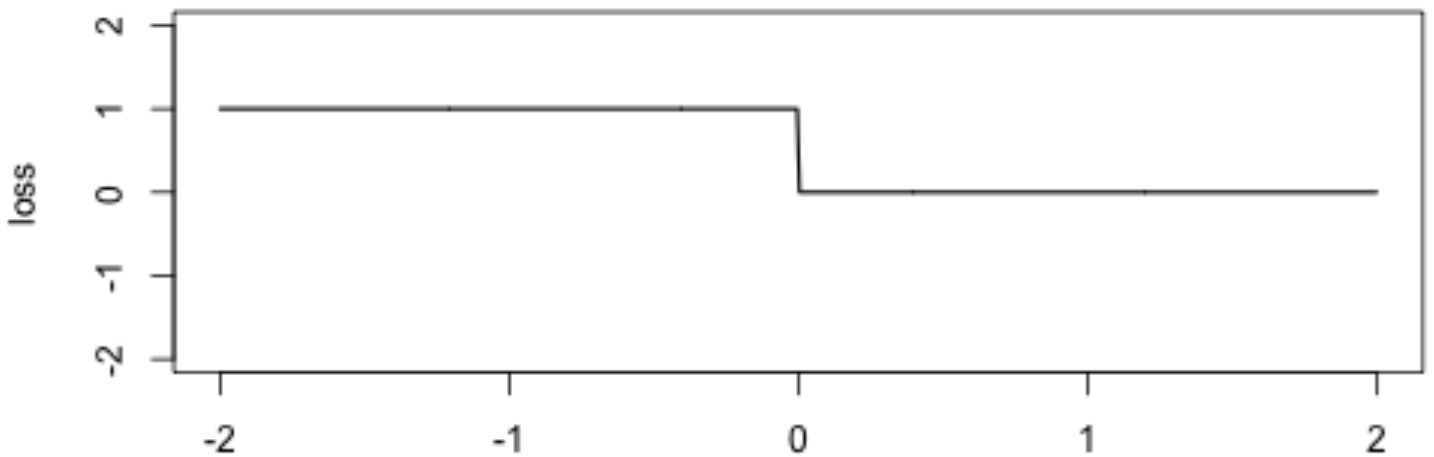
and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer

Previously

- From now on, assume $Y \in \{-1, +1\}$, so a generic classifier can written as

$$h(\boldsymbol{x}) = \operatorname{sign}\big(f(\boldsymbol{x})\big) \quad \text{with} \quad R(h) = \mathbb{E}\,\Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \quad \text{here} \quad f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta} = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle_{\mathbb{R}^p}$$

- The *natural* loss is the 0-1 loss but this is annoying to optimize as not differentiable $\rightsquigarrow$ *surrogates*



- **Idea:** come up (geometrically!) with a smoother loss that *approximates*/*bounds* 0-1 loss $\rightsquigarrow$ the *hinge*

SVM: ...a question of margin(s)!

Confidence or Functional Margin

- $R(h) = \mathbb{E}\,\Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \rightsquigarrow$ getting the class wrong is **not** so bad if we are *near* the decision boundary
  - ...getting the class right is *better* if we are **far** from the decision boundary

SVM: ...a question of margin(s)!

Confidence or Functional Margin

- $R(h) = \mathbb{E}\, \Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \rightsquigarrow$ getting the class wrong is **not** so bad if we are *near* the decision boundary
  - ...getting the class right is *better* if we are **far** from the decision boundary
- This notion of **margin** also known as **confidence or functional margin**, is $y \cdot f(\boldsymbol{x})$
  - > 0 if the point $(y, \boldsymbol{x})$ is correctly classified
  - < 0 if the point $(y, \boldsymbol{x})$ is incorrectly classified
  - Bigger (margin) is better

SVM: ...a question of margin(s)!

Confidence or Functional Margin

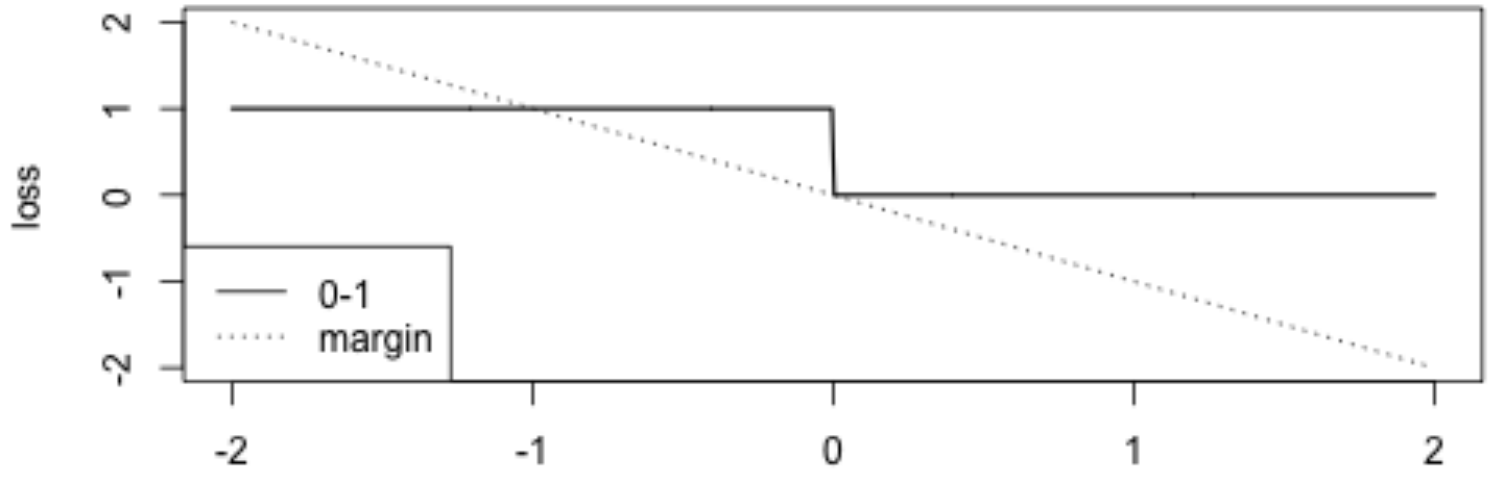
- $R(h) = \mathbb{E}\, \Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \rightsquigarrow$ getting the class wrong is **not** so bad if we are *near* the decision boundary
  - ...getting the class right is *better* if we are **far** from the decision boundary
- This notion of **margin** also known as **confidence or functional margin**, is $y \cdot f(\boldsymbol{x})$
  - > 0 if the point $(y, \boldsymbol{x})$ is correctly classified
  - < 0 if the point $(y, \boldsymbol{x})$ is incorrectly classified
  - Bigger (margin) is better
- The **margin loss** is $-y \cdot f(\boldsymbol{x})$
  - Keep with our minimize-all-the-things habits

SVM: ...a question of margin(s)!

Confidence or Functional Margin

- $R(h) = \mathbb{E}\, \Phi_{0-1}\big(Y \cdot f(\boldsymbol{X})\big) \rightsquigarrow$ getting the class wrong is **not** so bad if we are *near* the decision boundary
  - ...getting the class right is *better* if we are **far** from the decision boundary
- This notion of **margin** also known as **confidence or functional margin**, is $y \cdot f(\boldsymbol{x})$
  - > 0 if the point $(y, \boldsymbol{x})$ is correctly classified
  - < 0 if the point $(y, \boldsymbol{x})$ is incorrectly classified
  - Bigger (margin) is better
- The **margin loss** is $-y \cdot f(\boldsymbol{x})$
  - Keep with our minimize-all-the-things habits
- Margin at data point i is: $\rho_i = y_i \cdot f(\boldsymbol{x}_i)$
- Over-all margin $\rightsquigarrow$ driven by the data-point closest to the boundary: $M(f) = \min_i \rho_i$

0-1 loss vs margin loss



SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\text{sign}\big(f(\boldsymbol{x})\big) = \text{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$

SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\text{sign}\big(f(\boldsymbol{x})\big) = \text{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier

SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\text{sign}\big(f(\boldsymbol{x})\big) = \text{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier
- But the (confidence) margin, as defined, goes up if $c > 1$
  - Why not find a good plane and then "scale up" its $\boldsymbol{\beta}$ as far as the computer allows? $\rightsquigarrow$ *non-sense*

SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\text{sign}\big(f(\boldsymbol{x})\big) = \text{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier
- But the (confidence) margin, as defined, goes up if $c > 1$
  - Why not find a good plane and then "scale up" its $\boldsymbol{\beta}$ as far as the computer allows? $\rightsquigarrow$ *non-sense*
- Two responses: **geometric** margin, or **fixed parameter scaling**

SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\operatorname{sign}\big(f(\boldsymbol{x})\big) = \operatorname{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier
- But the (confidence) margin, as defined, goes up if $c > 1$
  - Why not find a good plane and then "scale up" its $\boldsymbol{\beta}$ as far as the computer allows? $\rightsquigarrow$ *non-sense*
- Two responses: **geometric** margin, or **fixed parameter scaling**
- **Geometric margin** (see page 150 of ESL)
  Like margin, but *geometrically* equivalent to the (signed) **distance** from the (linear) boundary:

$$y \cdot \left(\frac{\beta_0}{\|\boldsymbol{\beta}\|_2} + \left\langle \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|_2}, \boldsymbol{x} \right\rangle \right) = \frac{y \cdot (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle)}{\|\boldsymbol{\beta}\|_2}$$

SVM: ...a question of margin(s)!

Geometric Margin

- **Notice:** $\operatorname{sign}\big(f(\boldsymbol{x})\big) = \operatorname{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier
- But the (confidence) margin, as defined, goes up if $c > 1$
  - Why not find a good plane and then "scale up" its $\boldsymbol{\beta}$ as far as the computer allows? $\rightsquigarrow$ *non-sense*
- Two responses: **geometric** margin, or **fixed parameter scaling**
- **Geometric margin** (see page 150 of ESL)
  Like margin, but *geometrically* equivalent to the (signed) **distance** from the (linear) boundary:

$$y \cdot \left(\frac{\beta_0}{\|\boldsymbol{\beta}\|_2} + \left\langle \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|_2}, \boldsymbol{x} \right\rangle \right) = \frac{y \cdot (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle)}{\|\boldsymbol{\beta}\|_2}$$

- **Fixed parameter scaling**: insist that $\|\boldsymbol{\beta}\|_2 = 1$
- **Notice (A):** when actually $\|\boldsymbol{\beta}\|_2 = 1 \rightsquigarrow$ confidence margin $=$ geometric margin

SVM: ...a question of margin(s)!

Geometric Margin

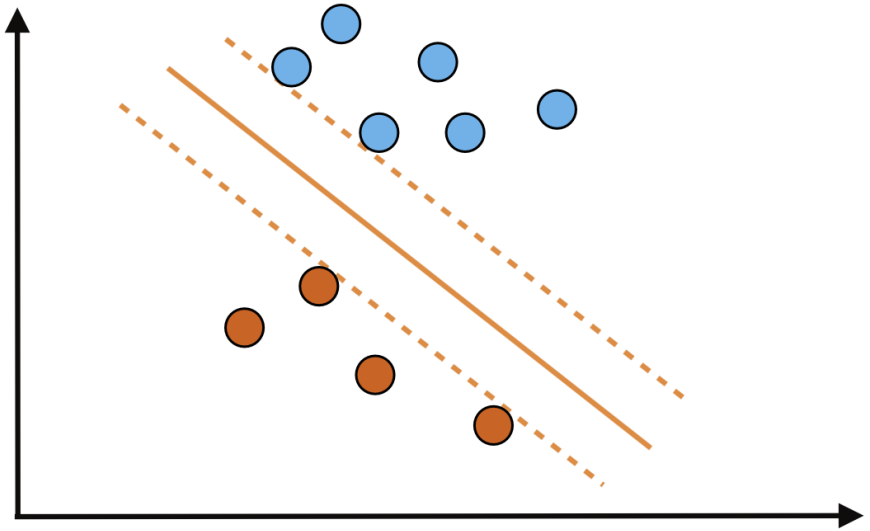
- **Notice:** $\operatorname{sign}\big(f(\boldsymbol{x})\big) = \operatorname{sign}\big(cf(\boldsymbol{x})\big)$ for *any* $c > 0$
- **Implication:** the weights $\boldsymbol{\beta}$ and $c\,\boldsymbol{\beta}$ give us the same (linear) classifier
- But the (confidence) margin, as defined, goes up if $c > 1$
  - Why not find a good plane and then "scale up" its $\boldsymbol{\beta}$ as far as the computer allows? $\rightsquigarrow$ *non-sense*
- Two responses: **geometric** margin, or **fixed parameter scaling**
- **Geometric margin** (see page 150 of ESL)
  Like margin, but *geometrically* equivalent to the (signed) **distance** from the (linear) boundary:

$$y \cdot \left(\frac{\beta_0}{\|\boldsymbol{\beta}\|_2} + \left\langle \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|_2}, \boldsymbol{x} \right\rangle \right) = \frac{y \cdot (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle)}{\|\boldsymbol{\beta}\|_2}$$

- **Fixed parameter scaling**: insist that $\|\boldsymbol{\beta}\|_2 = 1$
- **Notice (B):** different scaling/norms $\rightsquigarrow$ different margin $\rightsquigarrow$ different algorithm!

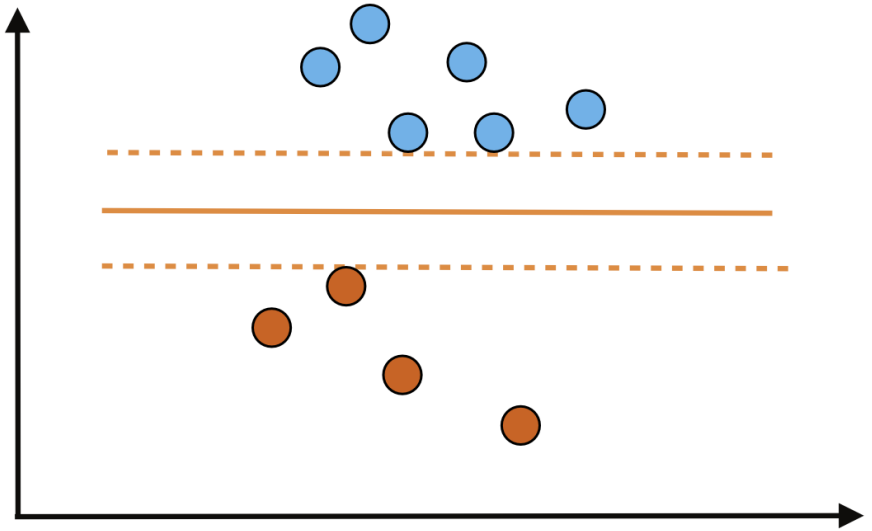
SVM: ...a question of margin(s)!

(Other) Geometric Margin(s)



Norm $\|\cdot\|_2$

(Linear SVM)



Norm $\|\cdot\|_1$

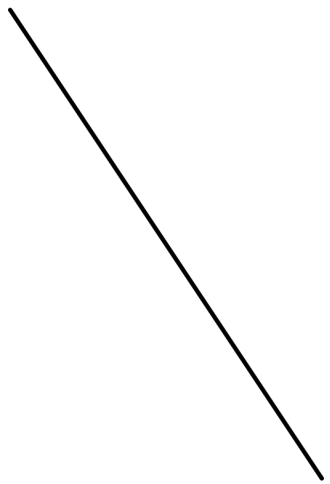
(Linear Ensembles $\rightsquigarrow$ AdaBoost)

SVM: ...a question of margin(s)!

Geometric Margin

Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes

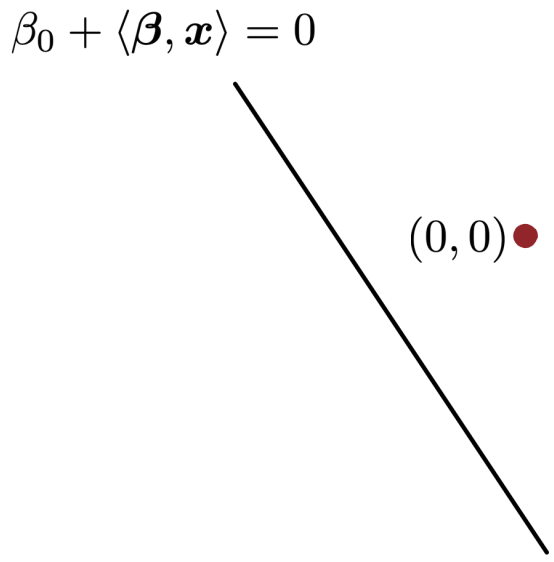
$$\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = 0$$



SVM: ...a question of margin(s)!

Geometric Margin

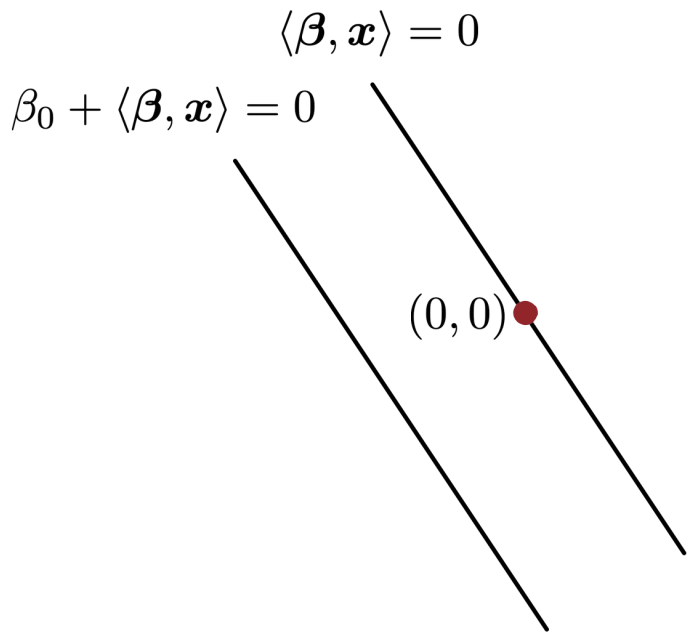
Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes



SVM: ...a question of margin(s)!

Geometric Margin

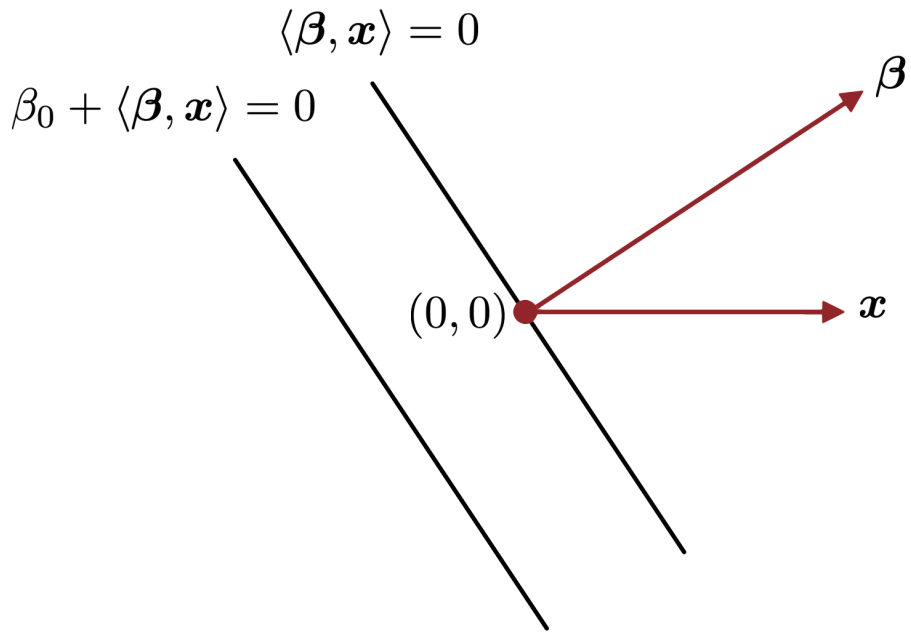
Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes



SVM: ...a question of margin(s)!

Geometric Margin

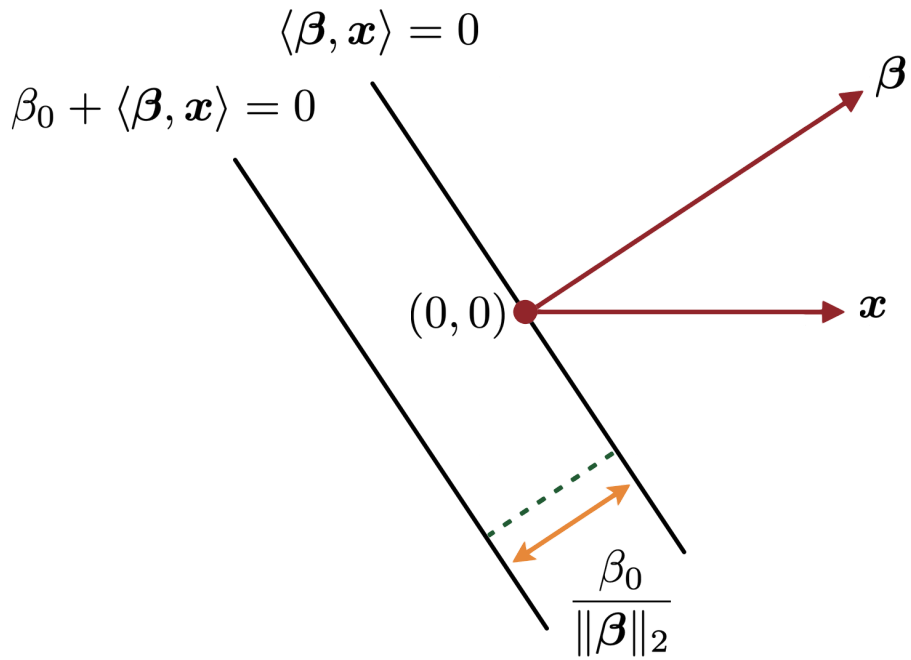
Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes



SVM: ...a question of margin(s)!

Geometric Margin

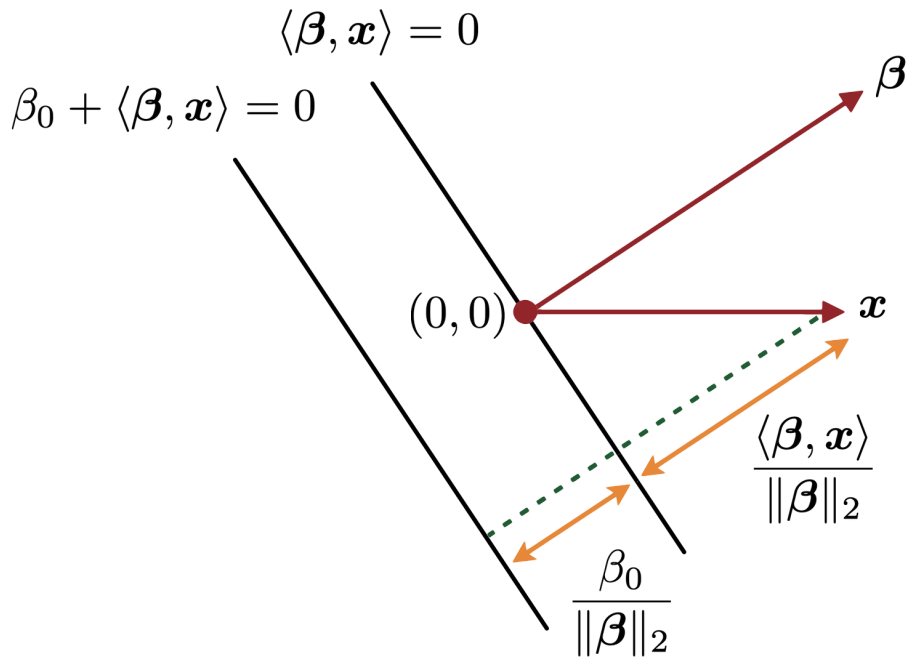
Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes



SVM: ...a question of margin(s)!

Geometric Margin

Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes



SVM: ...a question of margin(s)!

Geometric Margin

Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes

- The **geometric margin** $\rho(\boldsymbol{x})$ at a point $\boldsymbol{x}$ is its **Euclidean distance** to the hyperplane $\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = 0$

$$\rho(\boldsymbol{x}) = \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle \right|}{\|\boldsymbol{\beta}\|_2}$$

SVM: ...a question of margin(s)!

Geometric Margin

Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes

- The **geometric margin** $\rho(\boldsymbol{x})$ at a point $\boldsymbol{x}$ is its **Euclidean distance** to the hyperplane $\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = 0$

$$\rho(\boldsymbol{x}) = \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle \right|}{\|\boldsymbol{\beta}\|_2}$$

- The **geometric margin** ρ of a linear classifier for a (training) sample $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is the *minimum geometric margin* over the points in the sample

$$\rho = \min_i \rho(\boldsymbol{x}_i)$$

  that is, the distance of the hyperplane defining the classifier to the closest sample points

SVM: ...a question of margin(s)!

Geometric Margin

Step into this other type of margin geometrically:
Signed **distance** from the (linear) boundary between classes

- The **geometric margin** $\rho(\boldsymbol{x})$ at a point $\boldsymbol{x}$ is its **Euclidean distance** to the hyperplane $\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = 0$

$$\rho(\boldsymbol{x}) = \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle \right|}{\|\boldsymbol{\beta}\|_2}$$

- The **geometric margin** ρ of a linear classifier for a (training) sample $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is the *minimum geometric margin* over the points in the sample

$$\rho = \min_i \rho(\boldsymbol{x}_i)$$

  that is, the distance of the hyperplane defining the classifier to the closest sample points

- The SVM solution is the separating hyperplane with the **maximum** (geometric) margin
  - It can be viewed as the **safest** choice: a **test** point is classified correctly by such a plane even when it falls within a distance ρ of the *training samples* sharing the same label.

SVM: ...a question of margin(s)!

Geometric Margin

Now a bit of algebra and thinking...

$$\rho = \max_{\substack{\texttt{All} \\ \texttt{Separating Hyperplane}}} \min_i \rho(\boldsymbol{x}_i) = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 0}} \min_i \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right|}{\|\boldsymbol{\beta}\|_2} = \max_{(\beta_0, \boldsymbol{\beta})} \frac{\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)}{\|\boldsymbol{\beta}\|_2}$$

SVM: ...a question of margin(s)!

Geometric Margin

Now a bit of algebra and thinking...

$$\rho \quad = \max_{\substack{\texttt{All} \\ \texttt{Separating Hyperplane}}} \min_i \rho(\boldsymbol{x}_i) \quad = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 0}} \min_i \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right|}{\|\boldsymbol{\beta}\|_2} = \max_{(\beta_0, \boldsymbol{\beta})} \frac{\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)}{\|\boldsymbol{\beta}\|_2}$$

- The $3^{\texttt{rd}}$ equality follows from the fact that, since the sample is linearly separable, at the maximiming pair $(\beta_0, \boldsymbol{\beta})$, $y_i \cdot \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)$ must be non negative for each i.

SVM: ...a question of margin(s)!

Geometric Margin

Now a bit of algebra and thinking...

$$\rho \quad = \max_{\substack{\texttt{All} \\ \texttt{Separating Hyperplane}}} \min_i \rho(\boldsymbol{x}_i) \quad = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 0}} \min_i \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right|}{\|\boldsymbol{\beta}\|_2} = \max_{(\beta_0, \boldsymbol{\beta})} \frac{\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)}{\|\boldsymbol{\beta}\|_2}$$

- The $3^{\texttt{rd}}$ equality follows from the fact that, since the sample is linearly separable, at the maximiming pair $(\beta_0, \boldsymbol{\beta})$, $y_i \cdot \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)$ must be non negative for each i.
- **Observe:** the last expression is invariant to multiplication of $(\beta_0, \boldsymbol{\beta})$ by any positive scalar c
- **Consequence:** Restrict the search to $(\beta_0, \boldsymbol{\beta})$ scaled such that

$$\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right) = 1 \quad \Leftrightarrow \quad \min_i \left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right| = 1$$

SVM: ...a question of margin(s)!

Geometric Margin

Now a bit of algebra and thinking...

$$\rho \quad = \max_{\substack{\texttt{All} \\ \texttt{Separating Hyperplane}}} \min_i \rho(\boldsymbol{x}_i) \quad = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 0}} \min_i \frac{\left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right|}{\|\boldsymbol{\beta}\|_2} = \max_{(\beta_0, \boldsymbol{\beta})} \frac{\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)}{\|\boldsymbol{\beta}\|_2}$$

- The $3^{\texttt{rd}}$ equality follows from the fact that, since the sample is linearly separable, at the maximiming pair $(\beta_0, \boldsymbol{\beta})$, $y_i \cdot \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right)$ must be non negative for each i.
- **Observe:** the last expression is invariant to multiplication of $(\beta_0, \boldsymbol{\beta})$ by any positive scalar c
- **Consequence:** Restrict the search to $(\beta_0, \boldsymbol{\beta})$ scaled such that

$$\min_i y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right) = 1 \quad \Leftrightarrow \quad \min_i \left| \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \right| = 1$$

$$\implies \quad \rho \quad = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ \min_i y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) = 1}} \frac{1}{\|\boldsymbol{\beta}\|_2} \quad = \max_{\substack{(\beta_0, \boldsymbol{\beta}) \\ \forall\, i \,:\, y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1}} \frac{1}{\|\boldsymbol{\beta}\|_2} \quad = \min_{\substack{(\beta_0, \boldsymbol{\beta}) \\ \forall\, i \,:\, y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1}} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$$

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f)$
- **Subject to**: separability condition

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \ldots, n\}$

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1$ for each $i \in \{1, \dots, n\}$

Quick Convex Optimization Interlude

- Consider the constrained **convex** optimization problem

$$\min_{\theta} M(\theta) \quad \text{subject to} \quad \Omega_j(\theta) \leqslant 0 \quad j \in \{1, \dots, n\}$$

- Define the **Lagrangian**: $\mathcal{L}(\theta, \boldsymbol{\alpha}) = M(\theta) + \sum_j \alpha_j \, \Omega_j(\theta)$
- The **dual function** is define by $F(\boldsymbol{\alpha}) = \inf_\theta \mathcal{L}(\theta, \boldsymbol{\alpha})$
- A central result in **convex optimization** is that (under some conditions verified in our application), the original problem can be solved by **maximizing** the dual F *subject to*:

$$\alpha_j \geqslant 0 \quad \text{for each} \quad j \in \{1, \dots, n\}$$

Interlude: More on Lagrange Multipliers

Constrained optimization in general

- Get a bit more abstract for a moment and think about constrained optimization in general

$$\theta^\star = \min_{\theta \in \Theta} M(\theta)$$
$$\text{subject to}$$
$$\Omega(\theta) \leqslant c$$

- How might we solve this?

Constrained optimization in general

- Get a bit more abstract for a moment and think about constrained optimization in general

$$\theta^\star = \min_{\theta \in \Theta} M(\theta)$$
$$\text{subject to}$$
$$\Omega(\theta) \leqslant c$$

- How might we solve this?

1. Use the **constraint equation** $\Omega(\theta) = c$ to **eliminate a degree of freedom**
    - In other words, write one coordinate in θ as a function of the others and of c
    - Do **unconstrained** optimization over the remaining degrees of freedom

Constrained optimization in general

- Get a bit more abstract for a moment and think about constrained optimization in general

$$\theta^\star = \min_{\theta \in \Theta} M(\theta)$$
$$\text{subject to}$$
$$\Omega(\theta) \leqslant c$$

- How might we solve this?

1. Use the **constraint equation** $\Omega(\theta) = c$ to **eliminate a degree of freedom**
    - In other words, write one coordinate in θ as a function of the others and of c
    - Do **unconstrained** optimization over the remaining degrees of freedom
2. Ok, but what about the $\leqslant$ case?

Constrained optimization in general

- Get a bit more abstract for a moment and think about constrained optimization in general

$$\theta^\star = \min_{\theta \in \Theta} M(\theta)$$
$$\text{subject to}$$
$$\Omega(\theta) \leqslant c$$

- How might we solve this?

1. Use the **constraint equation** $\Omega(\theta) = c$ to **eliminate a degree of freedom**
    - In other words, write one coordinate in θ as a function of the others and of c
    - Do **unconstrained** optimization over the remaining degrees of freedom
2. Ok, but what about the $\leqslant$ case?
3. *Add* a new variable and do **unconstrained** optimization over a **larger** problem

Lagrange multipliers: Constraints $\leftrightsquigarrow$ Penalties

- If we have one *equality* constraint, say $\Omega(\theta) = c$, we'd add a **Lagrange multiplier**:

$$\min_{\theta \in \Theta, \alpha \in \mathbb{R}} M(\theta) + \alpha \left(\Omega(\theta) - c \right) = \min_{\theta \in \Theta, \alpha \in \mathbb{R}} \mathcal{L}(\theta, \alpha)$$

Lagrange multipliers: Constraints ⇜⇝ Penalties

- If we have one *equality* constraint, say $\Omega(\theta) = c$, we'd add a **Lagrange multiplier**:

$$\min_{\theta \in \Theta, \alpha \in \mathbb{R}} M(\theta) + \alpha \left(\Omega(\theta) - c \right) = \min_{\theta \in \Theta, \alpha \in \mathbb{R}} \mathcal{L}(\theta, \alpha)$$

- Do **unconstrained** optimization over *both* θ and α ⇝ derivatives

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \Omega(\theta) - c$$
$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial M}{\partial \theta} + \alpha \frac{\partial \Omega}{\partial \theta}$$

- Set derivatives to zero and solve for $\theta^\star, \alpha^\star$
- One extra unknown but also one extra equation!

Lagrange multipliers: Constraints $\leftrightsquigarrow$ Penalties

- Set to zero:

$$\Omega(\theta^\star) = c$$
$$\frac{\partial}{\partial\theta} M(\theta^\star) = -\alpha^\star \frac{\partial}{\partial\theta} \Omega(\theta^\star)$$

- First equation is the constraint again
  - So we always get: $\mathcal{L}(\theta^\star, \alpha^\star) = M(\theta^\star)$

Lagrange multipliers: Constraints $\leftrightsquigarrow$ Penalties

- Set to zero:

$$\Omega(\theta^\star) = c$$
$$\frac{\partial}{\partial\theta} M(\theta^\star) = -\alpha^\star \frac{\partial}{\partial\theta} \Omega(\theta^\star)$$

- First equation is the constraint again
  - So we always get: $\mathcal{L}(\theta^\star, \alpha^\star) = M(\theta^\star)$
- Second equation involves *both* $\theta^\star$ and $\alpha^\star$
- $\theta^\star$ will **not** be the same as the **unconstrained** optimum θ
  - ...unless the *unconstrained* optimum satisfies the constraint already, in which case $\alpha^\star = 0$
- Solving this system might be easy or hard, but the solution does give the **constrained** optimum

Lagrange multipliers are prices

- Changing the constraint level c changes $\theta^\star$ and $M(\theta)$:

$$\begin{aligned}
\frac{\partial M(\theta^\star)}{\partial c} &= \frac{\partial \mathcal{L}(\theta^\star, \alpha^\star)}{\partial c} \\
&= \frac{\partial}{\partial c}\Big(M(\theta^\star) + \alpha^\star\big(\Omega(\theta^\star) - c\big)\Big) \\
&= \left[\frac{\partial}{\partial \theta} M(\theta^\star) + \alpha^\star \frac{\partial}{\partial \theta}\Omega(\theta^\star)\right] \frac{\partial \theta^\star}{\partial c} + \big[\Omega(\theta^\star) - c\big] \frac{\partial \alpha^\star}{\partial c} - \alpha^\star \\
&= [0]\frac{\partial \theta^\star}{\partial c} + [0]\frac{\partial \alpha^\star}{\partial c} - \alpha^\star \\
&= -\alpha^\star
\end{aligned}$$

- $\alpha^\star =$ Rate at which the optimal *value* improves as the constraint is relaxed
- $\alpha^\star =$ How much would you pay for a (marginal) change in the level of the constraint, your **shadow price** for that constraint

Lagrange multipliers vs Penalties

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem
- The penalty factor α corresponds to the constraint level c
- "*A fine is a price*"

Moral
Lagrange multipliers turns
constrained optimization
into *penalized* optimization

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...but don't forget: $M(\theta)$ is typically an *empirical risk* that, we saw, can be thought as a *negative log-likelihood* for some probabilistic model!

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...but don't forget: $M(\theta)$ is typically an *empirical risk* that, we saw, can be thought as a *negative log-likelihood* for some probabilistic model!
- **Consequence:**
  - At least *formally*, $M(\theta) + \alpha\,\Omega(\theta)$ can be interpreted from a **Bayesian modeling** perspective as the numerator of a *negative log-posterior* with *negative log-prior* over θ equal to $\alpha\,\Omega(\theta)$

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \operatorname{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...but don't forget: $M(\theta)$ is typically an *empirical risk* that, we saw, can be thought as a *negative log-likelihood* for some probabilistic model!
- **Consequence:**
  - At least *formally*, $M(\theta) + \alpha\,\Omega(\theta)$ can be interpreted from a **Bayesian modeling** perspective as the numerator of a *negative log-posterior* with *negative log-prior* over θ equal to $\alpha\,\Omega(\theta)$
  - Here α is an *hyperparameter* that we may tune (e.g. Empirical Bayes) or treat hierarchically by selecting a suitable **hyperprior**.

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \text{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \text{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...but don't forget: $M(\theta)$ is typically an *empirical risk* that, we saw, can be thought as a *negative log-likelihood* for some probabilistic model!
- **Consequence:**
  - At least *formally*, $M(\theta) + \alpha\,\Omega(\theta)$ can be interpreted from a **Bayesian modeling** perspective as the numerator of a *negative log-posterior* with *negative log-prior* over θ equal to $\alpha\,\Omega(\theta)$
  - Here α is an *hyperparameter* that we may tune (e.g. Empirical Bayes) or treat hierarchically by selecting a suitable **hyperprior**.
  - This interpretation also suggests that, instead of just focusing on *optimization* to get the **MAP** (*maximum a posteriori*) for θ, we may move to **simulation** (e.g. MCMC methods) to explore the posterior surface and get a different/better value for θ (+ *uncertainty quantification* too!)

Lagrange multipliers vs Penalties vs Priors

- Once we know $\alpha^\star$, we just do the optimization with an extra term:

$$\begin{aligned}\theta^\star &= \text{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star(\Omega(\theta) - c) \\ &= \text{argmin}_{\theta\in\Theta} M(\theta) + \alpha^\star\Omega(\theta)\end{aligned}$$

- This is a **penalized** optimization problem...but don't forget: $M(\theta)$ is typically an *empirical risk* that, we saw, can be thought as a *negative log-likelihood* for some probabilistic model!
- **Consequence:**
  - At least *formally*, $M(\theta) + \alpha\,\Omega(\theta)$ can be interpreted from a **Bayesian modeling** perspective as the numerator of a *negative log-posterior* with *negative log-prior* over θ equal to $\alpha\,\Omega(\theta)$
  - Here α is an *hyperparameter* that we may tune (e.g. Empirical Bayes) or treat hierarchically by selecting a suitable **hyperprior**.
  - This interpretation also suggests that, instead of just focusing on *optimization* to get the **MAP** (*maximum a posteriori*) for θ, we may move to **simulation** (e.g. MCMC methods) to explore the posterior surface and get a different/better value for θ (+ *uncertainty quantification* too!)
  - Clearly here we are using the Bayesian prior-to-posterior machinery *pragmatically* as a algorithm-factory: the prior is a tool, not a quantification of our beliefs!

Many constraints

- For multiple equality constraints

$$\Omega_1(\theta) = c_1 \quad \Omega_2(\theta) = c_2 \quad \cdots \quad \Omega_n(\theta) = c_n$$

  we add n Lagrange multipliers:

$$\mathcal{L}(\theta, \alpha_1, \ldots, \alpha_n) = M(\theta) + \sum_{j=1}^{n} \alpha_j (\Omega_j(\theta) - c_j)$$

- Each constraint equation gets recovered when we take the derivative w.r.t. that multiplier
- Each multiplier tells us our shadow price for loosening each constraint
- Equivalently: adding many *penalty* terms

Inequality constraints

- What if the constraint is that $\Omega(\theta) \leqslant c$?
- We add on the Lagrange multiplier *anyway*, as though it were an equality

Inequality constraints

- What if the constraint is that $\Omega(\theta) \leqslant c$?
- We add on the Lagrange multiplier *anyway*, as though it were an equality
- **Case 1**: the global, **unconstrained** optimum obeys the constraint
  - The constraint does not **bind** or **bite**
  - We should get $\alpha^\star = 0$

Inequality constraints

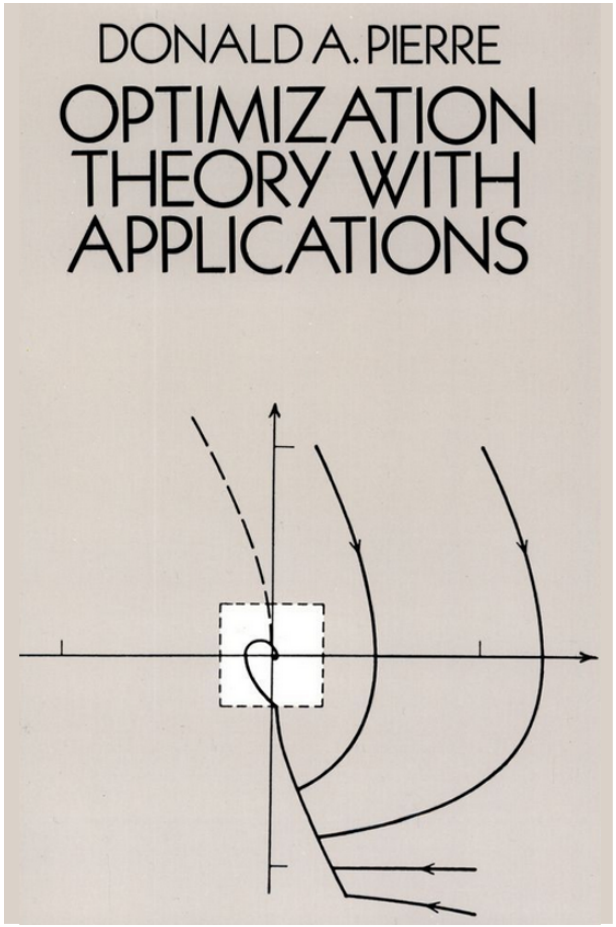
- What if the constraint is that $\Omega(\theta) \leqslant c$?
- We add on the Lagrange multiplier *anyway*, as though it were an equality
- **Case 1**: the global, **unconstrained** optimum obeys the constraint
  - The constraint does not **bind** or **bite**
  - We should get $\alpha^\star = 0$
- **Case 2**: the **unconstrained** optimum is *outside* in constrained feasible set
  - The constraint **binds**, or **is binding**, or **bites**
  - The constrained optimum $\theta^\star$ is a point where $\Omega(\theta) = c$
  - We'll get $\alpha^\star \neq 0$
  - Ignoring some subtleties, this is **Karush-Kuhn-Tucker theorem**

Inequality constraints

- What if the constraint is that $\Omega(\theta) \leqslant c$?
- We add on the Lagrange multiplier *anyway*, as though it were an equality
- **Case 1**: the global, **unconstrained** optimum obeys the constraint
  - The constraint does not **bind** or **bite**
  - We should get $\alpha^\star = 0$
- **Case 2**: the **unconstrained** optimum is *outside* in constrained feasible set
  - The constraint **binds**, or **is binding**, or **bites**
  - The constrained optimum $\theta^\star$ is a point where $\Omega(\theta) = c$
  - We'll get $\alpha^\star \neq 0$
  - Ignoring some subtleties, this is **Karush-Kuhn-Tucker theorem**

Inequality constraints: Remark on KKT conditions

...they were **not** always *this* big...



1st edition 2nd edition

This Dover edition, first published in 1986, is an unabridged and corrected republication of the work first published by John Wiley & Sons, Inc., New York, 1969. The author has written a new Preface for this edition.

Manufactured in the United States of America
Dover Publications, Inc., 31 East 2nd Street, Mineola, N.Y. 11501

Library of Congress Cataloging-in-Publication Data

Pierre, Donald A.
Optimization theory with applications.

Reprint. Originally published: New York : Wiley, c1969. With new pref.
Includes bibliographies and index.
1. Mathematical optimization. 2. Programming (Mathematics)
I. Title.
QA402.5.P5 1986 519 86-6278
ISBN 0-486-65205-X

PREFACE TO THE DOVER EDITION

The advances in computing capabilities during the past two decades have been impressive. Many types of optimization algorithms are routinely available to the user of a computer center, and small software packages for personal computers are widespread. The understanding of the theory behind these computer routines is important for the user who wishes to gain the most from them or who wishes to embed them as part of a larger system. The strength of this book lies in the breadth of optimization concepts that are presented.

In looking over the contents of the book, my main regret is that I had not included enough on the first- and second-order Kuhn-Tucker conditions which have come to play a major role in many nonlinear programming algorithms. To compensate for this, the major features of the Kuhn-Tucker conditions are presented near the end of this preface; the presentation is

Summing up on constraints and Lagrange multipliers

- Equality constraints act like penalties
  - Penalty factor $\alpha \leftrightsquigarrow$ Lagrange multiplier enforcing the constraint

Summing up on constraints and Lagrange multipliers

- Equality constraints act like penalties
  - Penalty factor α $\leftrightsquigarrow$ Lagrange multiplier enforcing the constraint
- Loosening the constraints $\leftrightsquigarrow$ weakening the penalties

Summing up on constraints and Lagrange multipliers

- Equality constraints act like penalties
  - Penalty factor α $\leftrightsquigarrow$ Lagrange multiplier enforcing the constraint
- Loosening the constraints $\leftrightsquigarrow$ weakening the penalties
- Inequality constraints, like $\Omega(\theta) \leqslant c$, get treated the same way
  - Some multipliers/penalty factors might be 0 if those constraints do **not** bite

Summing up on constraints and Lagrange multipliers

- Equality constraints act like penalties
  - Penalty factor $\alpha \leftrightsquigarrow$ Lagrange multiplier enforcing the constraint
- Loosening the constraints $\leftrightsquigarrow$ weakening the penalties
- Inequality constraints, like $\Omega(\theta) \leqslant c$, get treated the same way
  - Some multipliers/penalty factors might be 0 if those constraints do **not** bite
- Penalties and constraints are different ways of looking at the same thing

Mathematical programming

- Optimization under constraints is called **mathematical programming**
  - The name goes back to the 1930s and is older than "computer programming"
- **Linear programming**: optimize a linear objective function under linear constraints
  - Basically invented by Kantorovich for economic planning in the USSR in the 1930s, re-invented in the West for logistics and decision support in WWII (*Operations Research*), then adapted to corporate decision making, financial portfolio allocation, etc.
- **Convex programming**: optimize a convex function under convex constraints
  - Meaning: take any two points in the feasible set; every point in between them is also in the feasible set
  - Includes linear programming as a special case
- There are efficient (polynomial-time) algorithms for convex programming problems
  - Finding an ϵ-approximate optimum over p variables with k constraints takes time $O((k+p)^{3/2}p^2 \log 1/\epsilon)$

Back to biz!

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \dots, n\}$

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \ldots, n\}$

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

- Get the *KKT conditions* by setting the gradient of $\mathcal{L}$ w.r.t. the *primal* variables $(\beta_0, \boldsymbol{\beta})$ to zero and then writing the *complementary conditions*

$$\nabla_{\boldsymbol{\beta}} \mathcal{L} = \boldsymbol{\beta} - \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i = 0 \qquad \rightsquigarrow \qquad \boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i$$

$$\nabla_{\beta_0} \mathcal{L} = - \sum_i \alpha_i \, y_i = 0 \qquad \rightsquigarrow \qquad \sum_i \alpha_i \, y_i = 0$$

Learning classifiers by maximizing the margin

- Only consider "linear machines" $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ with suitably scaled coefficients such that:
- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: separability condition $\quad \rightsquigarrow \quad y_i (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \ldots, n\}$

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

- Get the *KKT conditions* by setting the gradient of $\mathcal{L}$ w.r.t. the *primal* variables $(\beta_0, \boldsymbol{\beta})$ to zero and then writing the *complementary conditions*

$$\nabla_{\boldsymbol{\beta}} \mathcal{L} = \boldsymbol{\beta} - \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i = 0 \qquad \rightsquigarrow \qquad \boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i$$

$$\nabla_{\beta_0} \mathcal{L} = - \sum_i \alpha_i \, y_i = 0 \qquad \rightsquigarrow \qquad \sum_i \alpha_i \, y_i = 0$$

- If we insert $\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i$ into $\mathcal{L}$, and use the fact that $\sum_i \alpha_i \, y_i = 0$, we get the **dual**...

Learning classifiers by maximizing the margin

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \tfrac{1}{2}\|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

KKT conditions

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

Dual

$$F(\boldsymbol{\alpha}) = \underbrace{\tfrac{1}{2}\Big\| \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \Big\|_2^2 - \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}_{-\frac{1}{2}\sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle} - \underbrace{\sum_i \alpha_i \, y_i \, \beta_0}_{=0} + \sum_i \alpha_i = \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

Learning classifiers by maximizing the margin

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \tfrac{1}{2} \|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

KKT conditions

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

Dual

$$F(\boldsymbol{\alpha}) = \underbrace{\tfrac{1}{2} \Big\| \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \Big\|_2^2 - \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}_{-\frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle} - \underbrace{\sum_i \alpha_i \, y_i \, \beta_0}_{=0} + \sum_i \alpha_i = \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

$$\implies \qquad \max_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) \quad \text{subject to} \quad \alpha_i \geqslant 0 \quad \forall \; i \in \{1, \ldots, n\} \quad \text{and} \quad \sum_i \alpha_i \, y_i = 0$$

Learning classifiers by maximizing the margin

Lagrangian

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \tfrac{1}{2} \|\boldsymbol{\beta}\|_2^2 - \sum_i \alpha_i \Big\{ y_i (\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) - 1 \Big\}$$

KKT conditions

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

Dual

$$\max_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) \quad \text{subject to} \quad \alpha_i \geqslant 0 \quad \forall \; i \in \{1, \ldots, n\} \quad \text{and} \quad \sum_i \alpha_i \, y_i = 0$$

- This is a **quadratic program** $\rightsquigarrow$ can be solved quickly!
- We do **not** need the $\boldsymbol{x}_i$'s: $F(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$
  We **only** need to store/know the **inner product** (a.k.a. *similarity measure*!) between pairs of the *original* feature vectors $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = k(\boldsymbol{x}_i, \boldsymbol{x}_j) \rightsquigarrow$ *kernelization*!

Consequences of margin maximization: algorithmic

- The new objective function, $F(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ is a convex (quadratic) function of the αs
- All the constraints on α are convex (linear)
- Convex objective function + convex constraints $\Rightarrow$ unique global minimum with no local minima
- Convex objective + convex constraints $=$ **convex programming** $\rightsquigarrow$ good algorithms which get within ϵ of the optimum in $O(n^{3.5} \log 1/\epsilon)$ time
- Quadratic objective + linear constraints $=$ **quadratic programming**, which can be solved even more efficiently (though not dramatically so)

Moral
Maximizing the margin is algorithmically tractable

Consequences of margin maximization: "support vectors"

- One of our constraints is that for each i

$$y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right) \geqslant 1$$

- This is an inequality constraint
- **Remember**: we can enforce inequality constraints with Lagrange multipliers, but they're only $\neq 0$ if the constraint "binds", so $\geqslant$ is really $=$

Consequences of margin maximization: "support vectors"

- One of our constraints is that for each i

$$y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right) \geqslant 1$$

- This is an inequality constraint
- **Remember**: we can enforce inequality constraints with Lagrange multipliers, but they're only $\neq 0$ if the constraint "binds", so $\geqslant$ is really $=$
- The multiplier enforcing this constraint for data point i is α_i

Consequences of margin maximization: "support vectors"

- One of our constraints is that for each i

$$y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right) \geqslant 1$$

- This is an inequality constraint
- **Remember**: we can enforce inequality constraints with Lagrange multipliers, but they're only $\neq 0$ if the constraint "binds", so $\geqslant$ is really $=$
- The multiplier enforcing this constraint for data point i is α_i
- **Consequence**: $\alpha_i \neq 0$ only for the training points where the margin $= 1 \rightsquigarrow$ active constraint
- If $\alpha_i \neq 0$ then $\boldsymbol{x}_i$ is a **support vector**
- Generally, most data points won't be support vectors and will have $\alpha_i = 0$, i.e., the solution will be **sparse** (w.r.t. the training data)

Consequences of margin maximization: "support vectors"

- One of our constraints is that for each i

$$y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right) \geqslant 1$$

- This is an inequality constraint
- **Remember**: we can enforce inequality constraints with Lagrange multipliers, but they're only $\neq 0$ if the constraint "binds", so $\geqslant$ is really $=$
- The multiplier enforcing this constraint for data point i is α_i
- **Consequence**: $\alpha_i \neq 0$ only for the training points where the margin $= 1 \rightsquigarrow$ active constraint
- If $\alpha_i \neq 0$ then $\boldsymbol{x}_i$ is a **support vector**
- Generally, most data points won't be support vectors and will have $\alpha_i = 0$, i.e., the solution will be **sparse** (w.r.t. the training data)

Consequences of margin maximization: the solution

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

- The optimal hyperplane can be written as

$$f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}} \boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle = \beta_0 \quad + \sum_{\substack{s \,:\, \boldsymbol{x}_s \\ \texttt{support point}}} \alpha_s \, y_s \, \langle \boldsymbol{x}_s, \boldsymbol{x} \rangle$$

Consequences of margin maximization: the solution

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

- The optimal hyperplane can be written as

$$f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\texttt{T}} \boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle = \beta_0 \quad + \sum_{\substack{s \,:\, \boldsymbol{x}_s \\ \texttt{support point}}} \alpha_s \, y_s \, \langle \boldsymbol{x}_s, \boldsymbol{x} \rangle$$

- Since support vectors lie on the marginal hyperplanes, for *any* support point, say $\boldsymbol{x}_s$, we get

$$y_s = \beta_0 + \langle \boldsymbol{\beta}_{\texttt{svm}}, \boldsymbol{x}_s \rangle \quad \rightsquigarrow \quad \beta_0 = y_s - \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x}_s \rangle$$

Consequences of margin maximization: the solution

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \qquad \text{and} \qquad \sum_i \alpha_i \, y_i = 0$$

- The optimal hyperplane can be written as

$$f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}} \boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle = \beta_0 \quad + \sum_{\substack{s \,:\, \boldsymbol{x}_s \\ \texttt{support point}}} \alpha_s \, y_s \, \langle \boldsymbol{x}_s, \boldsymbol{x} \rangle$$

- Since support vectors lie on the marginal hyperplanes, for *any* support point, say $\boldsymbol{x}_s$, we get

$$y_s = \beta_0 + \langle \boldsymbol{\beta}_{\texttt{svm}}, \boldsymbol{x}_s \rangle \quad \rightsquigarrow \quad \beta_0 = y_s - \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x}_s \rangle$$

- We can also derive a simple expression for the geometric margin ρ in terms of $\boldsymbol{\alpha}$ by multiplying both side of the last equation by $\alpha_s \, y_s$ and summing up (over support points)

$$\sum_s \alpha_s \, y_s \, \beta_0 = \sum_s \alpha_s \, y_s^2 - \sum_{i,s} \alpha_i \, \alpha_s \, y_i \, y_s \, \langle \boldsymbol{x}_i, \boldsymbol{x}_s \rangle \quad \Leftrightarrow \quad 0 = \sum_i \alpha_i - \|\boldsymbol{\beta}\|_2^2 \quad \Leftrightarrow \quad \rho^2 \overset{\texttt{def}}{=} \frac{1}{\|\boldsymbol{\beta}\|_2^2} = \frac{1}{\sum_s \alpha_s} \overset{\alpha_s > 0}{=} \frac{1}{\|\boldsymbol{\alpha}\|_1}$$

Level 2: Slightly Harder...but Soft...

The Non-Linearly Separable Case

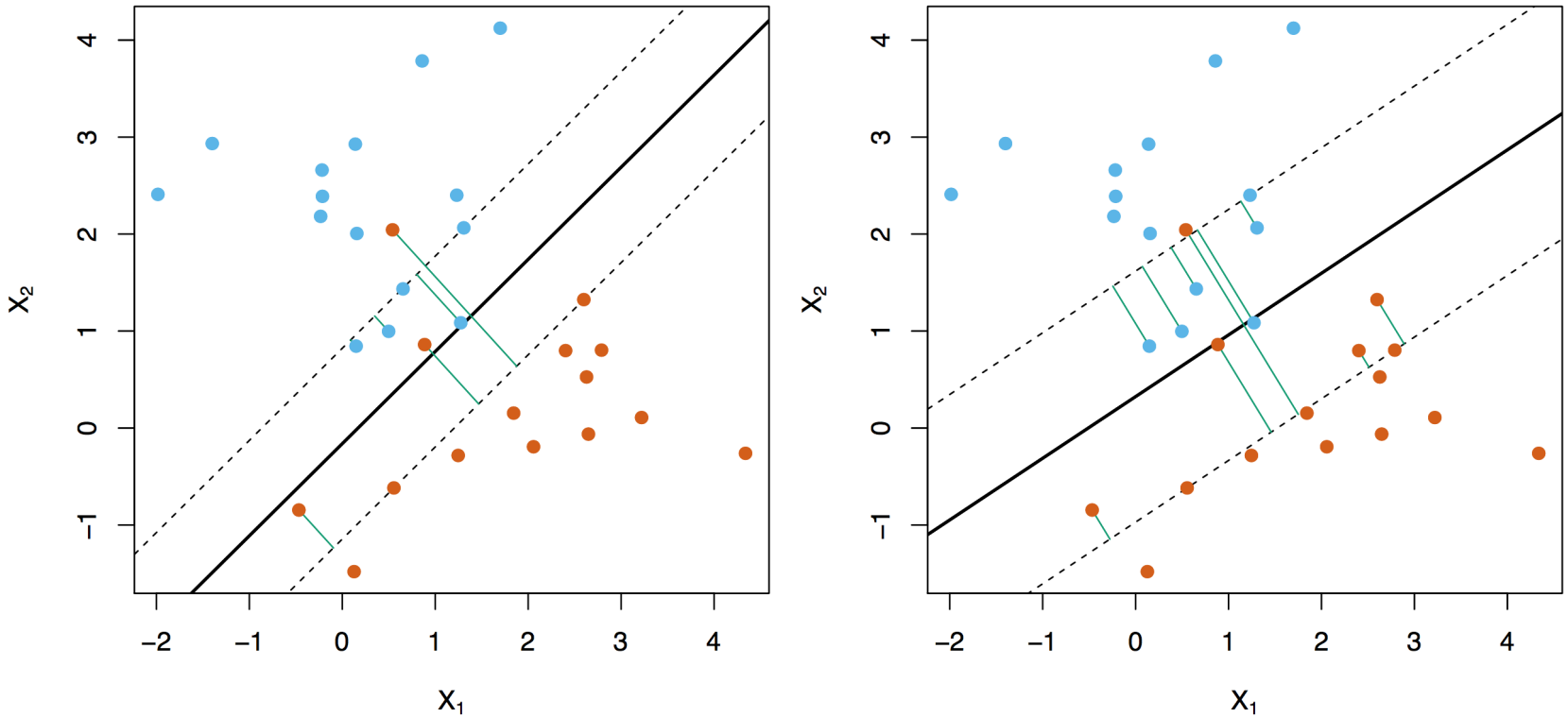
(Soft Margin)

"Soft" margin and slack

- Sometimes we just can't get perfect classification with a plane $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$

"Soft" margin and slack

- Sometimes we just can't get perfect classification with a plane $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$
- **Solution:** allow for margin violations under a *budget constraint* $\rightsquigarrow$ **tuning parameter**



"Soft" margin and slack

- Sometimes we just can't get perfect classification with a plane $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$
- **Solution:** allow for margin violations under a *budget constraint* $\rightsquigarrow$ **tuning parameter**

Before

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: **hard** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \ldots, n\}$

"Soft" margin and slack

- Sometimes we just can't get perfect classification with a plane $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$
- **Solution:** allow for margin violations under a *budget constraint* $\rightsquigarrow$ **tuning parameter**

Before

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**: **hard** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 \text{ for each } i \in \{1, \dots, n\}$

After

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \dots, n\}$
  - **budget contraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \ldots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- The slack $\xi_i > 0$ if $\boldsymbol{x}_i$ is "beyond" the margin $\rightsquigarrow$ we pay a penalty equal to $\sum_i \xi_i$
- C is the budget for the total amount of violations we allow.

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \ldots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- The slack $\xi_i > 0$ if $\boldsymbol{x}_i$ is "beyond" the margin $\rightsquigarrow$ we pay a penalty equal to $\sum_i \xi_i$
- C is the budget for the total amount of violations we allow.
- The solution to this new optimization problem has the same shape as before, except now the support set includes any vector on the margin as well as those that violate the margin.

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \dots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- The slack $\xi_i > 0$ if $\boldsymbol{x}_i$ is "beyond" the margin $\rightsquigarrow$ we pay a penalty equal to $\sum_i \xi_i$
- C is the budget for the total amount of violations we allow.
- The solution to this new optimization problem has the same shape as before, except now the support set includes any vector on the margin as well as those that violate the margin.
- The bigger $C \rightsquigarrow$ the larger the support set $\rightsquigarrow$ the more points that have an impact on the solution.

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \ldots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- The slack $\xi_i > 0$ if $\boldsymbol{x}_i$ is "beyond" the margin $\rightsquigarrow$ we pay a penalty equal to $\sum_i \xi_i$
- C is the budget for the total amount of violations we allow.
- The solution to this new optimization problem has the same shape as before, except now the support set includes any vector on the margin as well as those that violate the margin.
- The bigger $C \rightsquigarrow$ the larger the support set $\rightsquigarrow$ the more points that have an impact on the solution.
- Hence bigger $C \rightsquigarrow$ more stability $\rightsquigarrow$ lower variance.
- In fact even for separable data, allowing margin violations lets us *regularize* the solution.

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \ldots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- **Notice:** the constraint can be succintly captured as follows

$$\forall\ i \in \{1, \ldots, n\},\ \xi_i \geqslant 1 - y_i\, f(\boldsymbol{x}_i) \quad \rightsquigarrow \quad C = \sum_i \xi_i \geqslant \sum_i \big(1 - y_i\, f(\boldsymbol{x}_i)\big) \quad \overset{(\heartsuit)}{\Leftrightarrow} \quad \sum_i \big(1 - y_i\, f(\boldsymbol{x}_i)\big)_+ \leqslant C,$$

where we have introduced the positive part $(u)_+ = \max\{0, u\}$ in $(\heartsuit)$ because we must pay a price $\xi_i > 0$ $\Leftrightarrow$ there's a violation $\Leftrightarrow$ the margin $y_i\, f(\boldsymbol{x}_i)$ is less than 1 $\Leftrightarrow$ $1 - y_i\, f(\boldsymbol{x}_i) > 0$.

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2}\|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i \text{ for each } i \in \{1, \ldots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

- **Notice:** the constraint can be succintly captured as follows

$$\forall\ i \in \{1, \ldots, n\},\ \xi_i \geqslant 1 - y_i\, f(\boldsymbol{x}_i) \quad \rightsquigarrow \quad C = \sum_i \xi_i \geqslant \sum_i \big(1 - y_i\, f(\boldsymbol{x}_i)\big) \quad \overset{(\heartsuit)}{\Leftrightarrow} \quad \sum_i \big(1 - y_i\, f(\boldsymbol{x}_i)\big)_+ \leqslant C,$$

where we have introduced the positive part $(u)_+ = \max\{0, u\}$ in $(\heartsuit)$ because we must pay a price $\xi_i > 0$ $\Leftrightarrow$ there's a violation $\Leftrightarrow$ the margin $y_i\, f(\boldsymbol{x}_i)$ is less than 1 $\Leftrightarrow$ $1 - y_i\, f(\boldsymbol{x}_i) > 0$.

Lagrangian

$$\min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \alpha \sum_i \big(1 - y_i\,(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle)\big)_+ \quad \rightsquigarrow \quad \text{multiply by } \lambda = \frac{1}{\alpha} \text{ (and forget the const. } 1/2)$$

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0, \boldsymbol{\beta})} \frac{1}{2} \|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i$ for each $i \in \{1, \dots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

Lagrangian

$$\min_{(\beta_0, \boldsymbol{\beta})} \sum_i \left(1 - y_i \left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right)\right)_+ + \lambda \|\boldsymbol{\beta}\|_2^2 \quad \overset{\mathcal{F} = \{\texttt{Linear in } \boldsymbol{x}\}}{\longleftrightarrow} \quad \min_{f \in \mathcal{F}} \widehat{R}_{\mathtt{H}}(f) + \lambda \operatorname{pen}(f)$$

More bias/Less variance $\quad \longleftrightarrow \quad$ Large $C \quad \longleftrightarrow \quad$ Small $\alpha \quad \longleftrightarrow \quad$ Large $\lambda \quad \longleftrightarrow \quad$ Heavily regularised

"Soft" margin and slack

- **Maximize** the margin $M(f) \quad \rightsquigarrow \quad \min_{(\beta_0,\boldsymbol{\beta})} \frac{1}{2}\|\boldsymbol{\beta}\|_2^2$
- **Subject to**:
  - **soft** separability condition $\quad \rightsquigarrow \quad y_i(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle) \geqslant 1 - \xi_i$ for each $i \in \{1, \dots, n\}$
  - **budget constraint** $\quad \rightsquigarrow \quad \xi_i \geqslant 0 \quad \text{and} \quad \sum_i \xi_i = C$

Lagrangian

$$\min_{(\beta_0,\boldsymbol{\beta})} \sum_i \left(1 - y_i\left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right)\right)_+ + \lambda \|\boldsymbol{\beta}\|_2^2 \quad \overset{\mathcal{F}=\{\texttt{Linear in } \boldsymbol{x}\}}{\longleftrightarrow} \quad \min_{f \in \mathcal{F}} \widehat{R}_{\mathtt{H}}(f) + \lambda \operatorname{pen}(f)$$

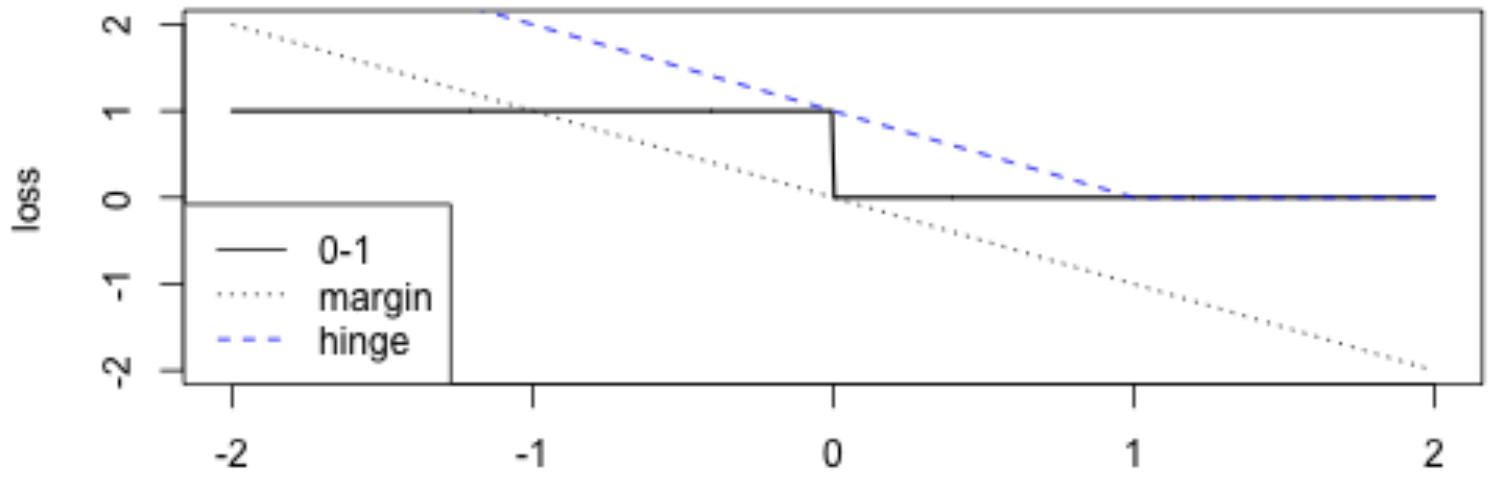
$$\text{More bias/Less variance} \quad \longleftrightarrow \quad \text{Large } C \quad \longleftrightarrow \quad \text{Small } \alpha \quad \longleftrightarrow \quad \text{Large } \lambda \quad \longleftrightarrow \quad \text{Heavily regularised}$$

Conclusion

As promised, from purely geometric principles
we obtained (almost) for free two fundamental analytic learning tools:
Surrogate Loss + **Regularization**

Hinge loss/risk

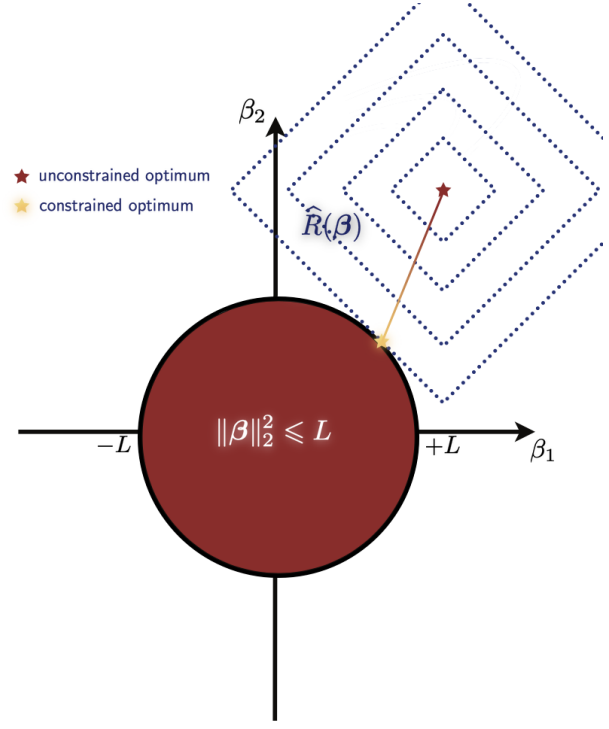
$$L\big(y, f(x)\big) = \big(1 - y\, f(\boldsymbol{x})\big)_+ \equiv \max\big\{0, 1 - y\, f(\boldsymbol{x})\big\}$$



- **Remember**: the 0-1 loss is always $\leqslant$ hinge loss, so if we can bound the hinge loss we will also bound the miss-classification/accuracy risk!

SVM: Remarks

Penalty $\leftrightsquigarrow$ Constraint



L2 regularization

(...weight decay, ridge penalty, Tikhonov regularization, Gaussian prior...)

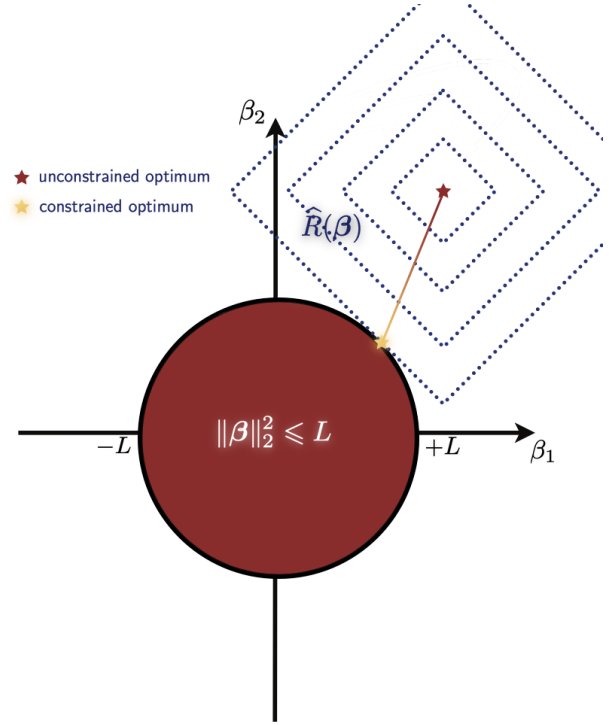
- Hinge and penalty are both **convex** in $\boldsymbol{\beta} \rightsquigarrow$ for each λ we can find an L s.t. the following constrained problem has the same solution

$$\min_{(\beta_0,\boldsymbol{\beta})} \widehat{R}_{\mathrm{H}}(\beta_0, \boldsymbol{\beta}) \quad \text{subject to} \quad \|\boldsymbol{\beta}\|_2^2 \leqslant L$$

- Typically the bias/intercept β_0 is *not* penalized.
- Typically we do **not** solve the optimization problem in this form, but we go for the *primal-dual* way detailed before.
- This rewriting hints to a plethora of variations:
  - **Change penalty:** $\|\boldsymbol{\beta}\|_2 = L^2 \rightsquigarrow L^1 = \|\boldsymbol{\beta}\|_1$ to force *sparsity* in $\boldsymbol{\beta}$ (e.g. LASSO, more soon) This is known as L^1-SVM (`penalizedSVM`)
  - *Pro*: it can be written as a **linear** programming problem
  - *Con*: **no** longer a maximum margin classifier!

SVM: Remarks

Penalty $\leftrightsquigarrow$ Constraint



L2 regularization

(...weight decay, ridge penalty, Tikhonov regularization, Gaussian prior...)

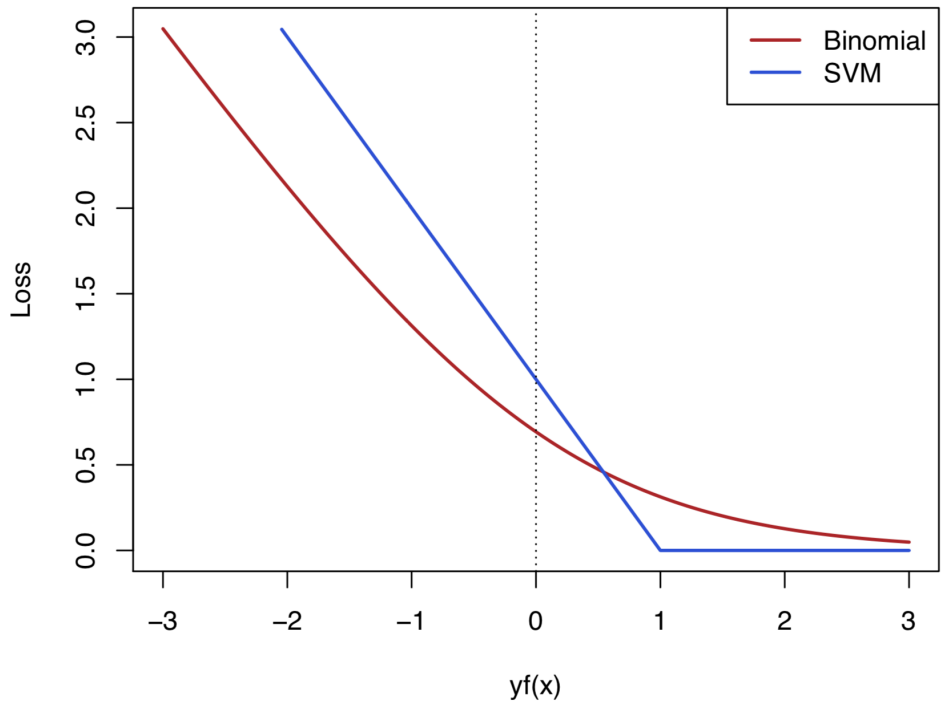
- Hinge and penalty are both **convex** in $\boldsymbol{\beta} \rightsquigarrow$ for each λ we can find an L s.t. the following constrained problem has the same solution

$$\min_{(\beta_0,\boldsymbol{\beta})} \widehat{R}_{\mathrm{H}}(\beta_0, \boldsymbol{\beta}) \quad \text{subject to} \quad \|\boldsymbol{\beta}\|_2^2 \leqslant L$$

- Typically the bias/intercept β_0 is *not* penalized.
- Typically we do **not** solve the optimization problem in this form, but we go for the *primal-dual* way detailed before.
- This rewriting hints to a plethora of variations:
  - **Change penalty:** $\|\boldsymbol{\beta}\|_2 = L^2 \rightsquigarrow L^1 = \|\boldsymbol{\beta}\|_1$ to force *sparsity* in $\boldsymbol{\beta}$ (e.g. LASSO, more soon)
  - **Change loss:** to handle different tasks (e.g. Regression) $\rightsquigarrow$ if done properly the *primal-dual* route may still be feasible
  - **Change optimizer:** to handle big data, go SGD by (numerically) differentiate it

SVM: Remarks

$$\text{Penalty} \leftrightsquigarrow \text{Constraint}$$



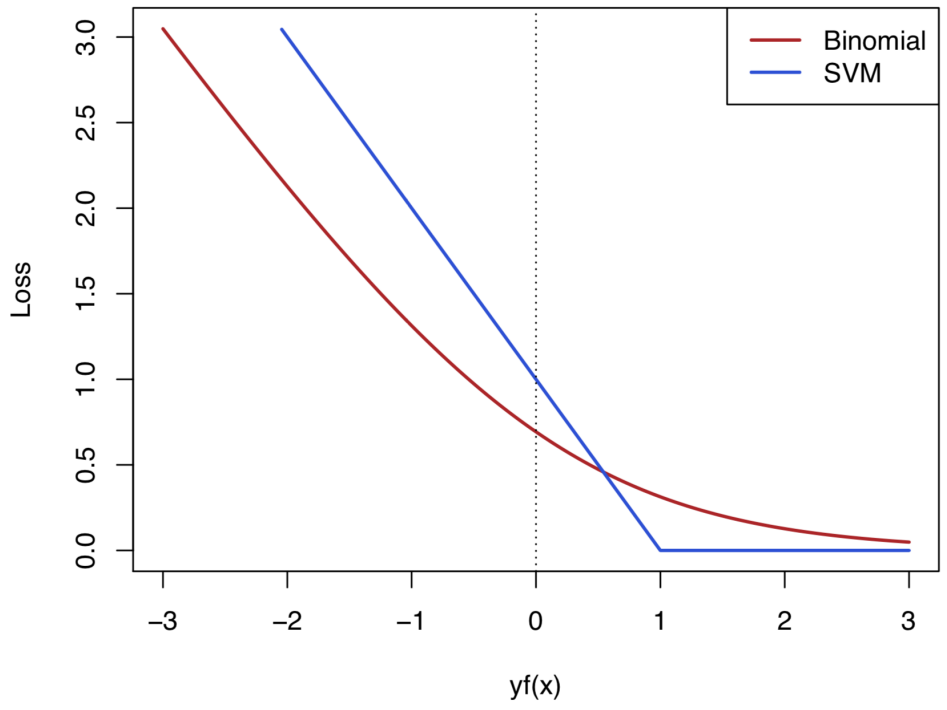
- *Penalized Empirical Risk* methods are ubiquitous: given two solutions $\widehat{\boldsymbol{\beta}}_1$ and $\widehat{\boldsymbol{\beta}}_2$with the same (empirical) risk, we prefer the more parsimonious of the two (as measured by the selected penalty/norm)
- For example, we know the **logistic-loss** needs some help (even under linear separability)
- We hint at *early-stopping* as a *regularizer*: the number of iterations *is* the tuning parameter to select via cross-validation, for example
- Another option is clearly to penalize the logistic deviance/loss, here for $Y \in \{-1, +1\}$

$$\min_{(\beta_0, \boldsymbol{\beta})} \sum_{i=1}^{n} \log \left[1 + \mathrm{e}^{-y_i \, (\beta_0 + \langle \boldsymbol{x}_i, \boldsymbol{\beta} \rangle)}\right] + \lambda \, \|\boldsymbol{\beta}\|_2^2.$$

Under some conditions, *early stoppping* $\approx L^2$ (see ESL, Section 10.12 or 16.2.3)

SVM: Remarks

$$\text{Penalty} \leftrightsquigarrow \text{Constraint}$$



$$\min_{(\beta_0,\boldsymbol{\beta})} \sum_{i=1}^{n} \log\left[1 + \mathrm{e}^{-y_i\,(\beta_0 + \langle \boldsymbol{x}_i, \boldsymbol{\beta}\rangle)}\right] + \lambda\,\|\boldsymbol{\beta}\|_2^2.$$

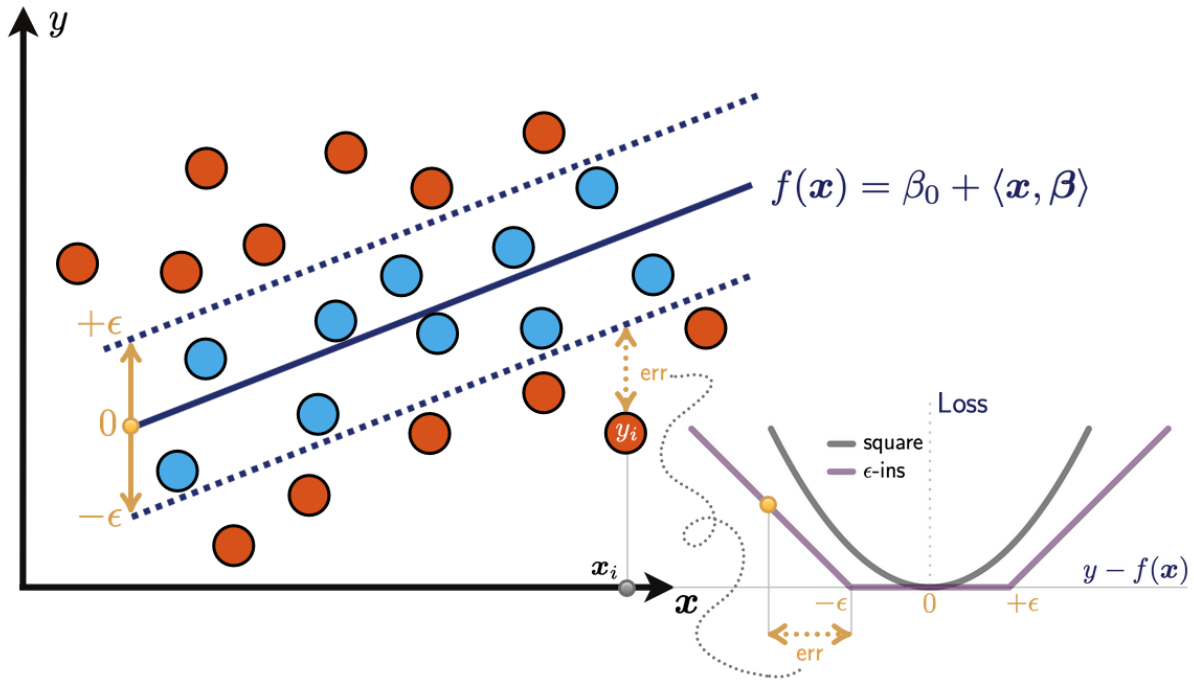
- *Hinge*: elbow at $+1$ (vs) *Logistic*: smooth bend $\rightsquigarrow$ the logistic-solution involves **all** the data: the weights $f_i(1-f_i)$ fade smoothly with distance from the decision boundary as opposed to the binary nature of support points.
- Rosset et al. (2004): for $\lambda \downarrow 0$, the scaled logistic coeff's $\boldsymbol{\beta}/\|\boldsymbol{\beta}\|_2$ converges to the SVM solution under linear separability of the data.
  - Because of the required normalization for the logistic, the SVM solution is preferable.
  - On the other hand, for non-linearly separable cases, the logistic solution may have some advantages since its target is the *logit* (log-odds) of class probabilities.
- Check the `liblinearR` package.

SVM: Remarks

Support Vector Regression

- **Hope:** by replacing the hinge with a suitable loss, we may experience the beauty of SVM (i.e. easy optimization problem that depends on inner-products *only* + data-sparse solution) also in *regression*...

$$L\big(y, f(\boldsymbol{x})\big) = |\, y - f(\boldsymbol{x}) \,|_\epsilon = \max\big\{0, |\, y - f(\boldsymbol{x}) \,| - \epsilon\big\} \qquad \epsilon\text{-insensitive loss}$$

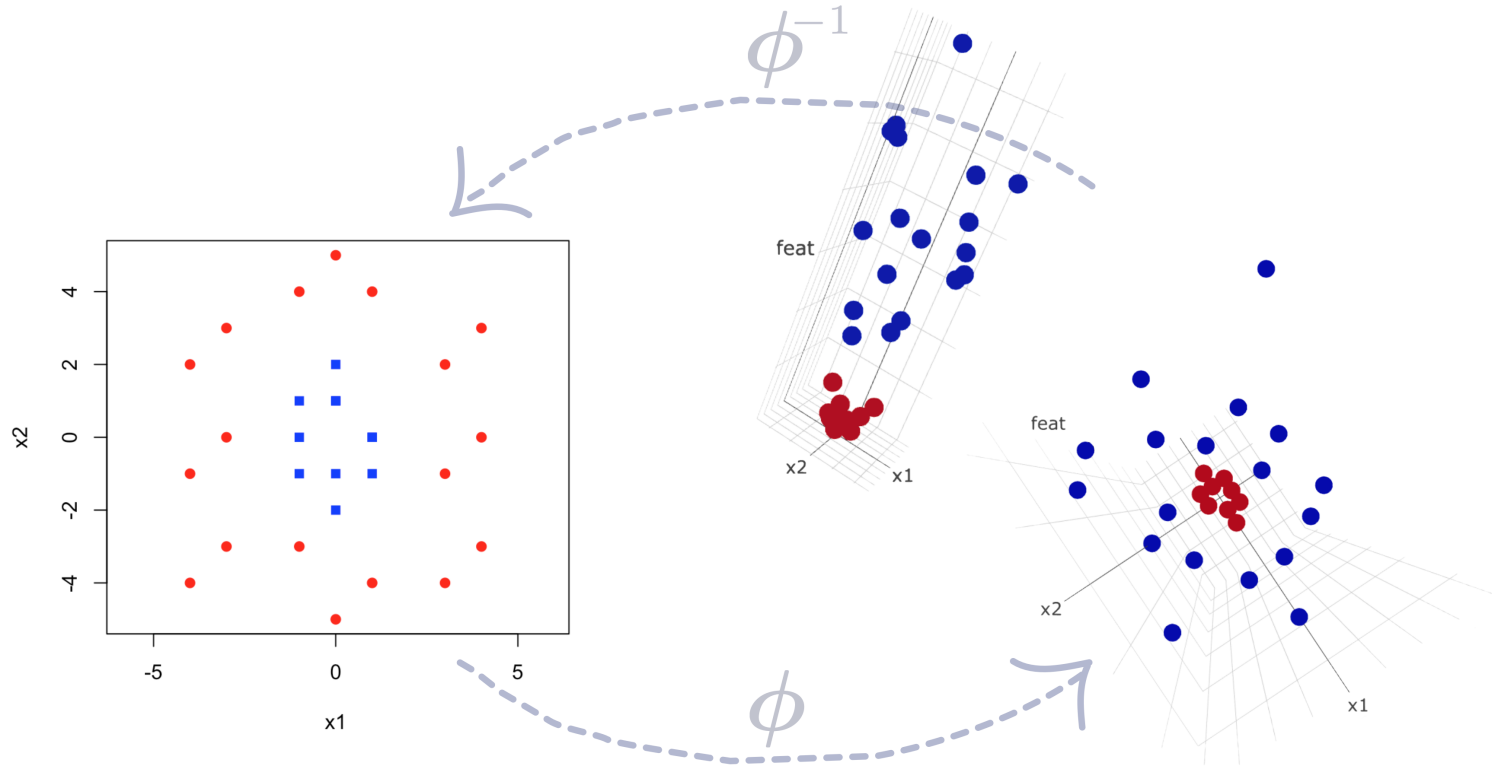


Level 3: Kernelization

From Linear to Nonlinear in a Trick!

Explicit Nonlinearity in Classification

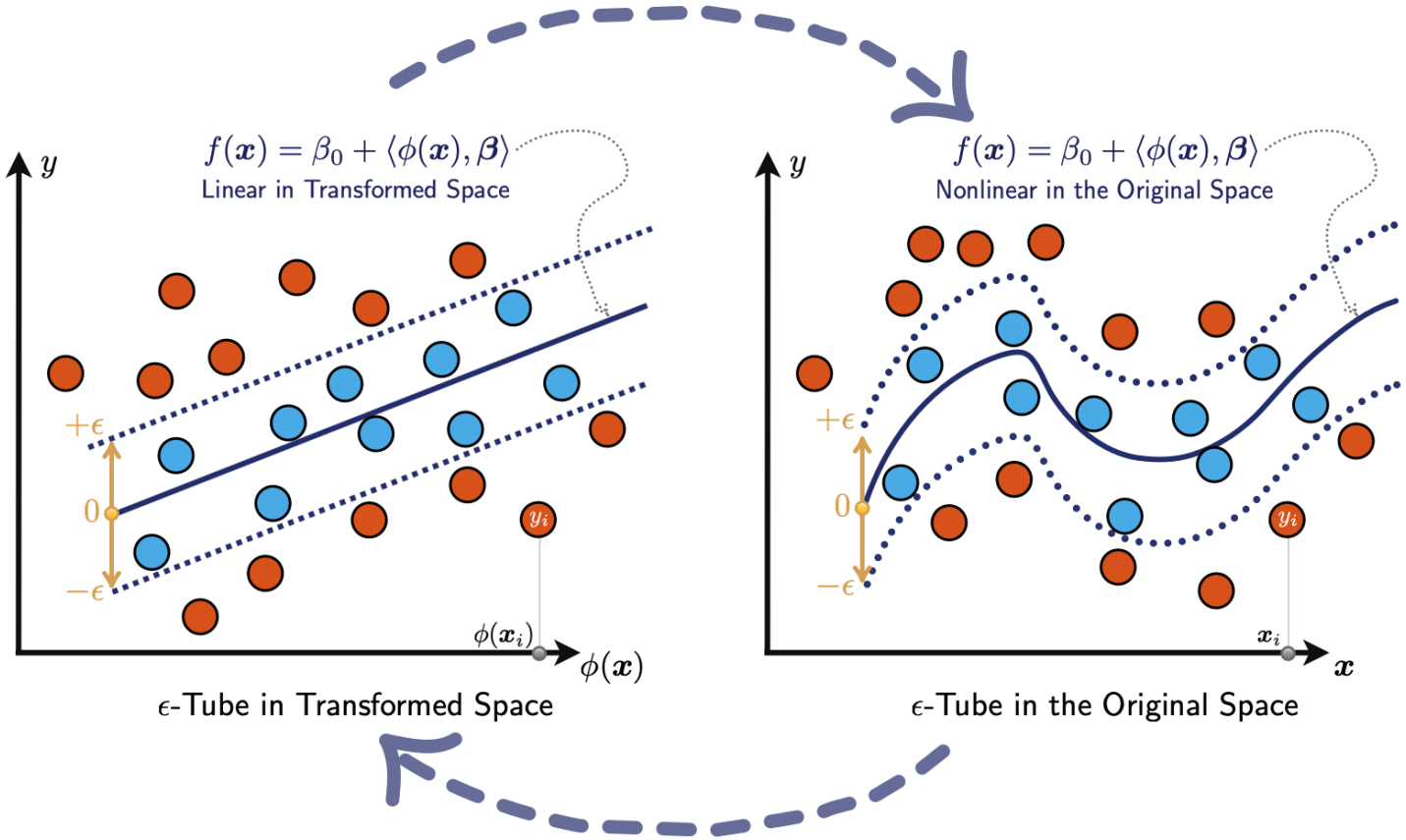
Nonlinearly Separable in the Original Space $\leftrightsquigarrow$ Linearly Separable in Transformed Space



$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\intercal \rightsquigarrow \phi(\boldsymbol{x}) = \boldsymbol{\phi} = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 \end{bmatrix}^\intercal = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix}^\intercal$$

Explicit Nonlinearity in Regression

Support Vector Regression



Back in the SVM-days...

Dual

$$F(\boldsymbol{\alpha}) = \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

$$\max_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) \quad \text{subject to} \quad \alpha_i \geqslant 0 \quad \forall \; i \in \{1, \dots, n\} \quad \text{and} \quad \sum_i \alpha_i \, y_i = 0$$

Optimal hyperplane

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_i \alpha_i \, y_i \, \boldsymbol{x}_i \rightsquigarrow f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^{\mathsf{T}} \boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \sum_i \alpha_i \, y_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle = \beta_0 + \sum_{\substack{s \, : \, \boldsymbol{x}_s \\ \texttt{support point}}} \alpha_s \, y_s \, \langle \boldsymbol{x}_s, \boldsymbol{x} \rangle$$

Consequences

Working in the original $\mathcal{X}$ space, we do **not** need the $\boldsymbol{x}_i$'s
Training: store **inner products** (*similarity measure*) between pairs of *training data* $\rightsquigarrow \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$
Prediction: evaluate **inner products** between the new feature vector $\boldsymbol{x}$ and the *training data* $\rightsquigarrow \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$

Back in the SVM-days...

Dual (matrix form)

$$F(\boldsymbol{\gamma}) = \mathbf{1}^{\mathsf{T}}\boldsymbol{\alpha} - \tfrac{1}{2}\left(\mathbb{Y}\boldsymbol{\alpha}\right)^{\mathsf{T}}\mathbb{K}\left(\mathbb{Y}\boldsymbol{\alpha}\right) \overset{\boldsymbol{\gamma}=\mathbb{Y}\boldsymbol{\alpha}}{=} \mathbf{1}^{\mathsf{T}}\left(\mathbb{Y}^{-1}\boldsymbol{\gamma}\right) - \tfrac{1}{2}\boldsymbol{\gamma}^{\mathsf{T}}\mathbb{K}\boldsymbol{\gamma} \quad \text{with} \quad \mathbf{1}^{\mathsf{T}} = [\,1 \;\; 1 \cdots 1\,] \in \mathbb{R}^{n} \quad \text{and} \quad \mathbb{Y} = \text{diag}(\boldsymbol{y})$$

$$\max_{\boldsymbol{\gamma}} F(\boldsymbol{\gamma}) \quad \text{subject to} \quad \left(\mathbb{Y}^{-1}\boldsymbol{\gamma}\right) \succcurlyeq 0 \quad \text{and} \quad \mathbf{1}^{\mathsf{T}}\boldsymbol{\gamma} = 0$$

Optimal hyperplane (matrix form)

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_{i=1}^{n} \gamma_i \, \boldsymbol{x}_i = \mathbb{X}^{\mathsf{T}}\boldsymbol{\gamma} \rightsquigarrow f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \underbrace{\sum_{j=1}^{p} \beta_j x_j}_{\substack{\texttt{sum over features} \\ \texttt{(primal)}}} = \beta_0 + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \underbrace{\sum_{i=1}^{n} \gamma_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle}_{\substack{\texttt{sum over training-points} \\ \texttt{(dual)}}} \overset{(\cdot)^{\mathsf{T}}=(\cdot)}{=} \beta_0 + \boldsymbol{\gamma}^{\mathsf{T}}\mathbb{X}\,\boldsymbol{x}$$

Consequences

Working in the original $\mathcal{X}$ space, we do **not** need the $\boldsymbol{x}_i$'s
Training: store the $(n \times n)$ **Gramian matrix** $\mathbb{K} = (\mathbb{X}\mathbb{X}^{\mathsf{T}}) \leftrightsquigarrow \mathbb{K}[i,j] = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$
Prediction: evaluate the **inner product** vector $\mathbb{X}\,\boldsymbol{x}$ (on support points only!)

Interlude: $f(\boldsymbol{x}) = \beta_0 + \langle \boldsymbol{x}, \boldsymbol{\beta} \rangle = \beta_0 + \boldsymbol{x}^\intercal \boldsymbol{\beta}$

Support Vector (Classification)	+	Support Vector (Regression)	vs	Ordinary Least Squares
$\frac{1}{n}\sum_{i=1}^{n}\left(1 - y_i f(\boldsymbol{x}_i)\right)_+$	vs	$\frac{1}{n}\sum_{i=1}^{n}\left\| y_i - f(\boldsymbol{x}_i) \right\|_\epsilon$	vs	$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(\boldsymbol{x}_i)\right)^2$

- *Computations*: differently from OLS/square-loss, hinge and ϵ-insensitive do **not** allow for a closed form solution, but we are happy with the efficiency of their *primal/dual* quadratic-programming formulation.
- *Primal/Dual view*: we have seen that the SVM-solution can be more naturally expressed in *dual* form:

$$\boldsymbol{\beta}_{\texttt{svm}} = \sum_{i=1}^{n} \gamma_i \, \boldsymbol{x}_i = \mathbb{X}^\intercal \boldsymbol{\gamma} \rightsquigarrow f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \underbrace{\sum_{j=1}^{p} \beta_j x_j}_{\substack{\texttt{sum over features} \\ \texttt{(primal)}}} = \beta_0 + \boldsymbol{x}^\intercal \boldsymbol{\beta}_{\texttt{svm}} = \beta_0 + \underbrace{\sum_{i=1}^{n} \gamma_i \, \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle}_{\substack{\texttt{sum over training-points} \\ \texttt{(dual)}}} \overset{(\cdot)^\intercal = (\cdot)}{=} \beta_0 + \gamma^\intercal \mathbb{X} \, \boldsymbol{x}$$

This is only apparently in contrast with the familiar closed form solution to the OLS problem. In fact...

$$\boldsymbol{\beta}_{\texttt{ols}} = (\mathbb{X}^\intercal \mathbb{X})^{-1} \mathbb{X}^\intercal \boldsymbol{y} \quad \rightsquigarrow \quad f_{\texttt{ols}}(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^\intercal \boldsymbol{\beta}_{\texttt{ols}} \overset{(\cdot)^\intercal = (\cdot)}{=} \beta_0 + \boldsymbol{y}^\intercal (\mathbb{X}\mathbb{X}^\intercal)^{-1} \mathbb{X} \, \boldsymbol{x} = \beta_0 + \boldsymbol{\gamma}_{\texttt{ols}}^\intercal \mathbb{X} \, \boldsymbol{x}$$

where now the optimal (*dual*) coefficients admit a closed form $\boldsymbol{\gamma}_{\texttt{ols}} = (\mathbb{K}^{-1})^\intercal \boldsymbol{y} \overset{\texttt{sym}}{=} \mathbb{K}^{-1} \boldsymbol{y} \rightsquigarrow$ **kernelizable**!

Toward Nonlinearity...and Beyond!

A) Classic route ⇜⇝ Explicit transformation

Idea: Embed the problem in a larger one by **explicitly** defining a non-linear transformation $\phi(\cdot)$, the **feature map**, which associates a possibly very high dimensional vector $\boldsymbol{\phi} = \phi(\boldsymbol{x}) \in \mathbb{R}^q$ to any feature profile $\boldsymbol{x} \in \mathcal{X}$.

Goal: Engineer the map $\phi(\cdot)$ in such a way that: 1. in the transformed space, linearity is enough to achieve our predictive goals, 2. there is a computationally efficient way to work it out

- *Remark*: generally speaking, for 1. we need some faith; while for 2, we have solid results

Practice: once the *feature map* $\phi(\cdot)$ has been designed, we can simply work **explicitly** in the new covariates space - as in the case of *polynomial/spline regression* - rewriting the optimization problem in an obvious way

- *Remark*: solving it, we still get a *linear* SVM-classifier **but** in the ϕ-transformed space!

$$F(\boldsymbol{\alpha}) = \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{R}^q} \quad \text{and} \quad f_{\texttt{svm}}(\boldsymbol{x}) = \beta_0 + \sum_i \alpha_i \, y_i \, \langle \boldsymbol{\phi}_i, \phi(\boldsymbol{x}) \rangle_{\mathbb{R}^q}$$

Analysis: the SVM computations scale linearly in q (although potentially cubic in n) $\rightsquigarrow$ doable but typically **not** efficient for very large q (in the tens of thousands or even millions) $\rightsquigarrow$ can we avoid remapping all the $\boldsymbol{x}_i$?

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftsquigarrow$ Implicit transformation

Idea: since what we really need is only the inner product $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{R}^q}$, we may try do design $\phi(\cdot)$ so that $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{R}^q}$ admits a simple representation in terms of the original covariate $\{\boldsymbol{x}_i\}_i \rightsquigarrow$ **no** explicit evaluation of the embeddings $\{\boldsymbol{\phi}_i\}_i$ is needed!

Example 1: let's go back to the initial example, and define the following *feature map* $\phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$:

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\intercal \rightsquigarrow \phi(\boldsymbol{x}) = \boldsymbol{\phi} = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 \end{bmatrix}^\intercal = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix}^\intercal$$

Given two feature profiles $\boldsymbol{x}$ and $\boldsymbol{y}$, let us now evaluate the inner-product:

$$\begin{aligned} \big\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \big\rangle_{\mathbb{R}^3} &= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix} \begin{bmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}\, y_1\, y_2 \end{bmatrix} = x_1^2 y_1^2 + x_2^2 y_2^2 + 2(x_1\, y_1)(x_2\, y_2) \\ &= (x_1\, y_1 + x_2\, y_2)^2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathbb{R}^2}^2 = K(\boldsymbol{x}, \boldsymbol{y}) \quad \text{(example of polynomial kernel)} \end{aligned}$$

Mission accomplished!

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Idea: since what we really need is only the inner product $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{R}^q}$, we may try do design $\phi(\cdot)$ so that $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{R}^q}$ admits a simple representation in terms of the original covariate $\{\boldsymbol{x}_i\}_i \rightsquigarrow$ **no** explicit evaluation of the embeddings $\{\boldsymbol{\phi}_i\}_i$ is needed!

Example 1: more in general, functions of two variables $\boldsymbol{x}$ and $\boldsymbol{y}$ both in $\mathcal{X} \subseteq \mathbb{R}^p$ with the following form

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(c + \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathbb{R}^p}\right)^d \quad \text{with} \quad d \in \mathbb{N}, c > 0 \quad \text{(to be tuned via CV)}$$

are called **polynomial kernels**, and *implicitly* code for an inner product in a high-dimensional space.

- Specifically, once we expand the power d and express K in terms of *all* monomials $x_1^{k_1} x_2^{k_2} \cdots x_p^{k_p}$, we are left with the problem of determining the number of monomials of degree exactly d in p input variables
- This is equivalent to find the number of ways of adding p non-negative integers to obtain a sum of at most d that can be shown to be equal to $\binom{p+d}{d}$
- In addition, by the binomial theorem, we deduce that the weight assigned to the monomial $x_j^{k_j}$ is $\binom{d}{j} c^{d-j}$ for each $j \leqslant d \rightsquigarrow$ increasing c decreases the weight of k_j, particularly that of k_j's with larger j.

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 2: *Gaussian Kernel/Radial Basis Function Kernel*

Important Remark
As long as we can compute inner products, there is **no** restriction on the dimension of the feature space.

Immediate Consequence
The most commonly used kernel corresponds to "projection" into an **infinite** dimensional space...*implicitly*, of course!

Assume for simplicity that $x \in \mathcal{X} = \mathbb{R} \rightsquigarrow d = 1$ and define

$$\phi_j(x) \quad = \quad \underbrace{x^j}_{\texttt{power part}} \quad \times \quad \underbrace{\left(\frac{2^j}{j!}\right)^{\frac{1}{2}}}_{\downarrow j\,:\,\texttt{damping factor}} \quad \times \quad \underbrace{\mathrm{e}^{-x^2}}_{\texttt{normalization}} \qquad j \in \{0, 1, 2, \ldots\} = \mathbb{N}_0 \;\rightsquigarrow\; \boldsymbol{\phi} = \begin{bmatrix} \phi_0(x) & \phi_1(x) & \phi_2(x) \cdots \end{bmatrix}^{\mathtt{T}}$$

Remarks:

- Notice we are defining a *feature map* $\phi : \mathcal{X} \subseteq \mathbb{R} \mapsto \ell^2 \approx \mathbb{R}^\infty$ in a *infinite* dimensional *sequence* space!
- *Normalization*: $\|\boldsymbol{\phi}\|_{\ell^2}^2 = \sum_{j=0}^\infty \phi_j^2(x) = \sum_{j=0}^\infty x^{2j} \left(\frac{2^j}{j!}\right) \left(\mathrm{e}^{-x^2}\right)^2 = \mathrm{e}^{-2x^2} \sum_{j=0}^\infty \frac{(2\,x)^j}{j!} \overset{\texttt{Taylor}}{=} \mathrm{e}^{-2x^2} \mathrm{e}^{2x^2} = 1$ (normalized!)

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Example 2: *Gaussian Kernel/Radial Basis Function Kernel*

$$\phi_j(x) = x^j \times \left(\tfrac{2^j}{j!}\right)^{\frac{1}{2}} \times \mathrm{e}^{-x^2} \;\; j \in \{0,1,2,\ldots\} = \mathbb{N}_0 \quad \rightsquigarrow \quad \boldsymbol{\phi} = \begin{bmatrix} \phi_0(x) & \phi_1(x) & \phi_2(x) \cdots \end{bmatrix}^\intercal$$

- In essence, we are mapping any $x \in \mathbb{R}$ into a norm 1, infinite dimensional vector $\boldsymbol{\phi}$ or, in other words, we are remapping the entire input space $\mathcal{X} = \mathbb{R}$ into the unit ball of the sequence space ℓ^2.
- As with the *polynomial kernel*, let us now look at the actual shape of the kernel implied by these $\{\phi_j\}_j$. Let x and y be two input values and evalute their inner product once transformed by our *feature map* ϕ:

$$\big\langle \phi(x), \phi(y) \big\rangle_{\ell^2} = \mathrm{e}^{-x^2}\mathrm{e}^{-y^2} \sum_{j=0}^{\infty} \frac{(2\,x\,y)^2}{j!} = \mathrm{e}^{-(x^2 - 2\,x\,y + y^2)} = \mathrm{e}^{-(x-y)^2} = K(x,y) \quad \text{(Gaussian kernel)}$$

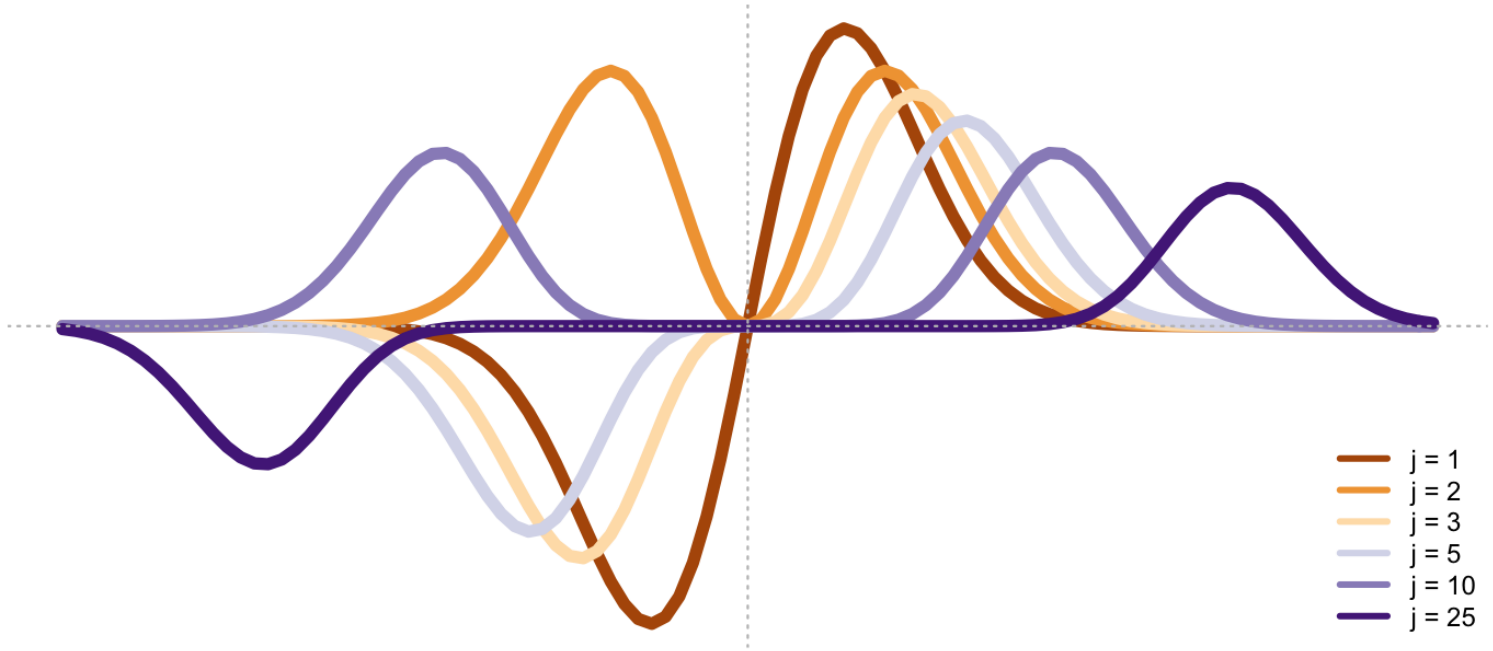
- If $\boldsymbol{x} \in \mathbb{R}^p$, we can define a (parametrized) Gaussian kernel more generally as

$$K(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{e}^{-\frac{1}{\sigma^2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2} \quad (\sigma^2 > 0 \text{ to be tuned via CV})$$

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝⇝ Implicit transformation

Example 2: *Gaussian Kernel/Radial Basis Function Kernel*



Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

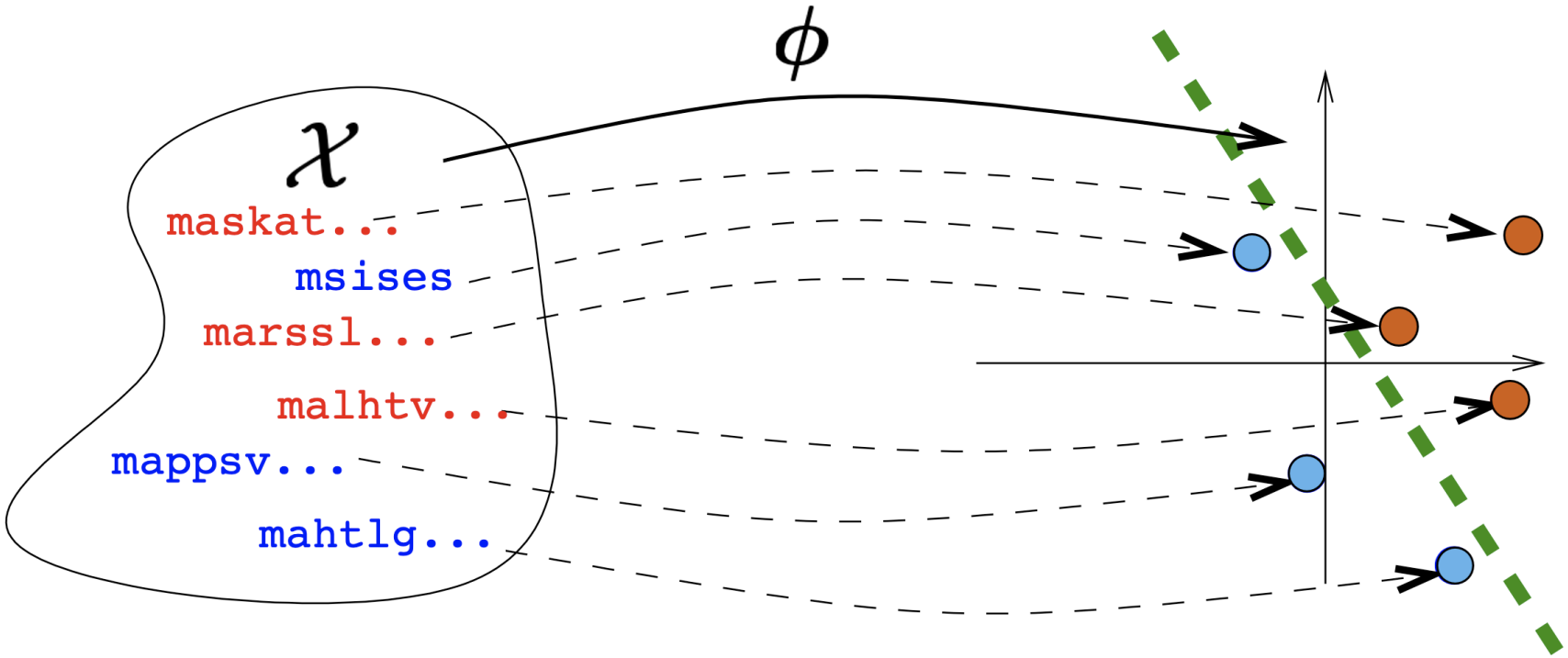
Example 3: *String Kernels*

- The examples given so far were all for kernels over Euclidean vector spaces.
  In many learning tasks found in practice, the input space $\mathcal{X}$ is **not** a vector space.
  - A few examples: classify protein sequences, images, graphs, parse trees, finite automata, or other discrete structures which may not be directly given as vectors.
- The kernel tricks provide a method for extending algorithms such as SVMs originally designed for a vectorial space to the classification of such objects.
- Here we will focus on the specific case of *sequence kernels*, that is, kernels for *sequences* or *strings*.
  - Kernels can be defined for other discrete structures in somewhat similar ways.
  - *Sequence kernels* are particularly relevant to learning algorithms applied to computational biology or natural language processing, which are both important applications.

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 3: *String Kernels*



Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝⇝ Implicit transformation

Example 3: *String Kernels*

- Kernels, i.e. similarity measures, for sequences, can be based on the idea of declaring two strings, e.g., two documents or two biosequences, as similar when they share **common substrings** or **subsequences**.
- One example could be the kernel between two sequences defined by the *sum of the product of the counts of their common substrings*.
    - But which substrings should be used in that definition?
    - Most likely, we would need some flexibility in the definition of the matching substrings.
    - For computational biology applications, for example, the match could be imperfect. Thus, we may need to consider some number of mismatches, possibly gaps, or wildcards.
    - More generally, we might need to allow various substitutions and might wish to assign different weights to common substrings to emphasize some matching substrings and deemphasize others.
- Hence, as it is clear, there are many different possibilities and frameworks for defining such kernels.
- Here we will consider **spectrum kernel** and **substring kernel**. For a more general approach based on *rational kernel*, see Section 6.5 of Mohri et al. (2018).

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Example 3: *String Kernels*

General Approach

- Remap the string-space $\mathcal{X}$ into *fixed-length strings*; that is, for each string $\boldsymbol{x} \in \mathcal{X}$ define the *feature map*

$$\phi(\boldsymbol{x}) = \left[\phi_u(\boldsymbol{x})\right]_{u \in \mathcal{A}^k} \quad \text{with} \quad \mathcal{A}^k = \{\text{strings of length } \ k \ \text{ from a given alphabet } \mathcal{A}\}$$

  where $\phi_u(\boldsymbol{x})$ can be:

  - The number of occurrences of u in $\boldsymbol{x}$ (without gaps) $\rightsquigarrow$ **spectrum kernel** (Leslie et al., 2002)
  - The number of occurrences of u in $\boldsymbol{x}$ up to m mismatches (without gaps) $\rightsquigarrow$ **mismatch kernel** (Leslie et al., 2004)
  - The number of occurrences of u in $\boldsymbol{x}$ allowing gaps, with a weight decaying exponentially with the number of gaps $\rightsquigarrow$ **all-substrings kernel** (Lohdi et al., 2002)

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 3a: *Spectrum Kernels* $= \{\text{count how many (contiguous) length-}k\text{ substrings }\boldsymbol{x}\text{ and }\boldsymbol{y}\text{ share}\}$

- For example, the **3-spectra** (with no gaps) of $\boldsymbol{x} = \texttt{STATISTICS}$ and $\boldsymbol{y} = \texttt{COMPUTATION}$ are:

  $$(\texttt{STA}, \texttt{TAT}, \texttt{ATI}, \texttt{TIS}, \texttt{IST}, \texttt{STI}, \texttt{TIC}, \texttt{ICS}) \quad \text{and} \quad (\texttt{COM}, \texttt{OMP}, \texttt{MPU}, \texttt{PUT}, \texttt{UTA}, \texttt{TAT}, \texttt{ATI}, \texttt{TIO}, \texttt{ION})$$

  They have in common the substrings $\texttt{TAT}$ and $\texttt{ATI}$ $\rightsquigarrow K(\boldsymbol{x}, \boldsymbol{y}) = 2$

- More in general, the k-**spectrum kernel** is then:

  $$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{u \in \mathcal{A}^k} \phi_u(\boldsymbol{x}) \phi_u(\boldsymbol{y}) \quad \text{with} \quad \phi_u(\boldsymbol{x}) = \{\text{Number of occurrences of } u \text{ in } \boldsymbol{x}\}$$

- The computation of the kernel is formally a sum over $\left|\text{card}(\mathcal{A})\right|^k$ terms, but at most $\text{card}(\boldsymbol{x}) - k + 1$ terms are non-zero in $\phi(\boldsymbol{x})$ $\rightsquigarrow$ computation in $O(\text{card}(\boldsymbol{x}) + \text{card}(\boldsymbol{y}))$ with strings pre-indexation.
  - Fast classification of a sequence $\boldsymbol{x}$ in $O(\text{card}(\boldsymbol{x}))$
  - Work with any string (natural language, time series, etc) + **fast** and **scalable**

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftsquigarrow$ Implicit transformation

Example 3b: *All-Substrings Kernels*

- For $1 \leqslant k \leqslant m \in \mathbb{N}$, denote by $\mathcal{I}(k, m)$ the set of **sequences of indices** $\boldsymbol{i} = (i_1, \ldots, i_k)$, with $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant m$
- For a length m string $\boldsymbol{x} = x_1 x_2 \cdots x_m \in \mathcal{X}$, and a sequence of indices $\boldsymbol{i} \in \mathcal{I}(k, m)$, define a *substring* as

$$\boldsymbol{x}(\boldsymbol{i}) = x_{i_1} x_{i_2} \cdots x_{i_k} \quad \text{and} \quad \ell(\boldsymbol{i}) = \big\{ \text{length of } \boldsymbol{x}(\boldsymbol{i}) \big\} = i_k - i_1 + 1$$

- For example, if $\boldsymbol{x} = \texttt{ABRACADABRA}$, then

$$\boldsymbol{i} = (3, 4, 7, 8, 10) \quad \rightsquigarrow \quad \boldsymbol{x}(\boldsymbol{i}) = \texttt{RADAR} \quad \rightsquigarrow \quad \ell(\boldsymbol{i}) = 10 - 3 + 1 = 8$$

- Let $k \in \mathbb{N}$ and $\lambda > 0$ be tuning parameter to be chosen via CV $\rightsquigarrow$ let $\phi_u : \mathcal{X} \mapsto \mathbb{R}$ be defined as

$$\phi_u(\boldsymbol{x}) \quad = \sum_{\boldsymbol{i} \in \mathcal{I}(k, \text{card}(\boldsymbol{x})) \text{ s.t. } \boldsymbol{x}(\boldsymbol{i}) = u} \lambda^{\ell(\boldsymbol{i})} \qquad \overset{\text{substring kernel}}{\rightsquigarrow} \qquad K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{u \in \mathcal{A}^k} \phi_u(\boldsymbol{x}) \phi_u(\boldsymbol{y})$$

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 3b: *All-Substrings Kernels*

u	ca	ct	at	ba	bt	cr	ar	br
$\phi_u(\texttt{cat})$	λ^2	λ^3	λ^2	0	0	0	0	0
$\phi_u(\texttt{car})$	λ^2	0	0	0	0	λ^3	λ^2	0
$\phi_u(\texttt{bat})$	0	0	λ^2	λ^2	λ^3	0	0	0
$\phi_u(\texttt{bar})$	0	0	0	λ^2	0	0	λ^2	λ^3

- For example:
  - $K(\texttt{cat}, \texttt{cat}) = K(\texttt{car}, \texttt{car}) = 2\lambda^4 + \lambda^6$
  - $K(\texttt{cat}, \texttt{car}) = \lambda^4$
  - $K(\texttt{cat}, \texttt{bar}) = 0$

Computations

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{u \in \mathcal{A}^k} \phi_u(\boldsymbol{x}) \phi_u(\boldsymbol{y}) = \sum_{u \in \mathcal{A}^k} \sum_{\boldsymbol{i}: \boldsymbol{x}(\boldsymbol{i}) = u} \sum_{\boldsymbol{i}': \boldsymbol{y}(\boldsymbol{i}') = u} \lambda^{\ell(\boldsymbol{i}) + \ell(\boldsymbol{i}')}$$

Enumerating the substrings is of order $O(\text{card}(\boldsymbol{x})^k)$ ⇝ **too slow** ⇝ but we can speed it up (see Ch. 11 of Shawe-Taylor and Cristianini, 2004)

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 4: *Kernel for Text*

- Probably the most important data type after vectors and sequences/strings is **free text**.
- Kernel methods provide a natural framework for pattern analysis in text ⇝ here similarity measures can often be obtained by borrowing methods from different disciplines
- Well-known techniques from *Information Retrieval* (IR), such as the rich class of vector space models, (VSM) can naturally be interpreted as kernel methods.
- An important property of the vector space representation is that the **primal**-vs-**dual** framework we have developed for SVM has an interesting counterpart in the interplay between **word-based**-vs-**document-based** representations.
- Other kernel constructions like *string-matching* (see above) can be applied to free texts, but clearly they are **not** specific to *natural language text*.
- Here we will consider the most basic VSMs only. For mode examples and comments see Ch. 10 of Shawe-Taylor and Cristianini, 2004)

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Example 4: *Kernel for Text* $\rightsquigarrow$ *Vector Space Models* (VSM) $\rightsquigarrow$ *Bag-of-Words* (BoW)

- BoW is the simplest possible version of VSM $\rightsquigarrow$ a *document* is represented by the *words* it contains (and their frequency) with the ordering and punctuation ignored
  - *Words/Terms* $(\texttt{tr})$: sequence of letters from an alphabet $\mathcal{A}$ separated by punctuation or spaces
  - *Document* $(\texttt{doc})$: arbitrary collection of words
  - *Corpus* $(\mathcal{C}_n)$: set of n documents we are processing
  - *Dictionary* $(\mathcal{D}_p)$: a permanent set of p terms or, more often, the set of p words in the *corpus*
- A BoW can be represented as a vector: **each component** is associated with **one term** from the dictionary

$$\phi(\texttt{doc}) = \left[\, \mathrm{tf}(\texttt{tr}_1, \texttt{doc}) \quad \mathrm{tf}(\texttt{tr}_2, \texttt{doc}) \quad \cdots \quad \mathrm{tf}(\texttt{tr}_p, \texttt{doc}) \,\right]^{\mathtt{T}} \in \mathbb{R}^p$$

where $\mathrm{tf}(\texttt{tr}_j, \texttt{doc}) = \{\text{frequency of the } j^{\texttt{th}} \text{ term in the document } \texttt{doc}\} \rightsquigarrow$ **sparse** but *very* high-dimensional embedding!

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 4: *Kernel for Text* $\rightsquigarrow$ *Vector Space Models* (VSM) $\rightsquigarrow$ *Bag-of-Words* (BoW)

- *Document-Term Matrix* (DTM) $\rightsquigarrow$ rows = docs = obs $/$ cols = words = variables $(n \times p)$

$$\mathbb{X} = \begin{bmatrix} \phi(\texttt{doc}_1)^\intercal \\ \phi(\texttt{doc}_1)^\intercal \\ \vdots \\ \phi(\texttt{doc}_n)^\intercal \end{bmatrix} = \begin{bmatrix} \text{tf}(\texttt{tr}_1, \texttt{doc}_1) & \cdots & \text{tf}(\texttt{tr}_p, \texttt{doc}_1) \\ \text{tf}(\texttt{tr}_1, \texttt{doc}_2) & \cdots & \text{tf}(\texttt{tr}_p, \texttt{doc}_2) \\ \vdots & \ddots & \vdots \\ \text{tf}(\texttt{tr}_1, \texttt{doc}_n) & \cdots & \text{tf}(\texttt{tr}_p, \texttt{doc}_n) \end{bmatrix}$$

$$\underset{\substack{\texttt{Term-Doc} \\ \texttt{TDM: } (p \times n)}}{\mathbb{X}^\intercal} \qquad \underset{\substack{\texttt{Term-Term} \\ \texttt{TTM: } (p \times p)}}{\mathbb{X}^\intercal\mathbb{X}} \qquad \underset{\substack{\texttt{Doc-Doc} \\ \texttt{DDM: } (n \times n)}}{\mathbb{X}\mathbb{X}^\intercal} \quad \rightsquigarrow \quad \underset{\substack{\texttt{kernel-matrix} \\ \texttt{(Gramian)}}}{\mathbb{K} = \mathbb{X}\mathbb{X}^\intercal} \quad \rightsquigarrow \quad K(\boldsymbol{x}, \boldsymbol{y}) = \big\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \big\rangle_{\mathbb{R}^p} = \sum_{j=1}^{p} \text{tf}(\texttt{tr}_j, \boldsymbol{x}) \cdot \text{tf}(\texttt{tr}_j, \boldsymbol{y})$$

- **Interpreting Duality**: the *dual* corresponds to a `doc` view, while the *primal* provides a `term` view

  $\texttt{doc} = \{\text{counts of the terms that appear in it}\} \quad \text{vs} \quad \texttt{tr} = \{\text{counts of the docs in which it appears}\}$

- **Remark:** often $n << p \rightsquigarrow$ *dual/docs* computationally better vs *primal/terms* better for interpretation

Toward Nonlinearity...and Beyond!

B) Kernel-trick ⇜⇝ Implicit transformation

Example 4: *Kernel for Text* ⇝ *Vector Space Models* (VSM) ⇝ *Bag-of-Words* (BoW)

Improvements and Limitations

The refinements below can be performed *sequentially* and thought of as a series of *successive embeddings* ⇝ we end up with a single but *composite* feature map that incorporates different aspects of **domain knowledge**.

- *Normalization*: the longer the `doc`, the more words it contains ⇝ the larger $\|\phi(\texttt{doc})\|$ is ⇝ normalize!

$$\widetilde{K}(\boldsymbol{x},\boldsymbol{y}) = \left\langle \frac{\phi(\boldsymbol{x})}{\|\phi(\boldsymbol{x})\|}, \frac{\phi(\boldsymbol{y})}{\|\phi(\boldsymbol{y})\|} \right\rangle_{\mathbb{R}^p} = \frac{K(\boldsymbol{x},\boldsymbol{y})}{\sqrt{K(\boldsymbol{x},\boldsymbol{x})}\cdot\sqrt{K(\boldsymbol{y},\boldsymbol{y})}} \quad \text{(typically the first or the last transformation)}$$

- *Nonlinear Embeddings*: starting from the baseline kernel $K(\boldsymbol{x},\boldsymbol{y})$ corresponding to the (normalised) BoW representation, we can always go **nonlinear** using standard kernel construction. For example

$$\widetilde{K}(\boldsymbol{x},\boldsymbol{y}) = \big(c + K(\boldsymbol{x},\boldsymbol{y})\big)^d \quad \text{(polynomial-kernel over BoW representation)}$$

  uses all k of words for $0 \leqslant k \leqslant d$ as features.

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Example 4: *Kernel for Text* $\rightsquigarrow$ *Vector Space Models* (VSM) $\rightsquigarrow$ *Bag-of-Words* (BoW)

Improvements and Limitations

- *Term weighting*: not all words have the same importance in a task (e.g. guess the *topic* of a doc) $\rightsquigarrow$ *stop-words* are removed before the analysis starts $\rightsquigarrow$ equivalent to assign *weight* 0 to these coordinates in ϕ
  There are *relative* (to a topic or task) or *absolute* measures of importance $\rightsquigarrow$ the latter do **not** use label/response info $\rightsquigarrow$ can be pre-computed on external-larger-unlabeled corpora.
  - $\texttt{idf} = \{\text{inverse doc frequency}\}$ is an *absolute* measure $\rightsquigarrow$ for any word $\texttt{tr} \in \mathcal{D}_p$, let

$$\texttt{df}(\texttt{tr}) = \{\text{number of docs in } \mathcal{C}_n \text{ containing } \texttt{tr}\} \quad \rightsquigarrow \quad \texttt{idf}(\texttt{tr}) = \ln\left(\tfrac{n}{\texttt{df}(\texttt{tr})}\right)$$

$$\mathbb{R} = \begin{bmatrix} \texttt{idf}(\texttt{tr}_1) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \texttt{idf}(\texttt{tr}_p) \end{bmatrix} \rightsquigarrow \widetilde{K}(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})\left[\mathbb{R}\,\mathbb{R}^{\intercal}\right]\phi(\boldsymbol{y})^{\intercal} = \sum_{j=1}^{p} \texttt{idf}(\texttt{tr}_j)^2 \mathrm{tf}(\texttt{tr}_j, \boldsymbol{x})\mathrm{tf}(\texttt{tr}_j, \boldsymbol{y})$$

$$\texttt{(tf-idf representation)}$$

log-scale $\rightsquigarrow$ no weight is too large relative to the weight of a term that occurs in roughly half the docs

Toward Nonlinearity...and Beyond!

B) Kernel-trick $\leftrightsquigarrow$ Implicit transformation

Example 4: *Kernel for Text* $\rightsquigarrow$ *Vector Space Models* (VSM) $\rightsquigarrow$ *Bag-of-Words* (BoW)

Improvements and Limitations

- *Term weighting*: not all words have the same importance in a task (e.g. guess the *topic* of a doc) $\rightsquigarrow$ *stop-words* are removed before the analysis starts $\rightsquigarrow$ equivalent to assign *weight* 0 to these coordinates in ϕ
- *Semantic issues*: **no** semantic relation between words is extraced $\rightsquigarrow$ homonyms/synonyms = problems `tf-idf` down-weights irrelevant terms + highlights potentially discriminative ones $\rightsquigarrow$ **no** semantic!
  - *Stemming*: removing inflections from words ensures that different forms of the same words are treated as equivalent terms $\rightsquigarrow$ basic type of semantic similarity implemented as part of the *tokeniser*.
  - **Solution**: explicitly introduce a **sematic similarity weight** $\rightsquigarrow$ a $(p \times p)$ *proximity matrix* $\mathbb{P}$ such that $\mathbb{P}[i,j] > 0 \Leftrightarrow \mathtt{tr}_i$ is *semantically* related to $\mathtt{tr}_j \rightsquigarrow \widetilde{K}(\boldsymbol{x},\boldsymbol{y}) = \phi(\boldsymbol{x})\left[\mathbb{P}\,\mathbb{P}^{\intercal}\right]\phi(\boldsymbol{y})^{\intercal}$.
  - Many methods are available to design proximity matrices by analysing **co-occurrence** information. For example: *generalized vector space models* (GVSM), latent semantic, semantic diffusion kernels, etc.
  - *Remember*: NN-based embeddings for text mentioned earlier $\rightsquigarrow$ BERT and similia, FastText, etc.

Happy ending...

Summing up

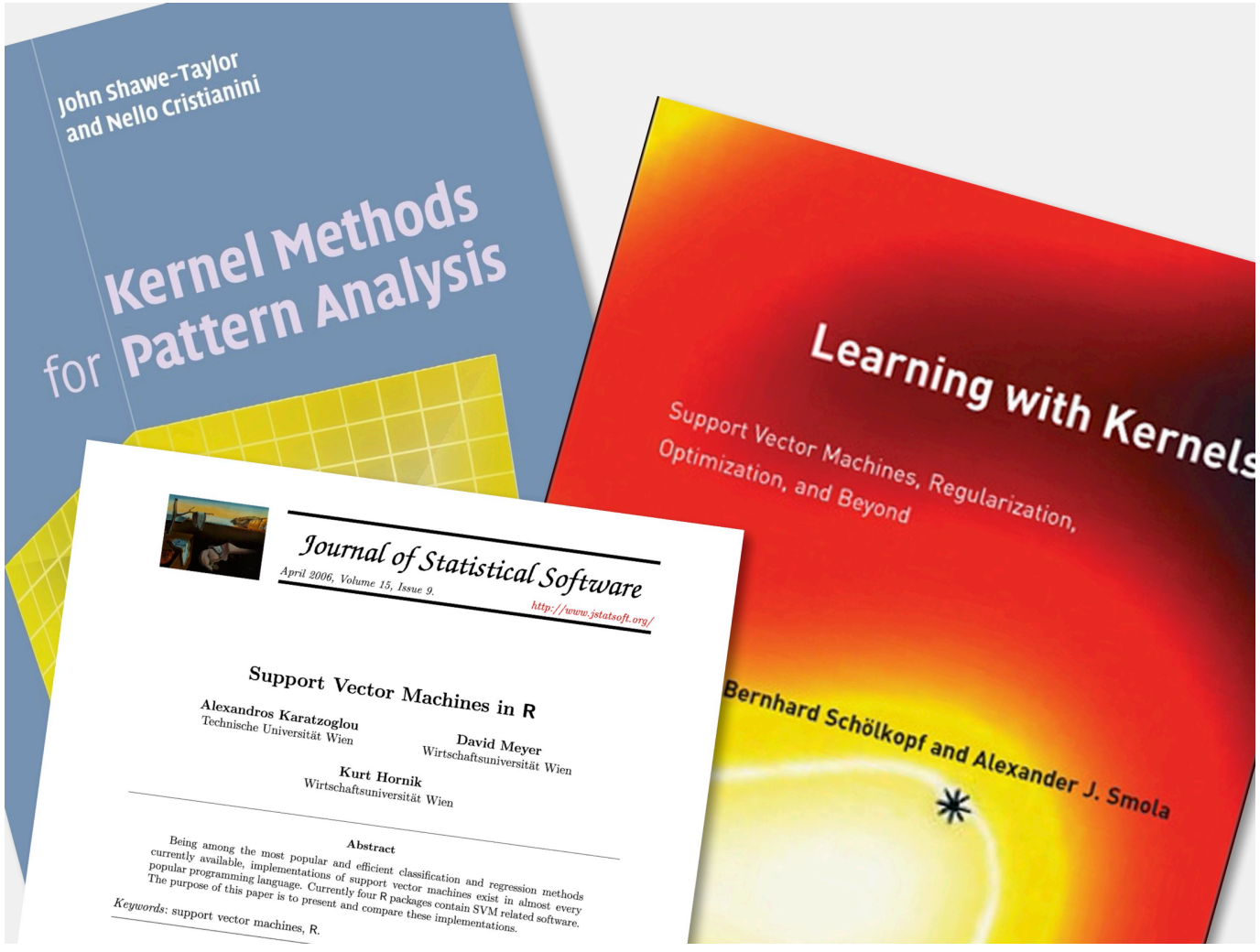
- SVMs can be *really* good classifiers/predictors; from about 1995--2012 they won all the contests and they might come around again...
- Following Vapnik's reductionism, **regularization** and **surrogate loss**, two pillars of modern learning theory, flow naturally from purely *geometric* arguments.
- The concept of margin plays a crucial rule and, typically, classifiers with large margin generalize well
  - Asking for a large margin is an extra constraint that limits the model complexity and *geometrically* code for a L^2-penalty
  - Maximizing the margin is computationally efficient and leads to sparse solutions (support vectors)
  - If we can't get perfect classification we allow ourselves some **slack**
- Nonlinearity can be nicely and efficiently brought into the picture by exploiting the *kernel-trick*.
  - Learning task where the input space $\mathcal{X}$ is **not** a vector space can also be treated effortlessly and in a unified manner.

Going forward

...broaden our understanding of (Mercer) kernel methods in the framework of RKHS...

Reproducing Kernel Hilbert Spaces

References


John Shawe-Taylor
and Nello Cristianini
Kernel Methods
for Pattern Analysis
Learning with Kernels
Support Vector Machines, Regularization,
Optimization, and Beyond
Bernhard Schölkopf and Alexander J. Smola
Journal of Statistical Software
April 2006, Volume 15, Issue 9.
http://www.jstatsoft.org/
Support Vector Machines in R
Alexandros Karatzoglou
Technische Universität Wien
David Meyer
Wirtschaftsuniversität Wien
Kurt Hornik
Wirtschaftsuniversität Wien
Abstract
Being among the most popular and efficient classification and regression methods
currently available, implementations of support vector machines exist in almost every
popular programming language. Currently four R packages contain SVM related software.
The purpose of this paper is to present and compare these implementations.
Keywords: support vector machines, R.