EECE-6822 Machine Learning

Non-parametric methods, k Nearest Neighbors, SVM, Kernel methods

Cris Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

1

PART 1
kNN, Kernels

2

Nonparametric models
k Nearest Neighbours (kNN)

Parametric vs non-parametric

- A model is **parametric** if **number of parameters** does not depend on **number of samples**

So far in this book, we have mostly focused on **parametric models**, either unconditional $p(\boldsymbol{y}|\boldsymbol{\theta})$ or conditional $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a fixed-dimensional vector of parameters. The parameters are estimated from a variable-sized dataset, $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$, but after model fitting, the data is thrown away.

- A model is **non-parametric** (does not mean absence of parameters!) if **number of parameters** increases with **number of samples**

In this section we consider various kinds of **nonparametric models**, that keep the training data around. Thus the effective number of parameters of the model can grow with $|\mathcal{D}|$. We focus on models that can be defined in terms of the **similarity** between a test input, $\boldsymbol{x}$, and each of the training inputs, $\boldsymbol{x}_n$. Alternatively, we can define the models in terms of a dissimilarity or distance function $d(\boldsymbol{x}, \boldsymbol{x}_n)$. Since the models keep the training examples around at test time, we call them **exemplar-based models**. (This approach is also called **instance-based learning** [AKA91], or **memory-based learning**.)

[***B2-Murphy***] Kevin Murphy, Probabilistic Machine Learning: An Introduction, 2022. (Free PDF online).

K Nearest Neighbors (KNN)

- Assume we have a classification task
- To classify a new point ***x***:
  - Find its ***k*** nearest neighbors in the training data
  - Set ***y*** to be the **majority vote** of the labels of these nearest neighbors
- Design choices / hyperparameters:
  - Number of nearest neighbors
  - **Distance metric**
  - Aggregation method

5

5

K nearest neighbor (KNN) classification

In this section, we discuss one of the simplest kind of classifier, known as the **K nearest neighbor** (**KNN**) classifier. The idea is as follows: to classify a new input x, we find the K closest examples to x in the training set, denoted $N_K(x, \mathcal{D})$, and then look at their labels, to derive a distribution over the outputs for the local region around x. More precisely, we compute

$$p(y = c|x, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(x, \mathcal{D})} \mathbb{I}(y_n = c) \tag{16.1}$$

We can then return this distribution, or the majority label.

The two main parameters in the model are the size of the neighborhood, K, and the distance metric $d(x, x')$. For the latter, it is common to use the **Mahalanobis distance**

$$d_{\mathbf{M}}(x, \mu) = \sqrt{(x - \mu)^{\mathsf{T}} \mathbf{M} (x - \mu)} \tag{16.2}$$

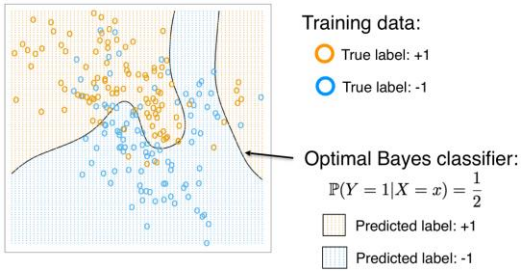
where $\mathbf{M}$ is a positive definite matrix. If $\mathbf{M} = \mathbf{I}$, this reduces to Euclidean distance. We discuss how to learn the distance metric in Section 16.2.

6

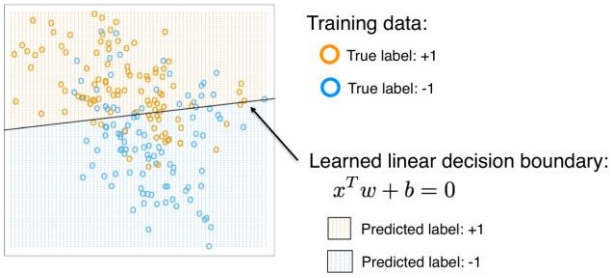
[***B2-Murphy***] Kevin Murphy, Probabilistic Machine Learning: An Introduction, 2022. (Free PDF online).

6

Example: Bayes classifier
(See Appendix A)



Linear decision boundary

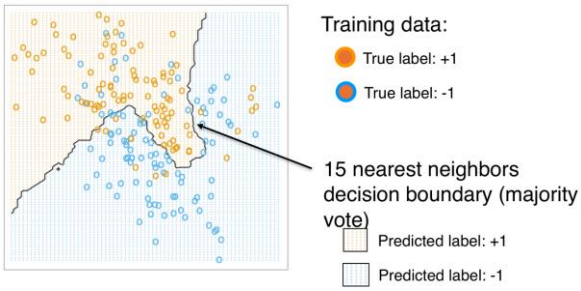


[**B2-Hastie**] Trevor Hastie, The Elements of Statistical Learning, 2017. (ESL, Free PDF online).

7

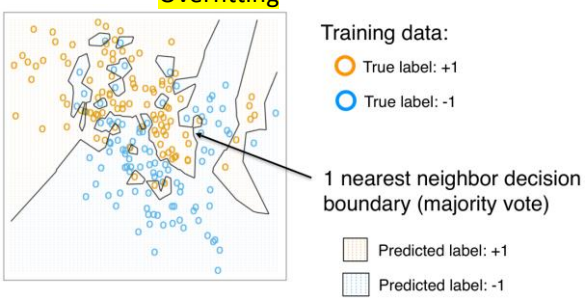
7

k=15 nearest neighbors boundary



k=1 nearest neighbors boundary

Overfitting

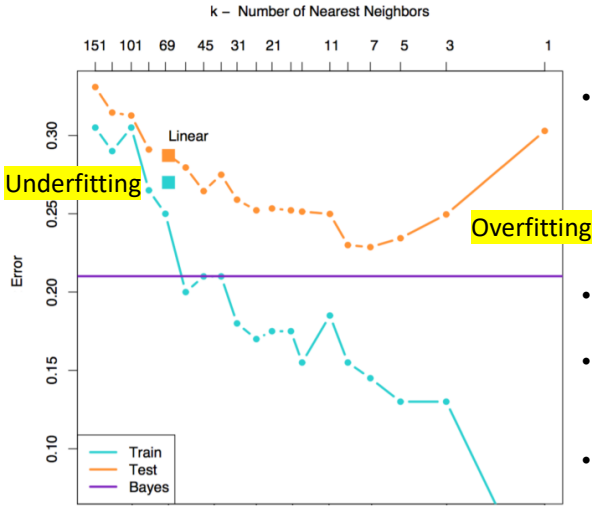


[**B2-Hastie**] Trevor Hastie, The Elements of Statistical Learning, 2017. (ESL, Free PDF online).

8

8

k nearest neighbors error



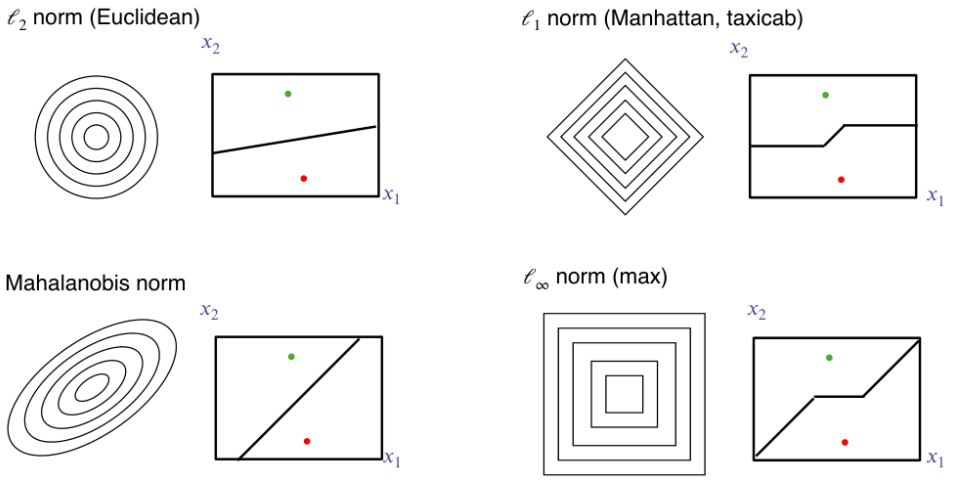
- A single **training sample** of size 200 was used, and a **test sample** of size 10,000.
  - "Training" means initial data samples that are stored in the memory and used for predictions. Does not have the same meaning as in "optimization of parameters θ"
- The orange curves are test and the blue are training error for k-nearest-neighbor classification.
- The results for linear regression are the bigger orange and blue squares at three degrees of freedom.
- The purple line is the optimal Bayes error rate.

[**B2-Hastie**] Trevor Hastie, The Elements of Statistical Learning, 2017. (ESL, Free PDF online).

9

9

Notable distance metrics and Level sets

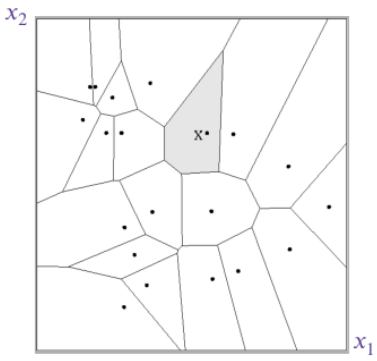


10

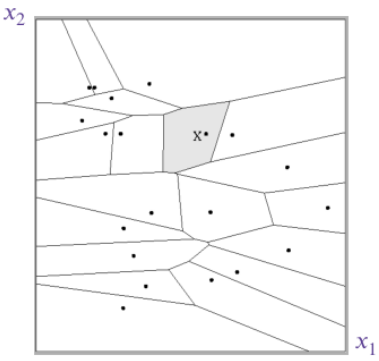
10

Example: distance metrics with *k=1* NN

$d(x, x') = (x_1 - x'_1)^2 + (x_2 - x'_2)^2$



$d(x, x') = (x_1 - x'_1)^2 + 9(x_2 - x'_2)^2$



- Voronoi diagram - partition of a plane into regions - each region consists of all points closer to one specific point.
- Constructed by drawing lines **equidistant** from pairs of points; forming convex polygons (or cells) around each point.

11

11

1-NN classification: Theoretical guarantees

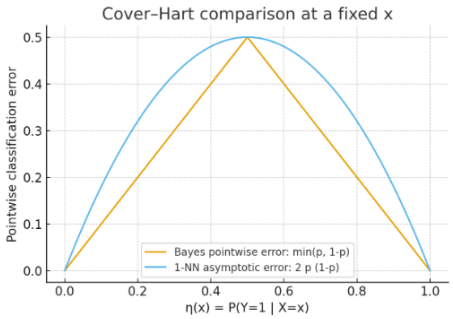
Theorem[Cover, Hart, 1967] If P_X is supported everywhere in $\mathbb{R}^d$ and $P(Y = 1|X = x)$ is smooth everywhere, then as $n \to \infty$ the 1-NN classification rule has error at most twice the Bayes error rate.

Let R^* be the **Bayes risk** (minimum possible 0–1 error) and let $R_{1\mathrm{NN}}$ be the **asymptotic risk** of the 1-nearest-neighbor classifier as the training sample size $n \to \infty$. Then, for binary classification,

$$R_{1\mathrm{NN}} \le 2R^*(1 - R^*) \le 2R^*$$

(The second inequality uses $R^* \le \frac{1}{2}$.)

Interpretation. With infinitely much data, 1-NN makes at most **twice the Bayes error** (and in fact the sharper bound above holds). So 1-NN is *universally competitive up to a factor of 2* in the large-sample limit.



12

12

The curse of dimensionality

"the silent killer of all local, distance-based methods"

- When data lives in a high-dimensional space makes algorithms - that rely on local neighborhoods, distances, or density estimation - behave poorly as dimension d increases.
- For kNN (predicts based on local neighborhood around a query point), curse manifests as:
  - Neighborhoods become sparse — almost every point is far away.
  - Distances become uninformative — all points look roughly equally distant.
  - Local averaging becomes high-variance and requires exponentially more data.

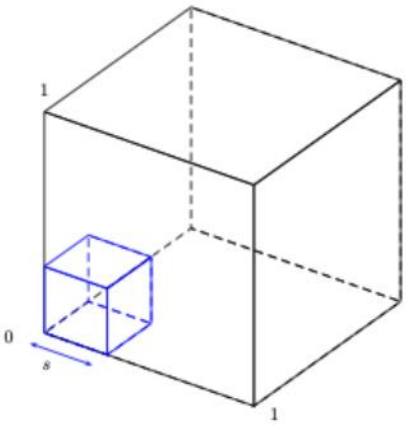
The curse of dimensionality

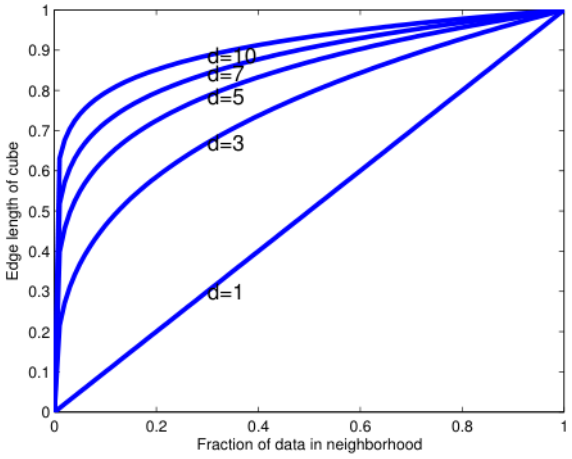
The main statistical problem with KNN classifiers is that they do not work well with high dimensional inputs, due to the **curse of dimensionality**.

The basic problem is that the volume of space grows exponentially fast with dimension, so you might have to look quite far away in space to find your nearest neighbor. To make this more precise, consider this example from [HTF09, p22]. Suppose we apply a KNN classifier to data where the inputs are uniformly distributed in the D-dimensional unit cube. Suppose we estimate the density of class labels around a test point x by "growing" a hyper-cube around x until it contains a desired fraction p of the data points. The expected edge length of this cube will be $e_p(s) \triangleq s^{1/D}$. this function is plotted in Figure 16.3(b). If $D = 10$, and we want to base our estimate on 10% of the data, we have $e_{10}(0.1) = 0.8$, so we need to extend the cube 80% along each dimension around x. Even if we only use 1% of the data, we find $e_{10}(0.01) = 0.63$. Since the range of the data is only 0 to 1 along each dimension, we see that the method is no longer very local, despite the name "nearest neighbor". The trouble with looking at neighbors that are so far away is that they may not be good predictors about the behavior of the function at a given point.

[***B2-Murphy***] Kevin Murphy, Probabilistic Machine Learning: An Introduction, 2022. (Free PDF online).

Curse of dimensionality



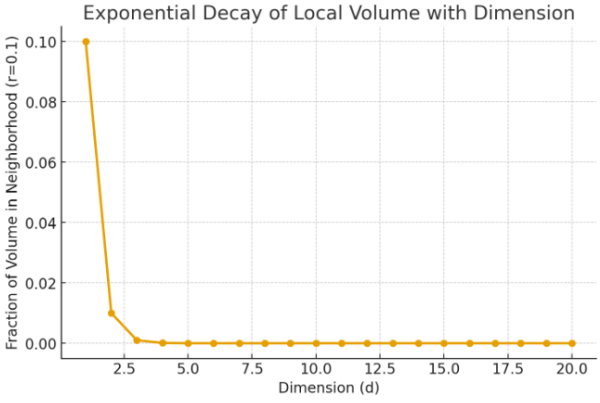


X is uniformly distributed in $[0,1]^d$. What is $P(X \in [0,1]^d)$?

How many samples do we need so that a nearest neighbor is within a cube of side-length ***s***? 15

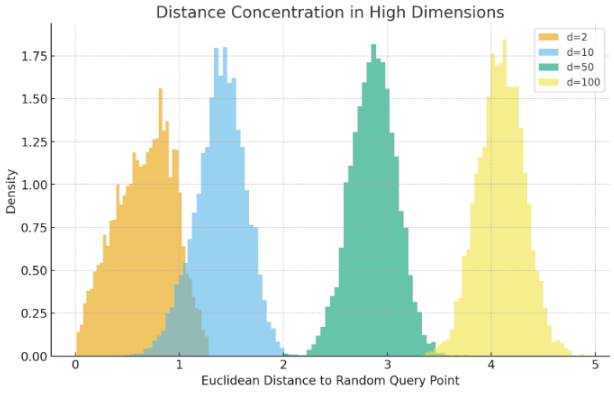
15

Curse of dimensionality



Exponential Decay of Local Volume with Dimension

By $d = 5$, fraction of nearby space is already essentially zero - meaning "local region" becomes almost empty



Distance Concentration in High Dimensions

As d grows:

All distances cluster tightly around a mean value.

Difference between **nearest** and **farthest** neighbors becomes negligible.

So, "near" and "far" lose meaning = destroying the foundation of kNN.

16

16

Mitigation Strategies – How to Deal with the Curse

- Feature Selection / Dimensionality Reduction
  - PCA, autoencoders, manifold learning - reduce effective dimensionality.
- Distance Metric Learning
  - Learn a Mahalanobis or cosine-based metric to focus on informative subspaces.
- Kernel and Smoothing Methods
  - Replace discrete neighbors with smooth Kernels

17

17

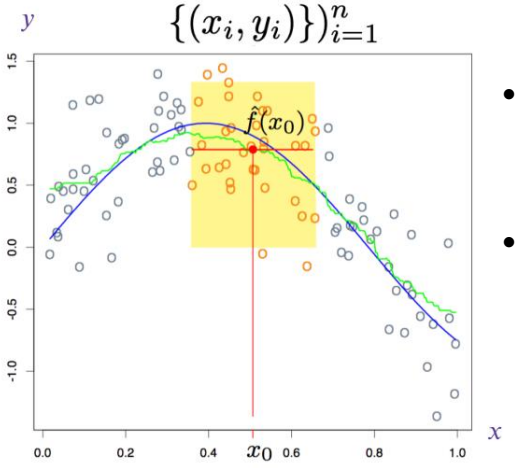
KNN Regression

- KNN regression is a **non-parametric** method used for predicting continuous values.
- The core idea is to predict the target value for a new data point by averaging the target values of the K nearest neighbors in the feature space.
- The distance between data points is typically measured using Euclidean distance, although other distance metrics can be used.

18

18

Nearest-neighbor Regression



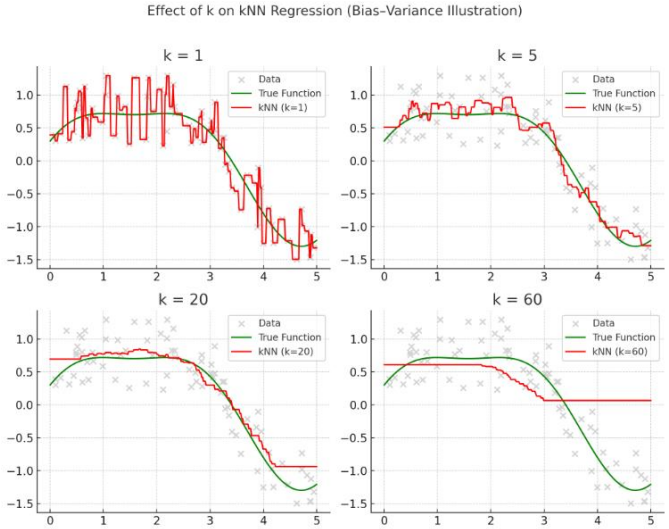
$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x_0)} y_i$$

- **Small k:** prediction uses few neighbors, fits local noise:
  - low bias, high variance
- **Large k:** averages over many distant points, smooths too much:
  - *high bias, low variance*

19

19

Bias-Variance Tradeoff



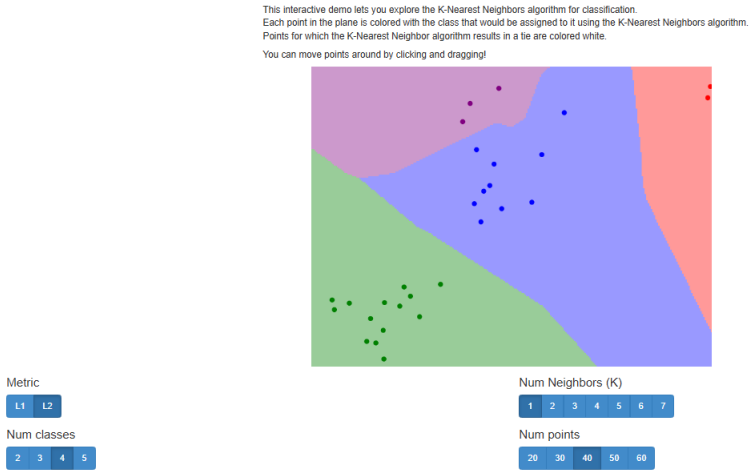
k=1: memorizes training data → low bias, high variance.
k→n: averages the entire dataset → high bias, low variance.
Somewhere in between lies the sweet spot that minimizes test error.

20

20

K-Nearest Neighbors Demo

- http://vision.stanford.edu/teaching/cs231n-demos/knn/

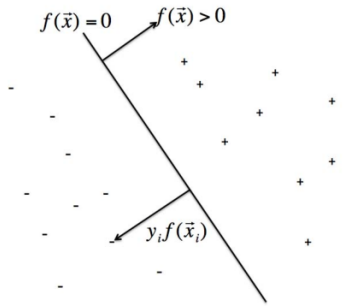


Support Vector Machines (SVM)

Main Idea

- An **SVM** is a supervised learning model for **binary classification** that finds the **maximum-margin hyperplane** separating two classes.
- It aims to find the decision boundary that not only separates the classes but does so with the **largest possible margin** (distance to the nearest data points).

The margin of an example $\mathbf{x}_i$ = "distance" from example to decision boundary
$= y_i f(\mathbf{x}_i)$



23

23

Linear Separation

Suppose we have data:

$$\{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\}.$$

A **linear classifier** predicts:

$$f(x) = \text{sign}(w^\top x + b)$$

where w defines the orientation of the hyperplane, and b shifts it.

Margin Definition

The **functional margin** of point x_i is:

$$\gamma_i = y_i(w^\top x_i + b)$$

The **geometric margin** (actual perpendicular distance to the decision boundary) is:

$$\hat{\gamma}_i = \frac{y_i(w^\top x_i + b)}{\|w\|}$$

24

24

Maximum Margin Formulation

We want to find the hyperplane that **maximizes the minimum margin**:

$$\max_{w,b} \min_{i} \frac{y_i(w^\top x_i + b)}{\|w\|}$$

This can be rewritten as a **convex optimization problem**:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \quad \forall i \end{aligned}$$

This is the **hard-margin SVM**.

"Street"

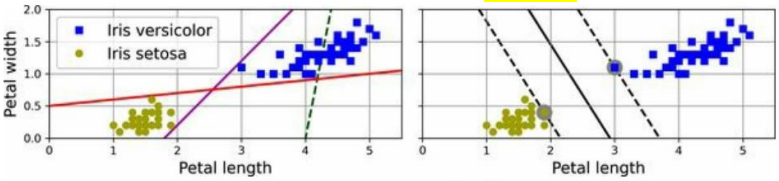
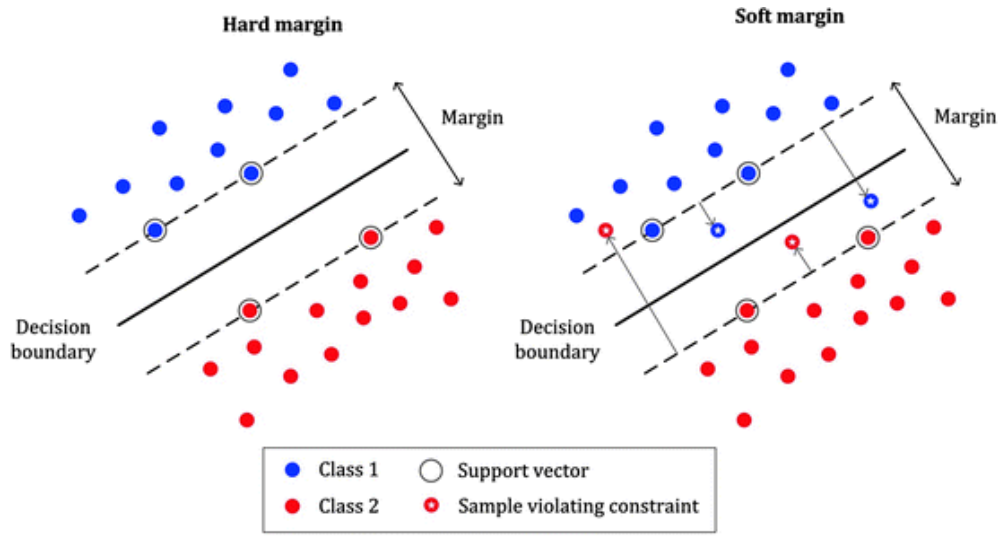


Figure 5-1. Large margin classification

25

25



26

26

Soft Margin SVM (Non-Separable Data)

For noisy or overlapping data, perfect separation isn't possible.

Introduce **slack variables** $\xi_i \geq 0$ to allow margin violations.

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

- C controls the tradeoff between **margin width** and **classification error**:
  - Large C: smaller margin, fewer misclassifications (less regularization)
  - Small C: larger margin, more tolerance for errors (more regularization)

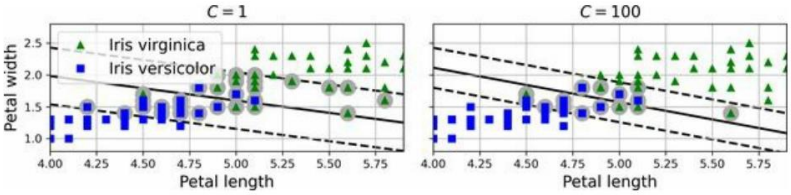


Figure 5-4. Large margin (left) versus fewer margin violations (right)

27

27

The hard margin and soft margin problems are both convex quadratic optimization problems with linear constraints. Such problems are known as *quadratic programming* (QP) problems. Many off-the-shelf solvers are available to solve QP problems by using a variety of techniques that are outside the scope of this book. [4]

28

[***B3-Geron**] Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2022.

28

The Dual Problem

Given a constrained optimization problem, known as the *primal problem*, it is possible to express a different but closely related problem, called its *dual problem*. The solution to the dual problem typically gives a lower bound to the solution of the primal problem, but under some conditions it can have the same solution as the primal problem. Luckily, the SVM problem happens to meet these conditions, [5] so you can choose to solve the primal problem or the dual problem; both will have the same solution. Equation 5-3 shows the dual form of the linear SVM objective. If you are interested in knowing how to derive the dual problem from the primal problem, see the extra material section in this chapter's notebook.

[***B3-Geron**] Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2022.

29

29

Dual Formulation (See **Appendix B** for details)

Using Lagrange multipliers α_i, we get the **dual problem**:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i^\top x_j)$$

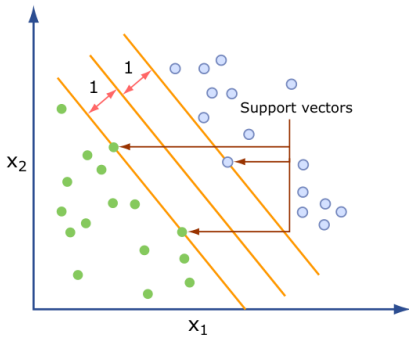
$$\text{s.t. } 0 \le \alpha_i \le C, \quad \sum_i \alpha_i y_i = 0$$

The optimal weight vector is:

$$w = \sum_i \alpha_i y_i x_i$$

Only **support vectors** have nonzero α_i.

- Dual problem is faster to solve than the primal one when the number of training instances is smaller than the number of features.
- Dual problem makes the **kernel trick** possible (primal problem does not!)



30

The Kernel Trick

In the dual, data points appear **only as inner products** $(x_i^\top x_j)$.

Replace them with a **kernel function**:

$$K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

→ allowing us to work in high-dimensional feature spaces implicitly.

Then the classifier becomes:

$$f(x) = \text{sign}\left(\sum_i \alpha_i y_i K(x_i, x) + b\right)$$

Heads-up summary of Kernel Trick:

- Feature map dot products are substituted by inner products of train data points.
- These products have same expression of what we define as **kernel function**.
- Kernel function is computed very fast.
- See later section on **Kernel Methods** for more

31

31

Table 5-1. Comparison of Scikit-Learn classes for SVM classification

Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
LinearSVC	$O(m \times n)$	No	Yes	No
SVC	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes
SGDClassifier	$O(m \times n)$	Yes	Yes	No

[***B3-Geron**] Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2022.

32

32

SciKit-Learn Classification

The following Scikit-Learn code loads the iris dataset and trains a linear SVM classifier to detect *Iris virginica* flowers. The pipeline first scales the features, then uses a LinearSVC with C=1:

```
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2)  # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
                        LinearSVC(C=1, random_state=42))
svm_clf.fit(X, y)
```

33

33

SciKit-Learn Classification

Fortunately, when using SVMs you can apply an almost miraculous mathematical technique called the *kernel trick* (which is explained later in this chapter). The kernel trick makes it possible to get the same result as if you had added many polynomial features, even with a very high degree, without actually having to add them. This means there's no combinatorial explosion of the number of features. This trick is implemented by the SVC class. Let's test it on the moons dataset:

```
from sklearn.svm import SVC

poly_kernel_svm_clf = make_pipeline(StandardScaler(),
                                    SVC(kernel="poly", degree=3, coef0=1, C=5))
poly_kernel_svm_clf.fit(X, y)
```

without actually doing so. Let's try the SVC class with the Gaussian RBF kernel:

```
rbf_kernel_svm_clf = make_pipeline(StandardScaler(),
                                   SVC(kernel="rbf", gamma=5, C=0.001))
rbf_kernel_svm_clf.fit(X, y)
```

34

34

SciKit-Learn Regression

You can use Scikit-Learn's LinearSVR class to perform linear SVM regression. The following code produces the model represented on the left in Figure 5-10:

```
from sklearn.svm import LinearSVR

X, y = [...]  # a linear dataset
svm_reg = make_pipeline(StandardScaler(),
                        LinearSVR(epsilon=0.5, random_state=42))
svm_reg.fit(X, y)
```

The following code uses Scikit-Learn's SVR class (which supports the kernel trick) to produce the model represented on the left in Figure 5-11:

```
from sklearn.svm import SVR

X, y = [...]  # a quadratic dataset
svm_poly_reg = make_pipeline(StandardScaler(),
                             SVR(kernel="poly", degree=2, C=0.01, epsilon=0.1))
svm_poly_reg.fit(X, y)
```

35

35

Kernel Methods Bootstrap

36

36

Main Idea

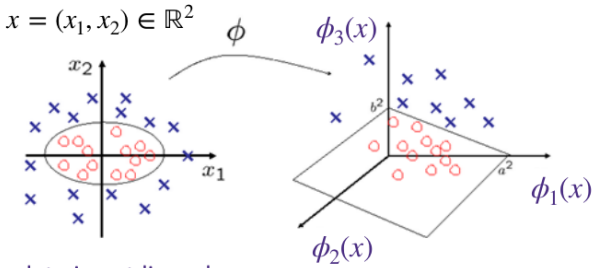
- Initial (train or input) dataset has samples (datapoints, d-dimensional) that are not linearly separable
- Idea: Use Feature Maps ϕ(x) to transform input d-dimensional space into p>>d dimensional feature space where (hopefully) transformed samples will be linearly separable.
- But issue: in transformed p-dimensional space, one must compute ϕ(x) themselves and operations like dot-product on them (as part of optimization algorithm such as SVM, or GD) – this is computationally and memory expensive!
- Solution to this issue: Kernel Trick
  - For some Feature Maps, dot-products of ϕ(x) have the same expression as dot(inner)-products of train samples themselves!
  - Now, call/define that expression as Kernel Function – which is much easier and faster to compute, when done from working directly with products train samples themselves perspective!
  - This observation, allows us to "replace" the "calculation of ϕ(x) and their products" with "calculations of input sample products" – eliminating the need to even know ϕ(x)! (we only know that, that substitution is valid)

37

37

What if the data is not linearly separable?

- Use features, for example,



This data is not linearly separable

Can you suggest some features $\phi_1(x_1, x_2), \phi_2(x_1, x_2), \phi_3(x_1, x_2)$ such that this data is linearly separable in this 3-dimensional space?

- **High dimensional feature spaces make it easier to linearly separate different classes**
- However, hard to know which **feature map** will work for given data

38

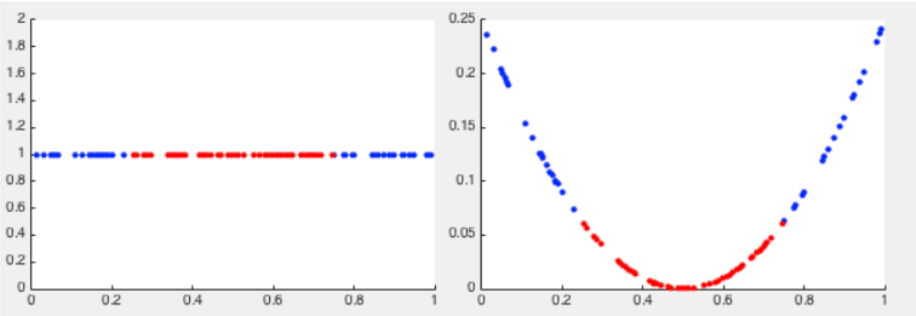
38

Example 1

In 1-D space, classes $C_- = \{x \big| (a \leq x \leq b)\}$ and $C_+ = \{x \big| (x \leq a) \text{ or } (x \geq b)\}$ are not linearly separable. By the following mappping from 1-D space to 2-D space:

$$\mathbf{z} = \phi(x) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} x \\ (x - (a+b)/2)^2 \end{bmatrix}$$

the two classes can be separated by a threshold in the second dimension of the 2-D space.



https://pages.hmc.edu/ruye/MachineLearning/lectures/ch8/node10.html

39

39

Example 2:

The method above can be generalized to higher dimensional spaces such as mapping from 2-D to 3-D space. Consider two classes in a 2-D space that are not linearly separable: $C_- = \{\mathbf{x},\ ||\mathbf{x}|| < D\}$ and $C_+ = \{\mathbf{x},\ ||\mathbf{x}|| > D\}$. However, by the following mappping from 2-D space to 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix} \tag{92}$$

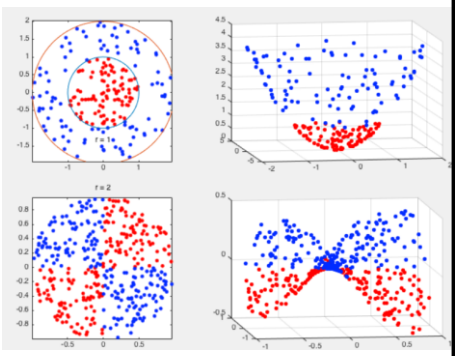
the two classes can be trivially separated linearly by thresholding in the third dimension of the 3-D space.

Example 3:

In 2-D space, in the exclusive OR dataset, the two classes of C_- containing points in quadrants I and III, and C_+ containing points in quadrants II and IV are not linearly separable. However, by mapping the data points to a 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \tag{93}$$

the two classes can be separated by simply thresholding in the third dimension of the 3-D space.



https://pages.hmc.edu/ruye/MachineLearning/lectures/ch8/node10.html

40

40

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1, one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

41

Creating Features

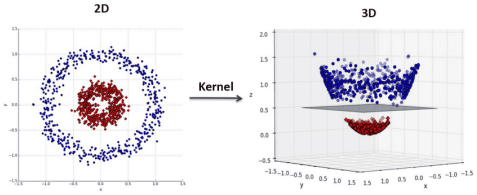
- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with p where $d < n \ll p$

42

42

Problem: High-Dimensional ϕ(x) is Expensive

- Explicitly computing $\phi(x)$ for large p is often impossible (e.g., infinite for Gaussian kernels).
- Luckily, most **learning algorithms** (SVM, ridge regression, etc.) only use inner products of feature vectors: $\langle \phi(x), \phi(x') \rangle$
- So, if we can compute these inner products directly without ever computing $\phi(x)$, we can operate in high-dimensional spaces **implicitly**.
- **Kernel Trick**: define a **Kernel Function**: $K(x, x') = \langle \phi(x), \phi(x') \rangle$
- Then, we can replace all inner products in our **learning algorithm** by $K(x, x')$.



43

The Kernel Trick

The kernel trick is that if you have an algorithm (like SVM) where the examples appear only in inner products, you can freely replace the inner product with a different one. (And it works just as well as if you designed the SVM with some map $\mathbf{\Phi}(\mathbf{x})$ in mind in the first place, and took the inner product in $\mathbf{\Phi}$'s space instead.)

Remember the optimization problem for SVM?

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,k=1}^{m} \alpha_i \alpha_j y_i y_k \, \mathbf{x}_i^T \mathbf{x}_k \leftarrow \text{inner product}$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, i = 1, ..., m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

You can replace this inner product with another one, *without even knowing* $\mathbf{\Phi}$. In fact there can be many different feature maps that correspond to the same inner product.

In other words, we'll replace $\mathbf{x}^T\mathbf{z}$ (i.e., $\langle \mathbf{x}, \mathbf{z} \rangle_{\mathbf{R}^n}$) with $k(\mathbf{x}_i, \mathbf{x}_j)$, where k happens to be an inner product in some feature space, $\langle \mathbf{\Phi}(\mathbf{x}), \mathbf{\Phi}(\mathbf{x}) \rangle_{\mathcal{H}_k}$. Note that k is also called the kernel (you'll see why later).

44

https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/resources/mit15_097s12_lec13/

How do we deal with high-dimensional data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a ***kernel*** for a map $\phi(\cdot)$
if $K(x, x') \triangleq \phi(x)^T \phi(x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle$

This notation is for dot product (which is the same as inner product)

Main idea: computing inner products can be much more efficient than explicitly computing the (very high-dimensional) features

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)
  and if we can find a **kernel** for our feature map such that

$$K(x, x') = \phi(x)^T \phi(x')$$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^n$
and instead only work with the kernel matrix of the training data
$\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

45

45

Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$
  takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn
  - The features are **implicit** and accessed only via kernels, making it efficient

46

46

Examples of popular Kernels

- Polynomials of degree exactly k

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

- Gaussian (squared exponential) kernel (a.k.a RBF kernel for Radial Basis Function)

$$K(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2} \right)$$

- Sigmoid

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

- All these kernels are efficient to compute, but the corresponding features are in high-dimensions

47

47

Matrix K is symmetric, positive-semidefinite (positive eigenvalues)

Say we have a finite input space $\{x_1, ..., x_m\}$. So there's only m possible states for the x_i's. (Think of the bag-of-words example where there are $2^{(\#\text{words})}$ possible states.) I want to be able to take inner products between any two of them using my function k as the inner product. Inner products by definition are symmetric, so $k(x_i, x_j) = k(x_j, x_i)$. In other words, in order for us to even consider k as a valid kernel function, the matrix:

$$\mathrm{K} = \begin{array}{c} \\ m \\ i \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & j & m \end{array} \\ \begin{bmatrix} & & \\ & k(x_i, x_j) & \\ & & \end{bmatrix} \end{array}$$

needs to be symmetric, and this means we can diagonalize it, and the eigendecomposition takes this form:

$$\mathbf{K} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}'$$

where $\mathbf{V}$ is an orthogonal matrix where the columns of $\mathbf{V}$ are eigenvectors, $\mathbf{v_t}$, and $\mathbf{\Lambda}$ is a diagonal matrix with eigenvalues λ_t on the diagonal.

48

https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/resources/mit15_097s12_lec13/

48

MERCER'S THEOREM

According to *Mercer's theorem*, if a function $K(\mathbf{a}, \mathbf{b})$ respects a few mathematical conditions called *Mercer's conditions* (e.g., K must be continuous and symmetric in its arguments so that $K(\mathbf{a}, \mathbf{b}) = K(\mathbf{b}, \mathbf{a})$, etc.), then there exists a function ϕ that maps **a** and **b** into another space (possibly with much higher dimensions) such that $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^\top \phi(\mathbf{b})$. You can use K as a kernel because you know ϕ exists, even if you don't know what ϕ is. In the case of the Gaussian RBF kernel, it can be shown that ϕ maps each training instance to an infinite-dimensional space, so it's a good thing you don't need to actually perform the mapping!

Note that some frequently used kernels (such as the sigmoid kernel) don't respect all of Mercer's conditions, yet they generally work well in practice.

[***B3-Geron**] Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2022.

49

High-level Example: **Features vs. Kernels**

- Ridge regression with feature map $\phi(\cdot) \in \mathbb{R}^p$
  - Solve for $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - w^T \phi(x_i)\right)^2 + \lambda \frac{1}{2} \|w\|_2^2$
  - Slow when $p \gg d$
- For now, suppose we are solving this using gradient descent with
  - $w_0 = 0$ and
  - $w_{t+1} \leftarrow w_t + \eta \sum_{i=1}^{n} \phi(x_i)\left(y_i - w_t^T \phi(x_i)\right) - \eta \lambda w_t$
- Claim: For any t, w_t can be represented as $w_t = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ for some n-dimensional parameter $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Prediction $\hat{y}_{\text{new}} = \hat{w}^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x_{\text{new}})$

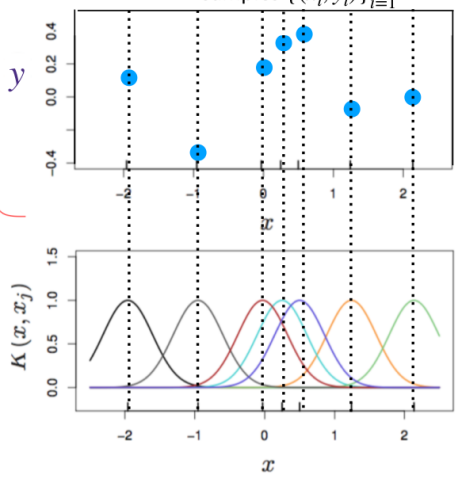
See **APPENDIX C**

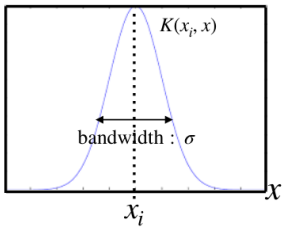
- One crucial observation is that all $w^{(t)}$'s (including the optimal solution $w^{(\infty)}$) lie on the subspace spanned by $\{\phi(x_1), \ldots, \phi(x_n)\}$, which is an n-dimensional subspace in $\mathbb{R}^p$
- Hence, it is sufficient to look for a solution that is represented as $\widehat{w} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ to find the optimal solution

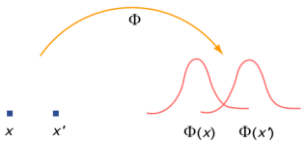
Prediction requires inner product in **Feature** space!
Can be replaced by **Kernel**, which makes computation feasible!

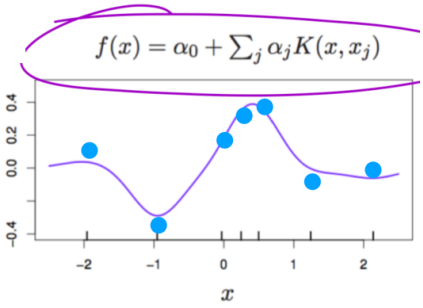
50

Example: RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$





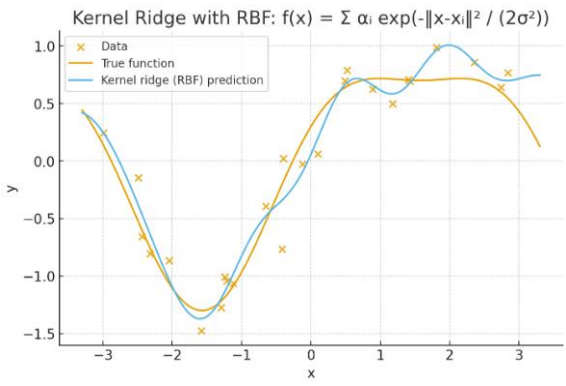


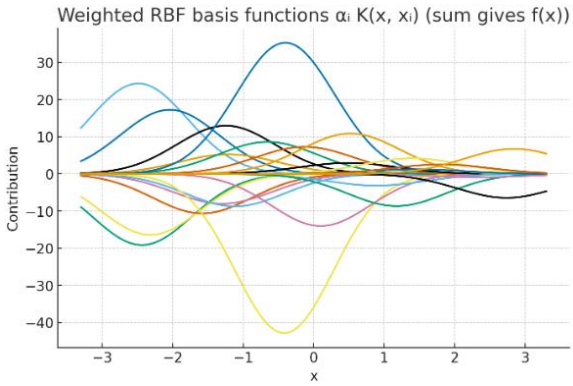


See **APPENDIX D** for more details on RBF

- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

51





52

52

How do you choose a Kernel?

- Choosing a **kernel function** is one of the most important design decisions in kernel methods (e.g., SVMs, Gaussian processes, Kernel Ridge Regression, etc.), because it defines the **geometry of similarity** in your feature space — effectively the model's "bias" about what patterns are learnable.

Common kernel families and their intuition

Kernel	Formula	Typical use	Intuition about data
Linear	$K(x, x') = x^\top x'$	Linear relations	Works well when data are linearly separable or features already engineered
Polynomial	$K(x, x') = (x^\top x' + c)^d$	Moderate nonlinearities	Captures feature interactions up to degree d; increasing d increases complexity
RBF (Gaussian)	$K(x, x') = \exp(-\|x - x'\|^2 / 2\sigma^2)$	Default for many tasks	Smooth local similarity; infinitely differentiable; decision boundary is smooth and flexible
Laplacian	$K(x, x') = \exp(-\|x - x'\| / \sigma)$	Noisy data	Similar to RBF but less smooth; more robust to outliers
Sigmoid (tanh)	$K(x, x') = \tanh(\alpha x^\top x' + c)$	Historical (MLP)	Acts like neural-net activation; less used due to tricky parameter ranges
Periodic	$K(x, x') = \exp\big(-2\sin^2(\pi\|x - x'\|/p)/\ell^2\big)$	Time-series with seasonality	Captures repeating patterns
Rational Quadratic	$K(x, x') = (1 + \frac{\|x - x'\|^2}{2\alpha\ell^2})^{-\alpha}$	Multi-scale smoothness	Mixture of RBFs with different scales

Heuristics for choosing the kernel

(a) Nature of your data

Data type	Good starting kernels
Vector, continuous	RBF (default), Linear
Sparse high-dimensional (text, bag-of-words)	Linear or polynomial (degree 2)
Structured (strings, trees, graphs)	Custom kernels: string kernel, subtree kernel, graph diffusion kernel
Temporal / periodic	RBF × periodic, or spectral mixture kernels
Spatial / image patches	RBF on spatial features, or convolutional kernel
Categorical (few categories)	δ-kernel (Kronecker delta) or embedding + RBF

55

55

Heuristics for choosing the kernel

(b) Complexity & smoothness

- If you expect a **smooth** function → use **RBF or polynomial**.
- If you expect **abrupt changes** or non-smoothness → **Laplacian or rational quadratic**.
- If the relationship is **linear or nearly linear** → **Linear kernel** suffices (and much faster).

(c) Dimensionality and sample size

- In **high-dimensional, small-n** problems, RBF may overfit (too many degrees of freedom). Try **linear** or **polynomial (low degree)**.
- In **low-dimensional, large-n**, RBF works beautifully.

(d) Cross-validation & model selection

Ultimately, kernel choice and parameters (σ, d, c, α) are **hyperparameters** — choose them by **grid search** or **Bayesian optimization** over CV performance.

56

56

Fixed Feature vs. Learned Feature

- Kernel method works well if we choose a good kernel such that the data is linearly separable in the corresponding (possibly infinite dimensional) feature space
- In practice, it is hard to choose a good kernel for a given problem
- Can we **learn** the feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ from data also?

Deep Learning? CNNs, DNNs can "generate" features automatically for us.

57

57

Bootstrap

Measuring uncertainty for your model

58

58

What the bootstrap does (intuition)

You have one dataset $D = \{z_i\}_{i=1}^n$ drawn from an unknown population P. You fit a model and compute a statistic $\hat{\theta} = s(D)$ (e.g., a coefficient, AUC, test MSE).
You'd like to know **how variable** $\hat{\theta}$ would be if you could repeatedly resample new datasets from P. You can't—so you **approximate** P **with the empirical distribution** $\hat{P}$ that puts mass $1/n$ on each observed z_i.
Then you **simulate "new datasets"** by sampling **with replacement** from D.

59

59

Core nonparametric bootstrap algorithm

For $b = 1, \ldots, B$ (e.g., $B = 1000$):

1. Draw a **bootstrap sample** $D^{*(b)}$ by sampling n points **with replacement** from $\{1, \ldots, n\}$.
2. Refit your model on $D^{*(b)}$ (or recompute the statistic) to get $\hat{\theta}^{*(b)} = s(D^{*(b)})$.

Use the empirical distribution of $\{\hat{\theta}^{*(b)}\}_{b=1}^B$ as an estimate of the sampling distribution of $\hat{\theta}$.

Standard error

$$\widehat{\mathrm{SE}}_{\mathrm{boot}}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\theta}^{*(b)} - \bar{\theta}^*\right)^2}, \quad \bar{\theta}^* = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}^{*(b)}.$$

60

60

Confidence Intervals (popular choices)

Let $\hat{\theta}^{*(\alpha)}$ be the α-quantile of $\{\hat{\theta}^{*(b)}\}$.

1. **Percentile CI** (simple, often good):

$$\left[\hat{\theta}^{*(\alpha/2)},\ \hat{\theta}^{*(1-\alpha/2)}\right].$$

2. **Basic CI** (inverts around $\hat{\theta}$):

$$\left[2\hat{\theta} - \hat{\theta}^{*(1-\alpha/2)},\ 2\hat{\theta} - \hat{\theta}^{*(\alpha/2)}\right].$$

3. **Studentized (bootstrap-t) CI** (more accurate if you can estimate a per-replicate SE):

$$\text{Compute } t^{*(b)} = \frac{\hat{\theta}^{*(b)} - \hat{\theta}}{\widehat{\text{SE}}^{*(b)}}, \quad \text{then } \left[\hat{\theta} - t^{*(1-\alpha/2)}\widehat{\text{SE}},\ \hat{\theta} - t^{*(\alpha/2)}\widehat{\text{SE}}\right].$$

4. **BCa (bias-corrected and accelerated)** (often best in practice; corrects bias & skew):

$$\text{CI}_{\text{BCa}} = \left[\hat{\theta}^{*(\alpha_1)},\ \hat{\theta}^{*(\alpha_2)}\right],$$

with adjusted quantiles α_1, α_2 computed from a **bias** term z_0 and an **acceleration** a (via jackknife). Most stats packages implement BCa.

61

61

Bootstrapping: doing something seemingly impossible

- pull oneself up by one's bootstraps — improve one's position by one's own efforts without help from others.
- In our context: A general method of calculating uncertainty estimates without any additional data. (Efron, 1979)

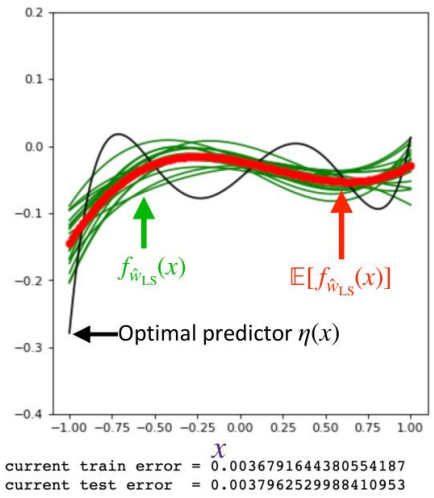
62

62

Bootstrapping: doing something seemingly impossible

to pull oneself up by one's bootstraps

Remember bias-variance trade-off?




```
current train error = 0.0036791644380554187
current test error  = 0.0037962529988410953
```

It is seemingly impossible to compute **Variance**, for example, with a single dataset.

63

63

Confidence interval

- suppose you have training data $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some true distribution $P_{x,y}$
- we train a kernel ridge regressor, with some choice of a kernel $K : \mathbb{R}^{d \times d} \to \mathbb{R}$

$$\text{minimize}_\alpha \; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

- the resulting predictor is

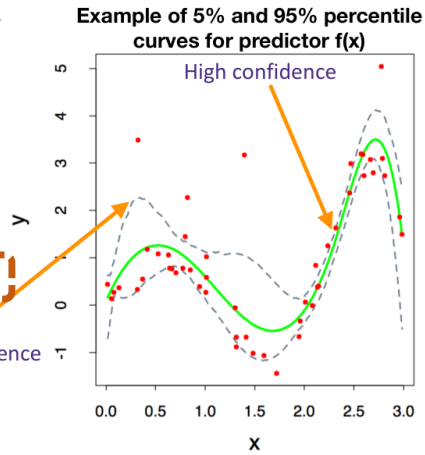
$$f(x) = \sum_{i=1}^n K(x_i, x)\hat{\alpha}_i,$$

where

$$\hat{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \quad \in \mathbb{R}^n$$

- we wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles

Low confidence
Why?

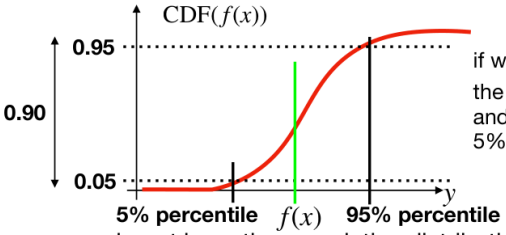


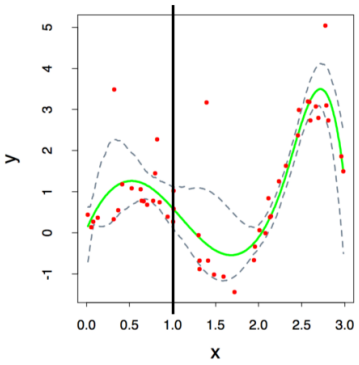
64

64

Confidence interval

- let's focus on a single $x \in \mathbb{R}^d$
- note that our predictor $f(x)$ is a random variable, whose randomness comes from the training data $S_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- if we know the statistics (in particular the CDF of the random variable $f(x)$) of the predictor, then the **confidence interval** with **confidence level 90%** is defined as





if we know the distribution of our predictor $f(x)$, the green line is the expectation $\mathbb{E}[f(x)]$ and the black dashed lines are the 5% and 95% percentiles in the figure above

- as we do not have the cumulative distribution function (CDF), we need to approximate them

65

65

Bootstrap

- as we cannot sample repeatedly (in typical cases), we use **bootstrap samples** instead
- bootstrap is a general tool for assessing statistical accuracy
- we learn it in the context of confidence interval for trained models

- a **bootstrap dataset** is created from the training dataset by taking n (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^n$

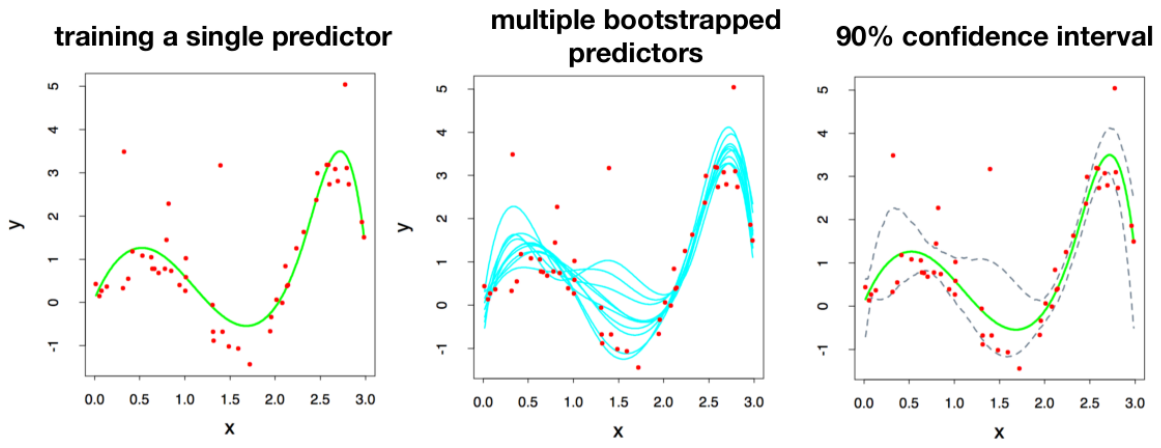
- for b=1,…,B
  - create a bootstrap dataset $S_{\text{bootstrap}}^{(b)}$
  - train a regularized kernel regression $\alpha^{*(b)}$
  - predict $\hat{y}^{(b)} = \sum_{i=1}^{n} K(x_i^{(b)}, x)\alpha_i^{*(b)}$

- compute the empirical CDF from the bootstrap datasets, and compute the confidence interval

66

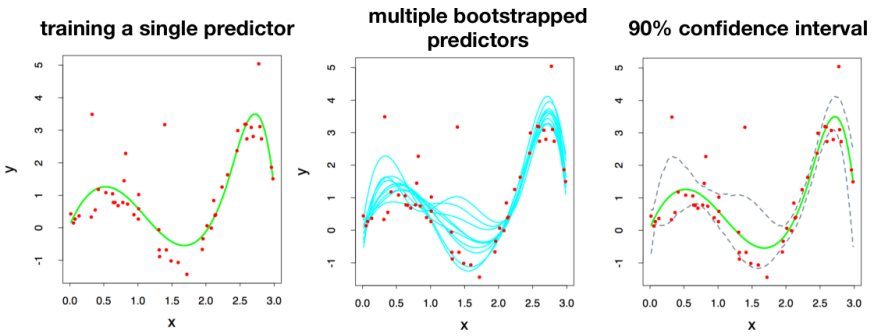
66

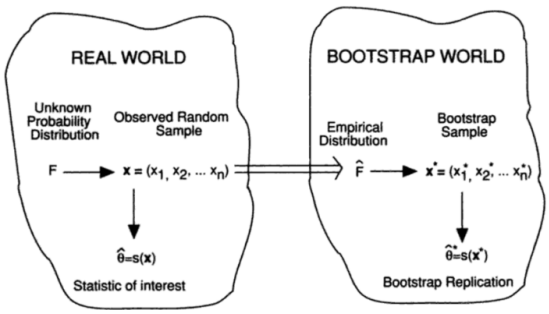
Bootstrap



67

67





68

68

Takeaways

Advantages:

- Very simple to use and generally applicable – build a confidence interval around anything
- Appears to give meaningful results even when the amount of data is very small
- Very strong asymptotic theory (as number of examples goes to infinity)

Disadvantages:

- Very few meaningful finite-sample guarantees
- Potentially computationally intensive
- Reliability hinges on test statistic and rate of convergence of empirical CDF to true CDF, which is unknown
- Poor performance on "extreme statistics" (e.g., the max)

69

69

PART 2
Code Time!

70

70

Code Time

- See demonstration and discussion in class.
- See also links in “Code Examples” for this Lecture Assignment.

71

71

Conclusion
Takeaways

72

72

KNN

- In kNN, there are no parameters to learn.
- Model is literally the dataset itself.
- kNN does not optimize anything
  - It stores the training data in memory – to be used for future predictions.

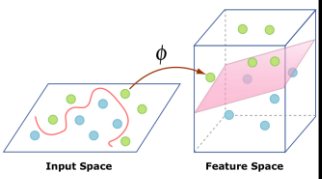
73

73

Kernel methods = nonlinear power + linear elegance, using similarity functions to project data implicitly into rich feature spaces

- **Kernels turn linear algorithms into nonlinear ones**

A kernel defines an implicit feature mapping $\phi(x)$ so that inner products in high-dimensional (possibly infinite-dimensional) spaces can be computed directly in input space as $K(x, z) = \langle \phi(x), \phi(z) \rangle$.
→ This lets linear models (SVM, ridge regression, PCA) learn **nonlinear patterns** without ever computing $\phi(x)$ explicitly.



- **Kernel functions measure similarity between datapoints**

Kernels express how "close" or "similar" two samples are under some geometry (e.g., RBF for locality, polynomial for global interactions).
→ The choice of kernel is the **choice of feature space** and defines the model's inductive bias.

- **Learning with kernels = linear combination of training examples**

The learned predictor has the form

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i),$$

where coefficients α_i weight each training example's influence.
→ The model's complexity grows with data, not fixed parameters — bridging statistics and geometry.

74

74

References and Credits

Many of the teaching materials for this course have been adapted from various sources. We are very grateful and thank the following professors, researchers, and practitioners for sharing their teaching materials (in no particular order):

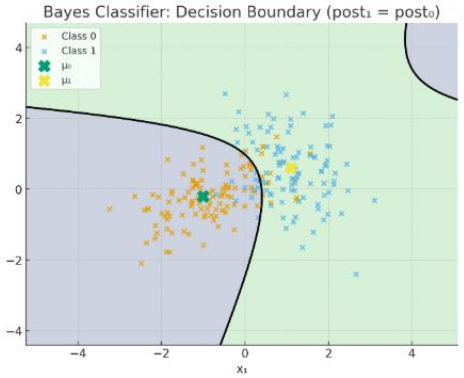
- Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. https://amlbook.com/slides.html
- Ethem Alpaydin. https://www.cmpe.boun.edu.tr/~ethem/i2ml3e/
- Natasha Jaques. https://courses.cs.washington.edu/courses/cse446/25sp/
- Lyle Ungar. https://alliance.seas.upenn.edu/~cis520/dynamic/2022/wiki/index.php?n=Lectures.Lectures
- Aurelien Geron. https://github.com/ageron/handson-ml3
- Sebastian Raschka. https://github.com/rasbt/machine-learning-book
- Trevor Hastie. https://www.statlearning.com/resources-python
- Andrew Ng. https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU
- Richard Povineli. https://www.richard.povinelli.org/teaching
- … and many others.

Appendix A – The Bayes Classifier

The **Bayes classifier** is a probabilistic decision rule that assigns each sample to the class with the highest posterior probability $P(C_k|x)$.

When class-conditional densities are Gaussian, this yields elegant, often smooth (linear or quadratic) decision boundaries that reflect the underlying probability structure of the data.

Concept	Description
Goal	Minimize classification error by using posterior probabilities.
Rule	Assign x to the class with highest posterior (P(C_k
Decision boundary	Set of points where posteriors are equal.
Shape	Linear if covariances are equal, quadratic otherwise.
Optimality	Bayes classifier achieves the minimum possible error (Bayes error rate).



1. The Bayes Classifier — Core Idea

The **Bayes classifier** is the **optimal classifier** (in terms of minimum probability of misclassification) when the true underlying class-conditional distributions are known.

We predict the class with the **highest posterior probability**:

$$\boxed{\hat{y}(x) = \arg\max_k \; P(C_k \mid x)}$$

---

2. Bayes' Theorem and Posterior Computation

Bayes' theorem gives the posterior as:

$$P(C_k \mid x) = \frac{P(x \mid C_k)\, P(C_k)}{P(x)}$$

- $P(x \mid C_k)$ — **Likelihood**: how probable the data is under class C_k
- $P(C_k)$ — **Prior**: how probable class C_k is overall
- $P(x)$ — **Evidence**: same for all classes, acts as a normalizer

Since $P(x)$ is constant across classes, we can compare **unnormalized posteriors**:

$$\hat{y}(x) = \arg\max_k \; P(x \mid C_k)\, P(C_k)$$

77

77

3. Gaussian Example (Typical in Plots)

Assume each class follows a **multivariate Gaussian**:

$$P(x \mid C_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right]$$

If priors are equal ($P(C_1) = P(C_2)$), the classifier compares likelihoods — geometrically, it separates regions of space where one Gaussian dominates the other.

---

4. Decision Boundary

The **decision boundary** occurs where posteriors are equal:

$$P(x \mid C_1)P(C_1) = P(x \mid C_2)P(C_2)$$

Taking logs (and simplifying quadratic forms) leads to a **linear** or **quadratic** boundary depending on whether covariances Σ_k are equal or not.

- If $\Sigma_1 = \Sigma_2$: **Linear Discriminant Boundary**
- If $\Sigma_1 \neq \Sigma_2$: **Quadratic Discriminant Boundary**

78

78

Appendix B – Dual Formulation of the Support Vector Machine (SVM)

1. Starting Point: The Primal (Hard-Margin SVM)

The **primal optimization problem** for a linearly separable dataset is:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \quad i = 1, \ldots, n \end{aligned}$$

- Objective: minimize $\frac{1}{2}\|w\|^2$ → maximize the margin.
- Constraints: every point lies on the correct side of the margin.

79

79

2. Introducing Lagrange Multipliers

To handle constraints, we use **Lagrange multipliers** $\alpha_i \geq 0$ (one per constraint).

The **Lagrangian** combines the objective and constraints:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^\top x_i + b) - 1].$$

Here:

- α_i penalizes violations of $y_i(w^\top x_i + b) \geq 1$.
- Large α_i means the i-th constraint is tight (support vector).

80

80

🧮 3. Stationarity Conditions (KKT Conditions)

To find the saddle point of the Lagrangian, we take partial derivatives and set them to zero.

(a) Derivative w.r.t. w:

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0 \quad \Rightarrow \quad \boxed{w = \sum_i \alpha_i y_i x_i.}$$

This shows w is a **linear combination of the training samples**.

(b) Derivative w.r.t. b:

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_i \alpha_i y_i = 0 \quad \Rightarrow \quad \boxed{\sum_i \alpha_i y_i = 0.}$$

This ensures the separating hyperplane is correctly centered between classes.

81

81

🔁 4. Substituting Back to Eliminate w and b

Plug $w = \sum_i \alpha_i y_i x_i$ into the Lagrangian:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \| \sum_i \alpha_i y_i x_i \|^2 - \sum_i \alpha_i [y_i ((\sum_j \alpha_j y_j x_j)^\top x_i + b) - 1].$$

Simplify — after algebra and using $\sum_i \alpha_i y_i = 0$:

$$\boxed{\mathcal{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i^\top x_j).}$$

82

82

📈 5. The Dual Optimization Problem

We have now eliminated w, b.

The SVM **dual problem** is purely in terms of α_i:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^\top x_j) \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad \sum_i \alpha_i y_i = 0. \end{aligned}$$

This is a **quadratic optimization problem (QP)** in the α_i's.

🧠 6. Interpretation

Symbol	Meaning
α_i	The importance (Lagrange weight) of each training point
$\alpha_i > 0$	Point lies on the margin → **support vector**
$\alpha_i = 0$	Point lies outside the margin → irrelevant for boundary
$w = \sum_i \alpha_i y_i x_i$	Weight vector expressed in terms of support vectors
Constraint $\sum_i \alpha_i y_i = 0$	Keeps balance between classes

7. Decision Function (Prediction Rule)

Once we solve for the optimal multipliers α_i^*, we can compute:

$$\boxed{f(x) = \text{sign}\left(\sum_i \alpha_i^* y_i (x_i^\top x) + b^*\right)}$$

The offset b^* can be found from any **support vector** x_s using:

$$b^* = y_s - \sum_i \alpha_i^* y_i (x_i^\top x_s)$$

85

85

8. The Dual Enables the Kernel Trick

Note that in the dual formulation, data appear **only as inner products** $(x_i^\top x_j)$.

We can replace these with **kernel functions** $K(x_i, x_j)$:

$$K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

Thus, the **kernelized dual problem** is:

$$\boxed{\begin{aligned} \max_\alpha \quad & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad \sum_i \alpha_i y_i = 0. \end{aligned}}$$

Prediction:

$$f(x) = \text{sign}\left(\sum_i \alpha_i^* y_i K(x_i, x) + b^*\right)$$

✅ This is how **nonlinear SVMs** arise — the dual allows us to substitute kernels without ever computing $\phi(x)$ explicitly.

36

86

📘 9. Summary Table

Step	Concept	Key Expression
Primal	Minimize norm of w	$\min \frac{1}{2}\|w\|^2$ s.t. constraints
Dual variable	Introduce $\alpha_i \geq 0$	Enforce $y_i(w^\top x_i + b) \geq 1$
Stationarity	$w = \sum_i \alpha_i y_i x_i$	Shows dependence only on data
Dual problem	Quadratic in α	$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i^\top x_j)$
Kernelized dual	Replace dot products by kernels	$K(x_i, x_j)$
Solution	Support vectors where $\alpha_i > 0$	w depends only on them

87

87

Appendix C – Representer Theorem underlies Kernel Methods

Optimization (gradient descent trajectory) lives in a finite-dimensional subspace even when the feature space is infinite-dimensional

Linear model in feature space

Suppose each input $x \in \mathcal{X}$ is mapped by a feature map

$$\phi : \mathcal{X} \to \mathbb{R}^p,$$

where p could be **huge or infinite** (e.g., RBF kernels).

A linear predictor in this space is

$$f_w(x) = w^\top \phi(x),$$

with parameters $w \in \mathbb{R}^p$.

You train it by minimizing some empirical risk plus regularization, e.g. in kernel ridge regression:

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top \phi(x_i))^2 + \lambda\|w\|^2.$$

88

88

Gradient descent updates stay in the span of training features

Initialize $w_0 = 0$ (or any vector orthogonal to the data span).
Each gradient step is:

$$w_{t+1} = w_t - \eta_t \nabla_w L(w_t),$$

where

$$\nabla_w L(w_t) = \frac{1}{n} \sum_{i=1}^{n} (w_t^\top \phi(x_i) - y_i)\, \phi(x_i) + \lambda w_t.$$

Observe the **key fact**:

The gradient is always a linear combination of the feature vectors $\{\phi(x_i)\}_{i=1}^n$.

So if w_0 lies in their span, every update adds a combination of those same vectors, keeping w_t in that span forever.

Formally:

$$w_t \in \text{span}\{\phi(x_1), \ldots, \phi(x_n)\} \subseteq \mathbb{R}^p.$$

That span has dimension at most n (even though p may be infinite).

89

89

At the optimum: Representer Theorem

The same conclusion holds at the **optimal** solution of any regularized risk that depends on w only through its evaluations at training points:

$$\boxed{w^* = \sum_{i=1}^{n} \alpha_i\, \phi(x_i)}$$

for some coefficients $\alpha_i \in \mathbb{R}$.

This is the **Representer Theorem** (Kimeldorf & Wahba, 1971).

Plugging it back in:

$$f_{w^*}(x) = \sum_{i=1}^{n} \alpha_i\, \phi(x_i)^\top \phi(x) = \sum_{i=1}^{n} \alpha_i\, K(x_i, x),$$

which is the familiar kernel form.

90

90

Appendix D – First: define RBF (Radial Basis Function) Kernel

The radial basis function (RBF) kernel

The RBF kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = e^{-||\mathbf{x}-\mathbf{x}'||^2/2\sigma^2} = e^{-\gamma||\mathbf{x}-\mathbf{x}'||^2} \tag{104}$$

where $\gamma = 1/2\sigma^2$ is a parameter that can be adjusted to fit each specific dataset. This kernel can be wriiten as the inner product of two infinite dimensional vectors (for simplicity, we assume $\sigma = 1$):

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= e^{-||\mathbf{x}-\mathbf{x}'||^2/2} = e^{-||\mathbf{x}||^2/2}\, e^{-||\mathbf{x}'||^2/2}\, e^{\mathbf{x}^T\mathbf{x}'} = e^{-||\mathbf{x}||^2/2}\, e^{-||\mathbf{x}'||^2/2} \sum_{n=0}^{\infty} \frac{(\mathbf{x}^T\mathbf{x}')^n}{n!} \\ &= e^{-||\mathbf{x}||^2/2}\, e^{-||\mathbf{x}'||^2/2} \sum_{n=0}^{\infty} \left[\frac{1}{n!} \sum_{\sum_{i=1}^{d} k_i = n} \frac{n!}{k_1! \cdots k_d!} \left((x_1 x_1')^{k_1} \cdots (x_d x_d')^{k_d} \right) \right] \\ &= \sum_{n=0}^{\infty} \sum_{\sum_{i=1}^{d} k_i = n} \left(e^{-||\mathbf{x}||^2/2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \left(e^{-||\mathbf{x}'||^2/2} \frac{x_1'^{k_1} \cdots x_d'^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \\ &= \phi(\mathbf{x})^T \phi(\mathbf{x}') = \mathbf{z}^T \mathbf{z}' \end{aligned} \tag{105}$$

https://pages.hmc.edu/ruye/MachineLearning/lectures/ch8/node10.html

91

91

where

$$\mathbf{z} = \phi(\mathbf{x}) = \left[e^{-||\mathbf{x}||^2/2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}}, \left(n = 0, \cdots, \infty, \sum_{k=1}^{n} k_i = n \right) \right]^T \tag{106}$$

is a vector in an infinite dimensional space. In particular, when $d = 1$ we have

$$\begin{aligned} K(x, x') &= e^{-(x-x')^2/2} = e^{-x^2/2}\, e^{-x'^2/2}\, e^{xx'} = e^{-x^2/2}\, e^{-x'^2/2} \sum_{n=0}^{\infty} \frac{(xx')^n}{n!} \\ &= \sum_{n=0}^{\infty} (e^{-x^2/2}\, x^n/\sqrt{n!})\, (e^{-x'^2/2}\, x'^n/\sqrt{n!}) \end{aligned} \tag{107}$$

where $\mathbf{z} = \phi(x) = \left[e^{-x^2/2}\, x^n/\sqrt{n!},\ (n = 0, \cdots, \infty) \right]^T$ is a kernel function that maps a 1-D space into an infinite dimensional space.

https://pages.hmc.edu/ruye/MachineLearning/lectures/ch8/node10.html

92

92

Second: Use RBF (Radial Basis Function) Kernels

Derivation: Kernel Ridge Regression with RBF kernels

1) Start from ridge regression in feature space

For data $\{(x_i, y_i)\}_{i=1}^n$ and feature map $\phi(\cdot)$,

$$\min_w \|y - \Phi w\|^2 + \lambda \|w\|^2, \qquad \Phi = [\phi(x_1)^\top; \dots; \phi(x_n)^\top] \in \mathbb{R}^{n \times D}.$$

Normal equations:

$$(\Phi^\top \Phi + \lambda I) w = \Phi^\top y.$$

Representer theorem implies $w = \Phi^\top \alpha$ for some $\alpha \in \mathbb{R}^n$.
Plug in and solve for α:

$$\underbrace{(\Phi\Phi^\top + \lambda I)}_{K+\lambda I}\alpha = y, \quad \Rightarrow \quad \alpha = (K + \lambda I)^{-1} y,$$

where $K = \Phi\Phi^\top$ is the **kernel (Gram) matrix** with entries
$K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$.

93

93

2) Prediction at a new x

$$f(x) \;=\; \phi(x)^\top w \;=\; \phi(x)^\top \Phi^\top \alpha \;=\; k(x)^\top \alpha,$$

with $k(x) = [K(x, x_1), \dots, K(x, x_n)]^\top$.

Combine:

$$\boxed{f(x) = k(x)^\top (K + \lambda I)^{-1} y}.$$

3) Specialize the kernel to RBF

The **RBF (Gaussian) kernel** in $\mathbb{R}^d$:

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right) = \exp\big(-\gamma \|x - z\|^2\big), \quad \gamma = \frac{1}{2\sigma^2}.$$

Therefore

$$\boxed{f(x) = \sum_{i=1}^n \alpha_i \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right), \quad \alpha = (K + \lambda I)^{-1} y.}$$

Interpretation: the predictor is a **weighted sum of Gaussian bumps** centered at the training points, with widths controlled by σ and weights α_i set by λ and the data.

94

94