Support Vector Machines

19 February 2026

Alex Lyman

Map of every battle since 2500 BC according to Wikipedia (10,624 battles)

🏛️ Steven 🏛️

@nonregemesse

Map of who is good at recording battles

Linear Regression Review

Linear Regression

- In linear regression, we try to fit a hyperplane (line in 2-d case) to data.
- We calculate the residuals (distance between the line and each point).
- We want to minimize the residuals.
- We usually square the residuals.

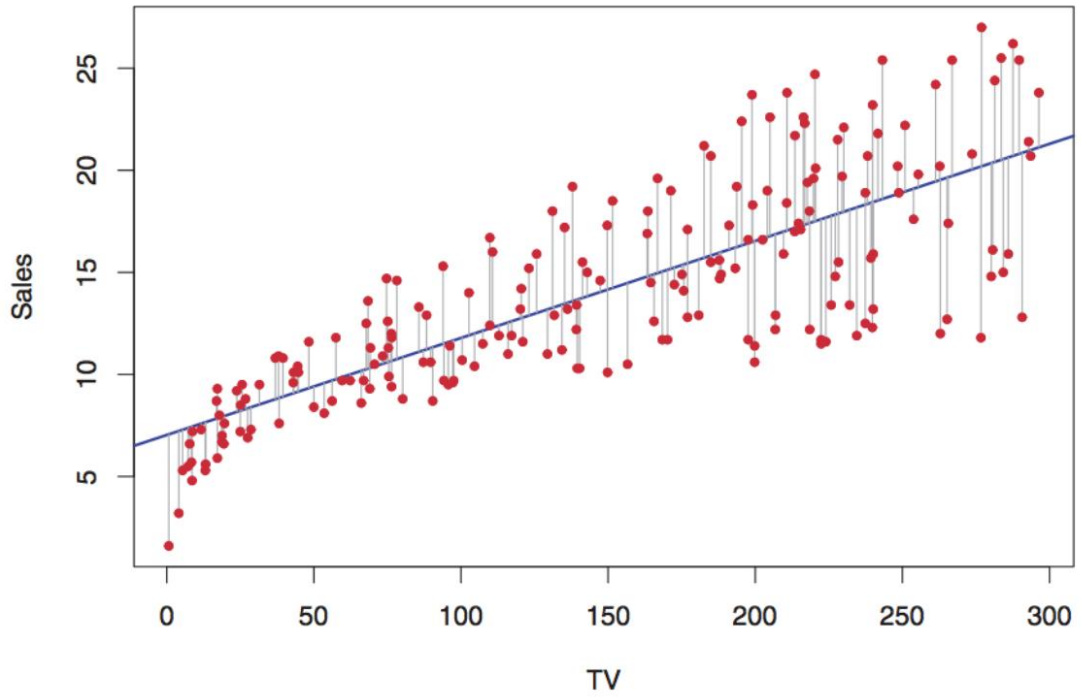


FIGURE 3.1, ISL (8th printing 2017)

Linear Regression Example

- L1 Loss
  - Add up the **absolute values** of all the distances between the points and the line.
- L2 Loss
  - Add up the **squares** of all the distances between the points and the line.
- Which one do we usually use?

$$\Sigma|t_j - z_j|$$

$$\Sigma(t_j - z_j)^2$$

y dependent variable (output)

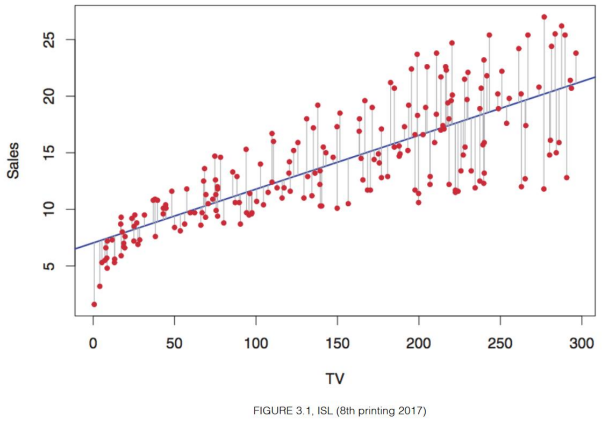
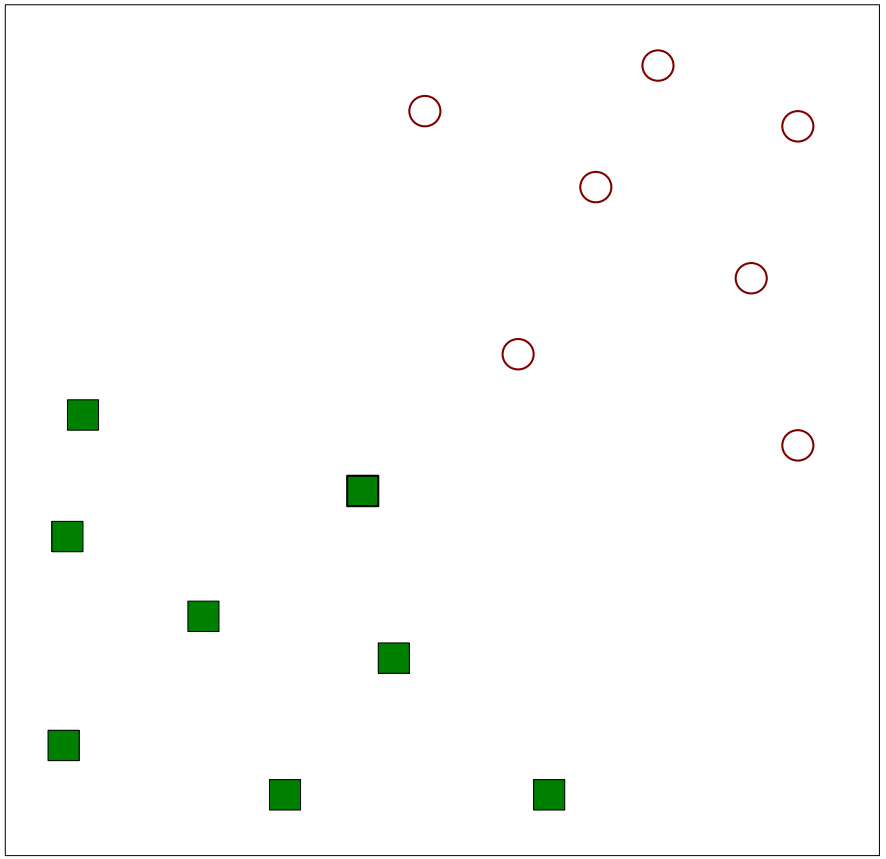


FIGURE 3.1, ISL (8th printing 2017)

x – independent variable (input)

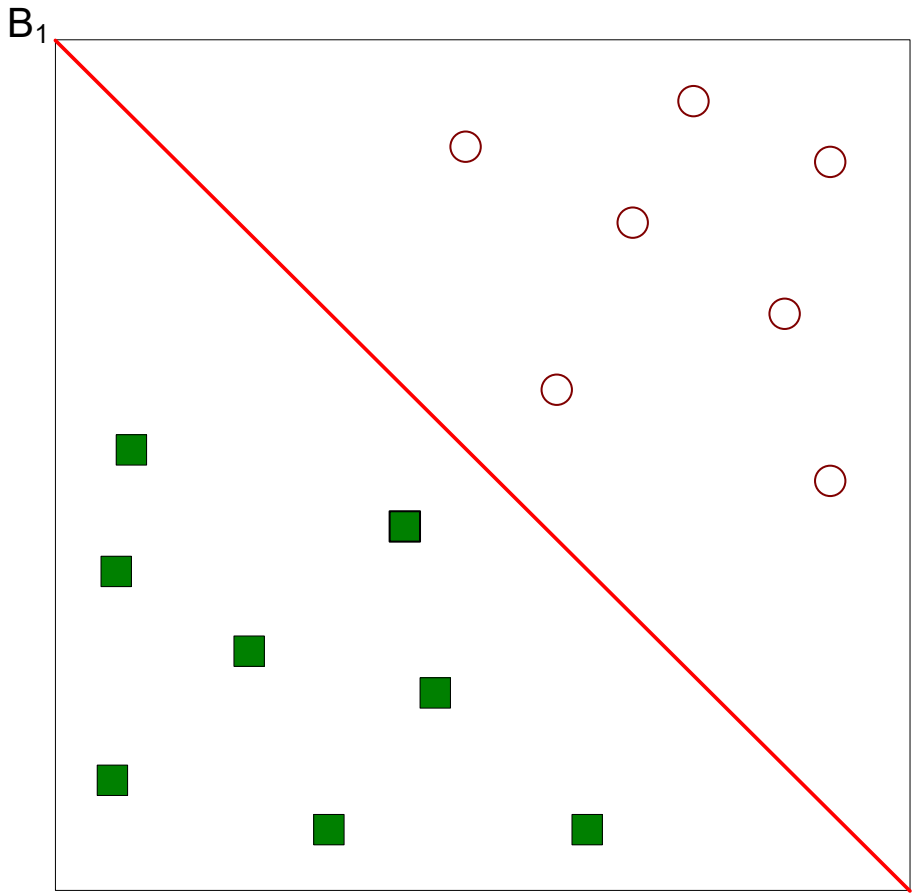
Maximum Margin Classifier

Maximum Margin Classifier



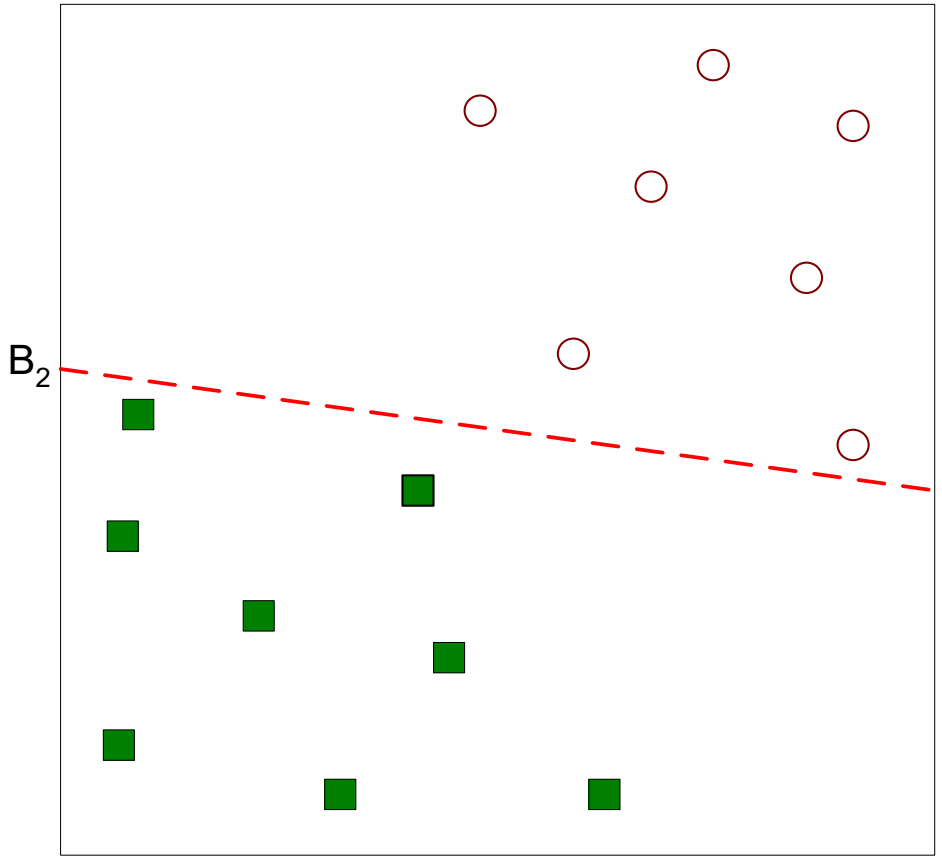
- Our goal: Find a linear hyperplane (decision boundary) that will separate the data
- Do we remember what a hyperplane is?

Maximum Margin Classifier



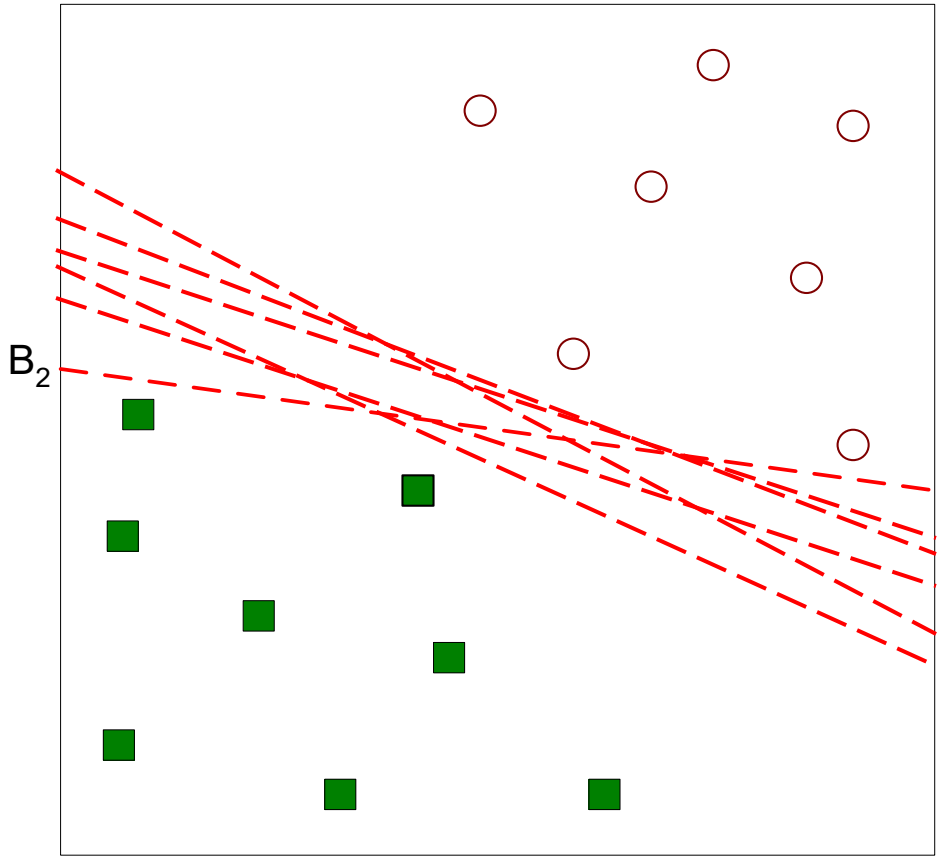
- One Possible Solution

Maximum Margin Classifier



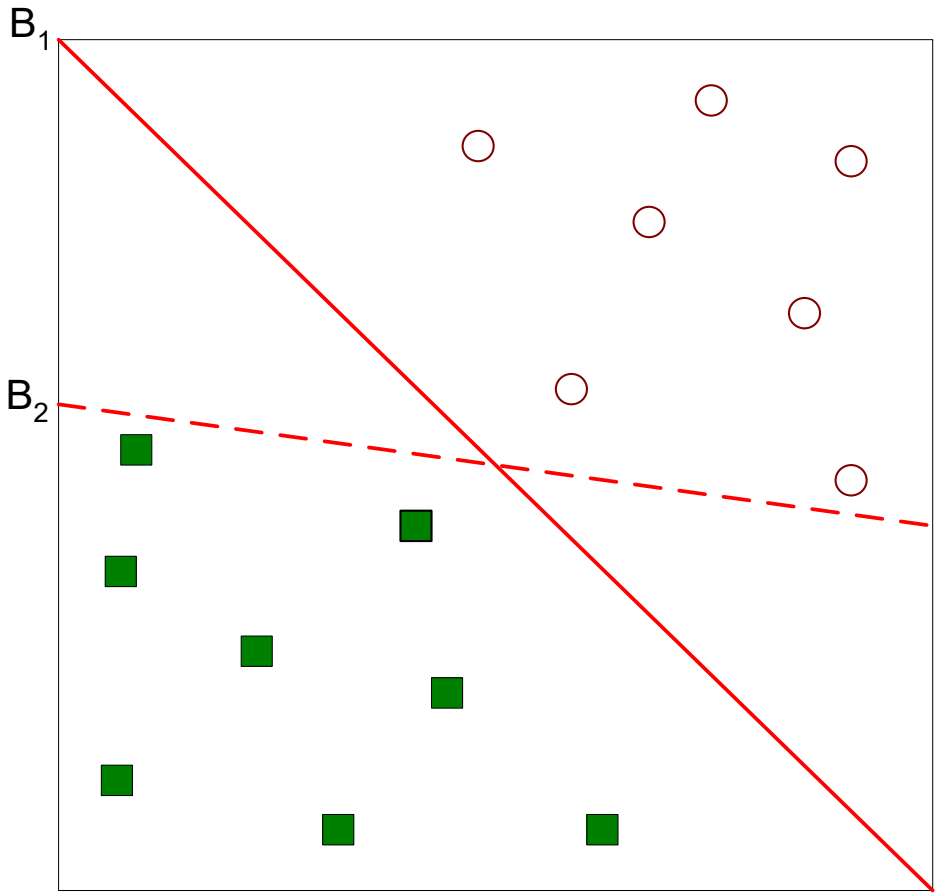
- Another possible solution

Maximum Margin Classifier



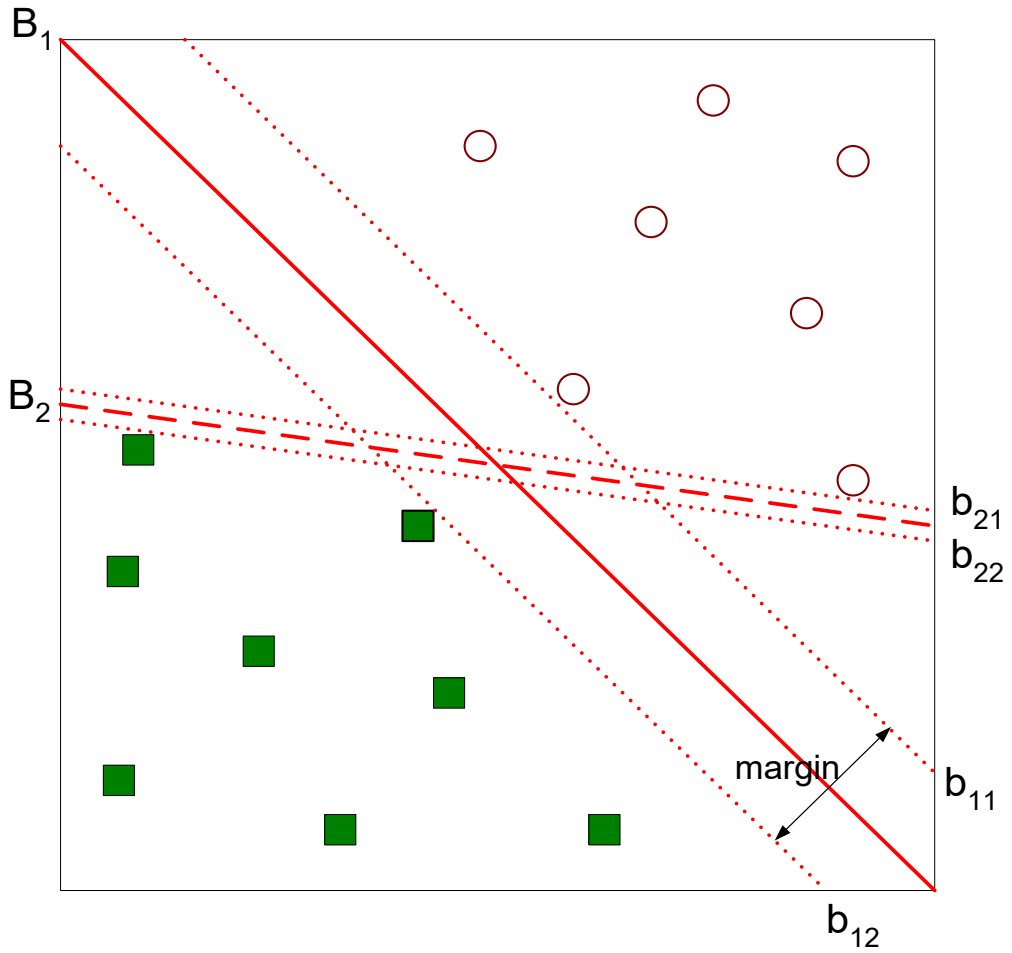
- Other possible solutions

Maximum Margin Classifier



- Which one is better? B1 or B2?
- How do you define better?

Maximum Margin Classifier



- Find hyperplane maximizes the margin => B1 is better than B2.
- Why?

Maximum Margin Classifier

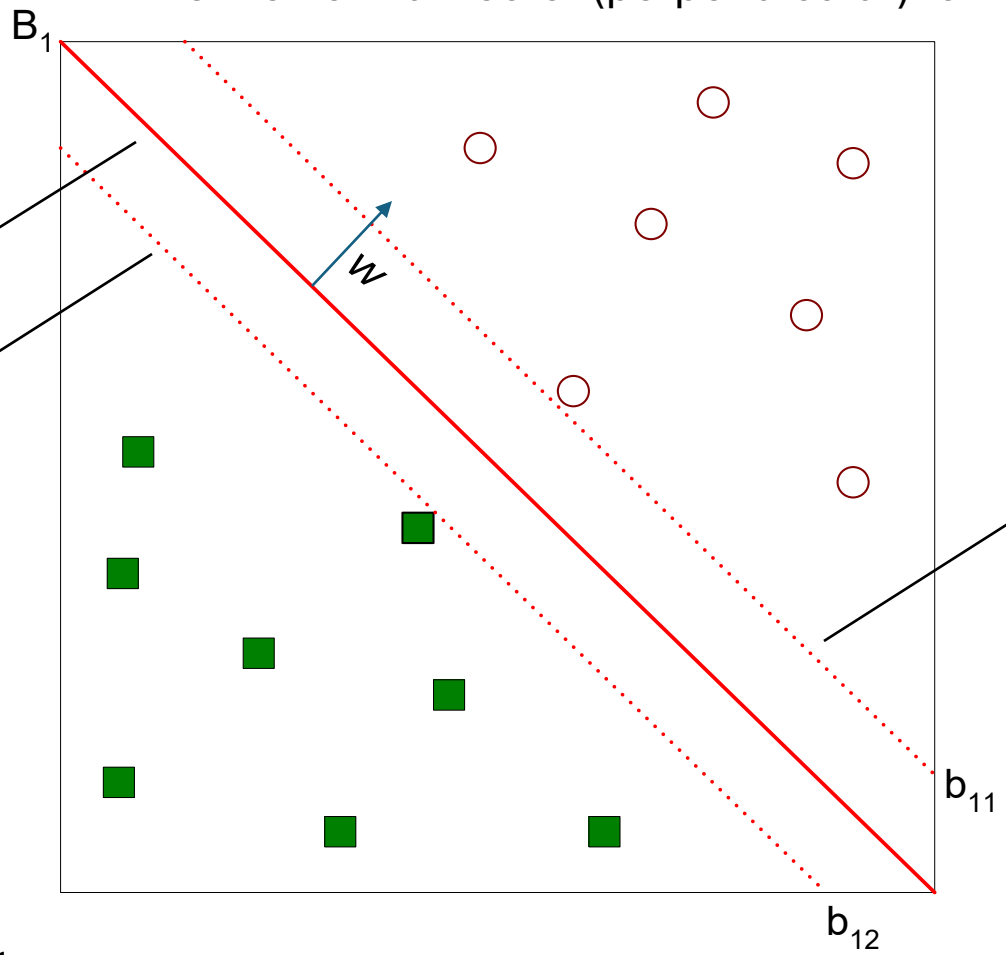
w is the normal vector (perpendicular) to B

Equation of the line or hyperplane in vector form

$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

b/||w|| determines the offset of the hyperplane from the origin along w



w **(The Weight Vector):** This vector is always perpendicular (normal) to the boundary. It determines the *orientation* or tilt of our line.

b **(The Bias):** This determines the offset. It shifts the line away from the origin.

$$\vec{w} \bullet \vec{x} + b = +1$$

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\| \vec{w} \|}$$

Maximum Margin Classifier

- You can think of this as a "Street"
- **Decision Boundary (Solid Line):**
  - This is where $w^T x + b = 0$.
  - The model isn't sure here. Score is 0.
- **Gutters (Dotted Lines):**
  - We insist that all positive points are *at least* +1 score: $w^T x + b_1 \geq 0$
  - We insist that all negative points are *at least* -1 score: $w^T x + b_1 \leq -1$
- **Width:**
  - The distance between these two dotted lines is mathematically $\frac{2}{||w||}$.
- **Intuition:**
  - To make the street **WIDER** (Maximize Margin), we need to make the denominator (||w||) **SMALLER**.
  - If you want a bigger slice of cake (Margin), you need to cut with a smaller knife (w).

Maximum Margin Classifier

- We want to maximize: $\text{Margin} = \frac{2}{||\vec{w}||}$
  - Which is equivalent to minimizing: $L(w) = \frac{\| \vec{w} \|^2}{2}$
  - But subjected to the following constraints:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 \text{ if } y_i = 1 \qquad \text{Class 1}$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 \text{ if } y_i = -1 \qquad \text{Class -1}$$

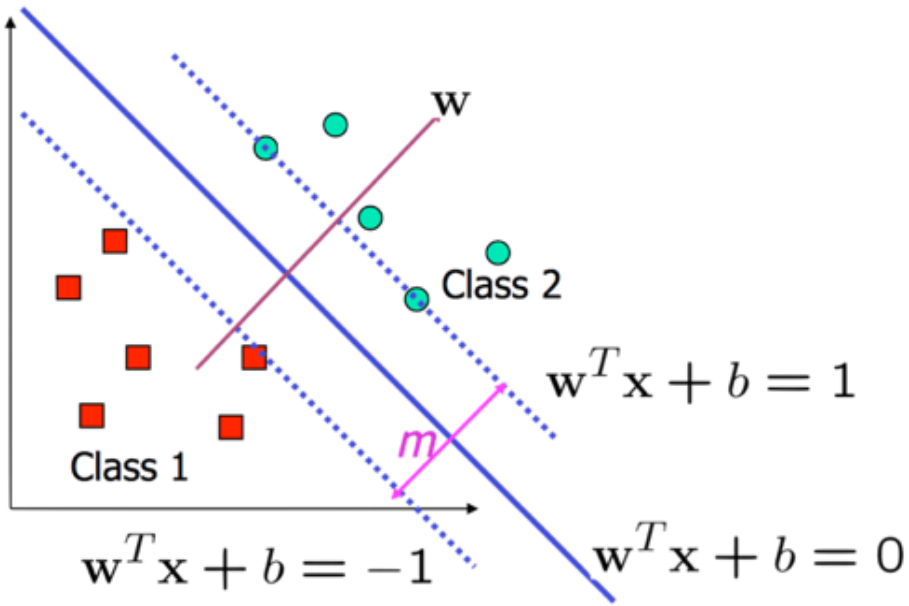
    - This is a constrained optimization problem
      - Numerical approaches to solve it (e.g., quadratic programming)

Maximizing Margins (cont.)

- Notice that maximizing the distance of ***all points*** to the decision boundary, is exactly the same as maximizing the distance to the ***closest points***.
- The points closest to the decision boundary are called ***support vectors***.

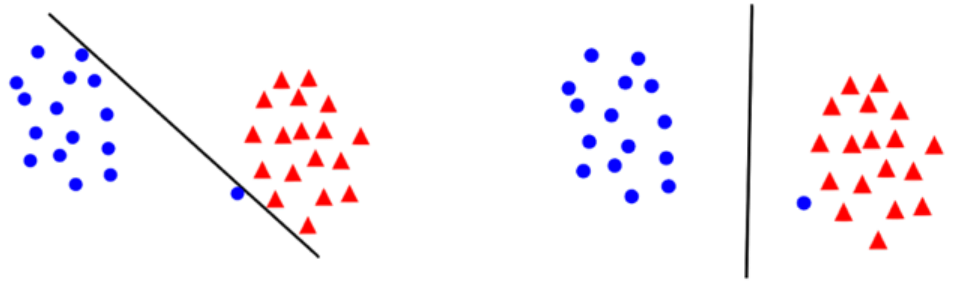
Maximizing Margins Illustration

- For points on planes $w^\top x + b = \pm 1$, their distance to the decision boundary is $\pm 1/\|w\|$.
- So we can define the ***margin*** of a decision boundary as the distance to its support vectors, $m = 2/\|w\|$.



Geometry of Data

- Maximizing the margin is a good idea as long as we assume that the underlying classes are linear separable and that the data is noise free.
- If data is noisy, we might be sacrificing generalizability to minimize classification error with a very narrow margin:

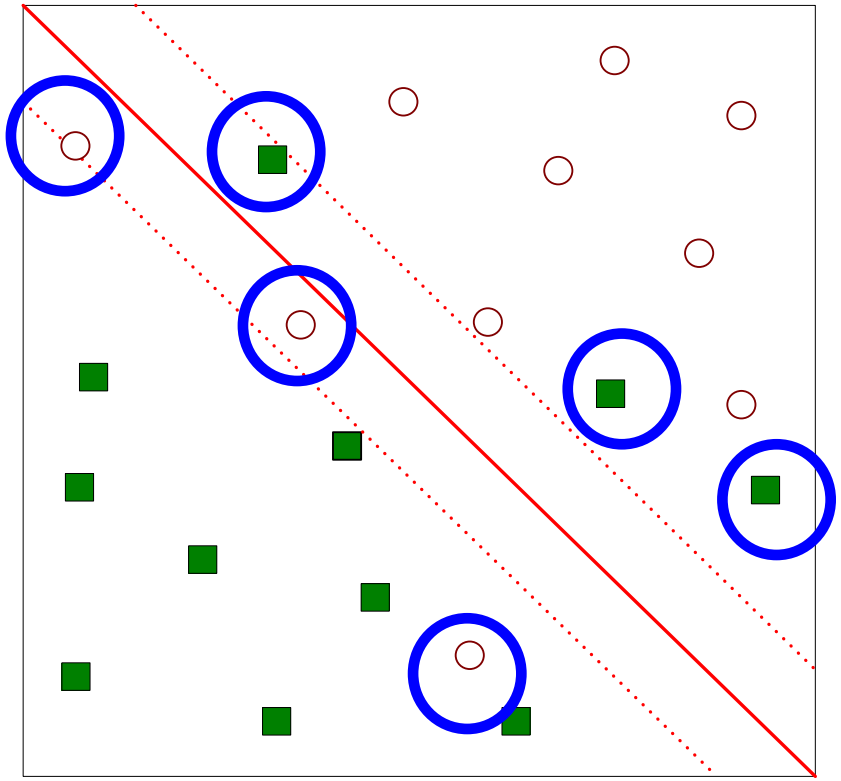


- With every decision boundary, there is a trade-off between maximizing margin and minimizing the error.

Support Vector Machines

Support Vector Machines

- What if the problem is not linearly separable?



Support Vector Machines

- If the problem is not linearly separable
  - Introduce slack variables
    - Need to minimize:

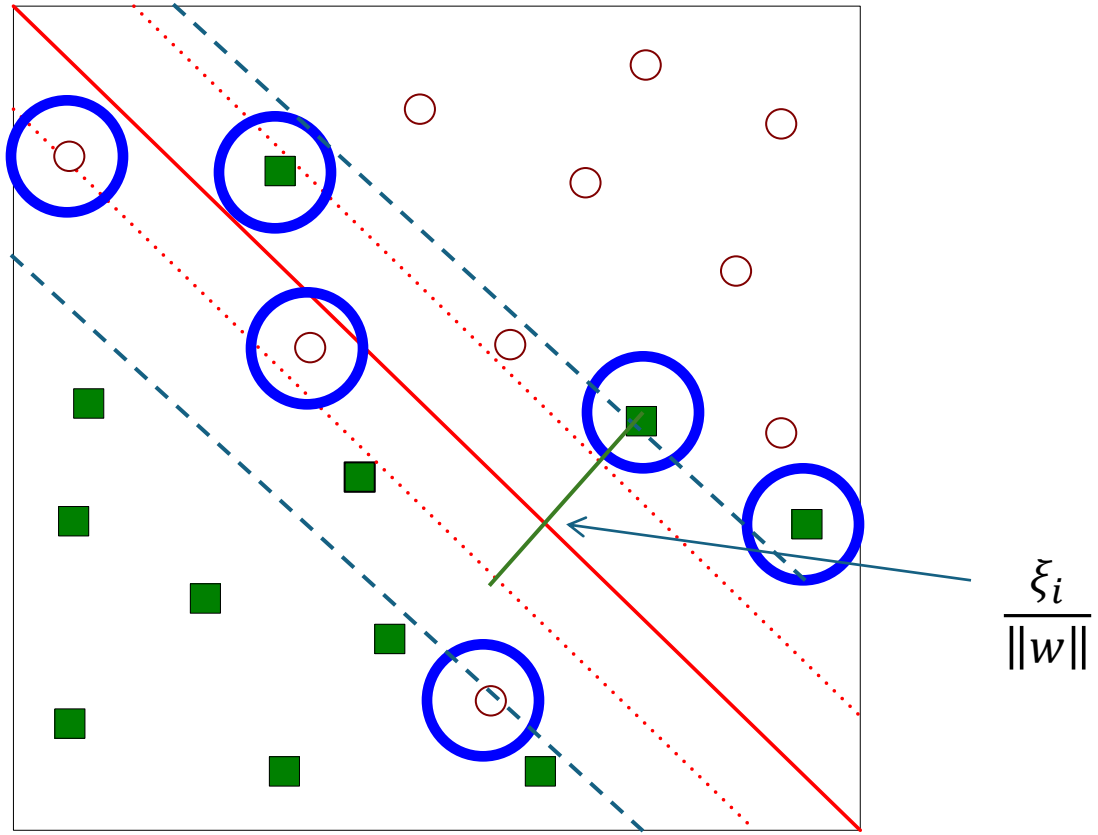
$$L(w) = \frac{\| \vec{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k \right)$$

    - Subject to:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 - \xi_i \text{ if } y_i = 1$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 + \xi_i \text{ if } y_i = -1$$

Support Vector Machines

- What if the problem is not linearly separable?



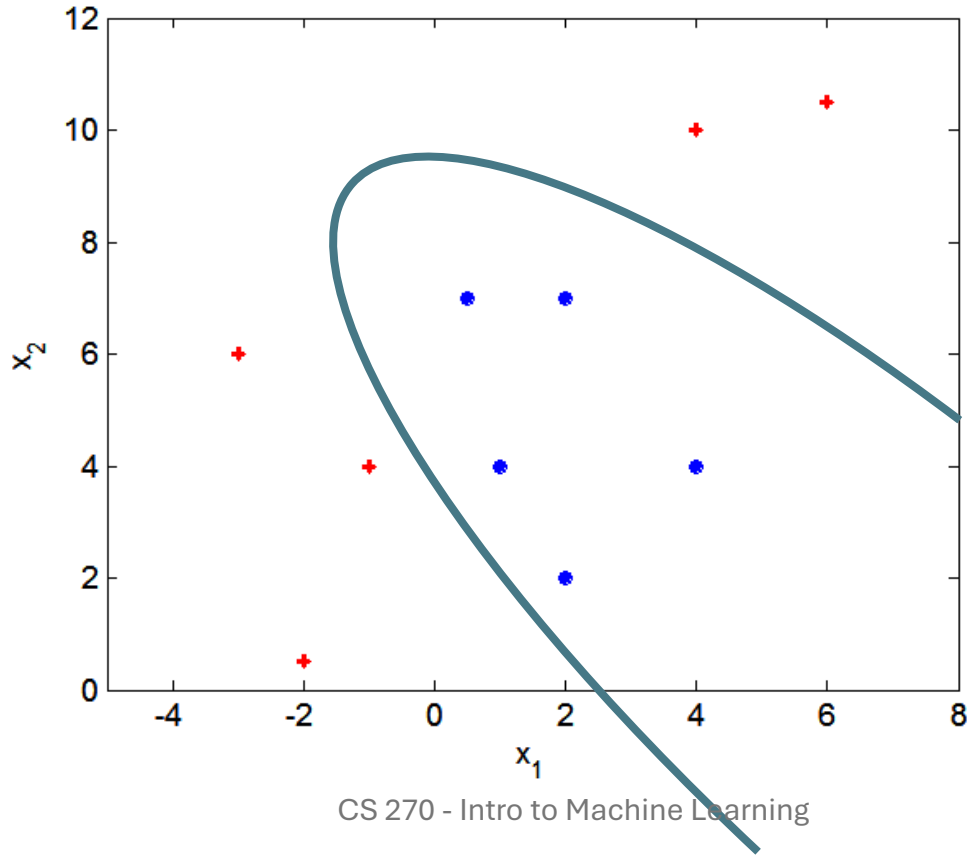
The "C" Hyperparameter: The Strictness Knob

- **The New Goal:** Minimize $||w||^2$ (Wide Margin) $+ C \times \sum(\text{Errors})$
- **What is C?**
  - C tells the SVM how much to punish misclassified points.
- **Low C (Lenient Parent):**
  - "I don't care if you make mistakes, just keep the street wide."
  - Result: Wider margin, more classification errors, **simpler model** (Underfitting risk).
- **High C (Strict Parent):**
  - "Do NOT make mistakes! I want every point correct!"
  - Result: Narrow margin, wiggles around data points, **complex model** (Overfitting risk).

Nonlinear Support Vector Machines

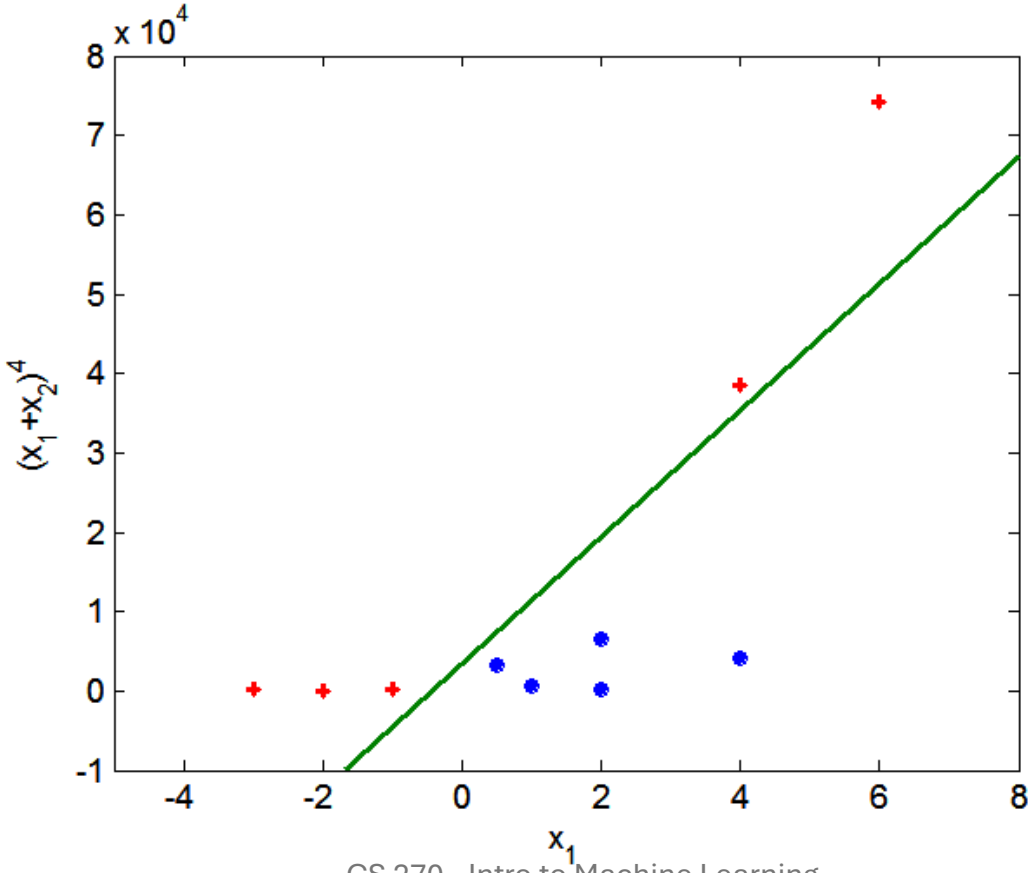
Nonlinear Support Vector Machines

- What if decision boundary is not linear?



Nonlinear Support Vector Machines

- Transform data into higher dimensional space



The Kernel Trick

- **Problem:**
  - Mapping data to higher dimensions (e.g., 100 dimensions or infinite dimensions) is computationally **expensive**.
  - Calculating the coordinates for every point would crash the computer.
- **The Trick:**
  - The math of SVMs only requires the **Dot Product** between points, not their coordinates.
- **Kernel Functions**
  - calculate the high-dimensional relationship *without* ever actually visiting the high-dimensional space.
- **The Benefit:**
  - We get the power of infinite dimensions with the speed of low dimensions.
  - It's like knowing the distance between two cities on a globe without actually having to build a globe.

The Kernel Trick

- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

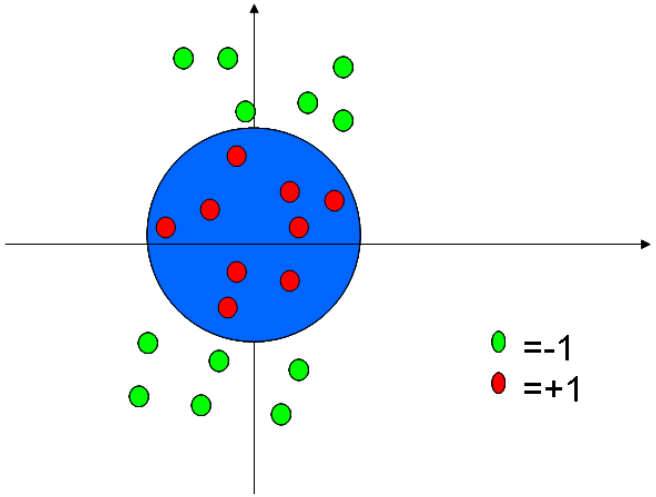
$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\phi}(\mathbf{x}_j)$$

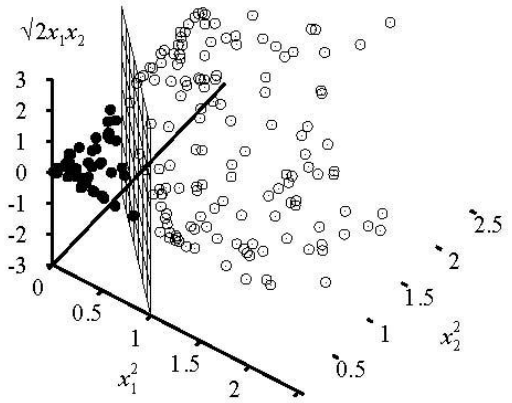
- A kernel trick requires a "kernel function" that can calculate the inner product between data points in a higher-dimensional feature space, without explicitly computing the coordinates of those points, allowing for non-linear classification by essentially performing a similarity comparison between data points in a transformed space, while only needing to operate in the original lower-dimensional space.

What is a Kernel?

- **Dot Products = Similarity**
- In a standard Linear SVM, the math relies entirely on taking the **Dot Product** between two data points ($x_i \cdot x_j$)
- Geometrically, a dot product is just a measure of **Similarity**.
  - If two points are pointing in the exact same direction, their dot product is large.
  - If they are perpendicular (unrelated), their dot product is 0.
- A **Kernel** $K(x_i, x_j)$) is simply a mathematical function that takes two data points from our original, messy space, and calculates their "Similarity Score."
- **The Catch:** It calculates what their similarity *would be* if we had moved them into a complex, high-dimensional space.
- $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$
- **ϕ(x)**: The function that lifts our data into 3D, 4D, or infinite dimensions.
- The Kernel function computes the final answer (a single number) directly from the original inputs, bypassing the ϕ(x) step entirely.

Kernel Trick





Data points are linearly separable in the space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$

We want to maximize $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left\langle F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) \right\rangle$

Define $K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) \right\rangle$

Cool thing : K is often easy to compute directly! Here,

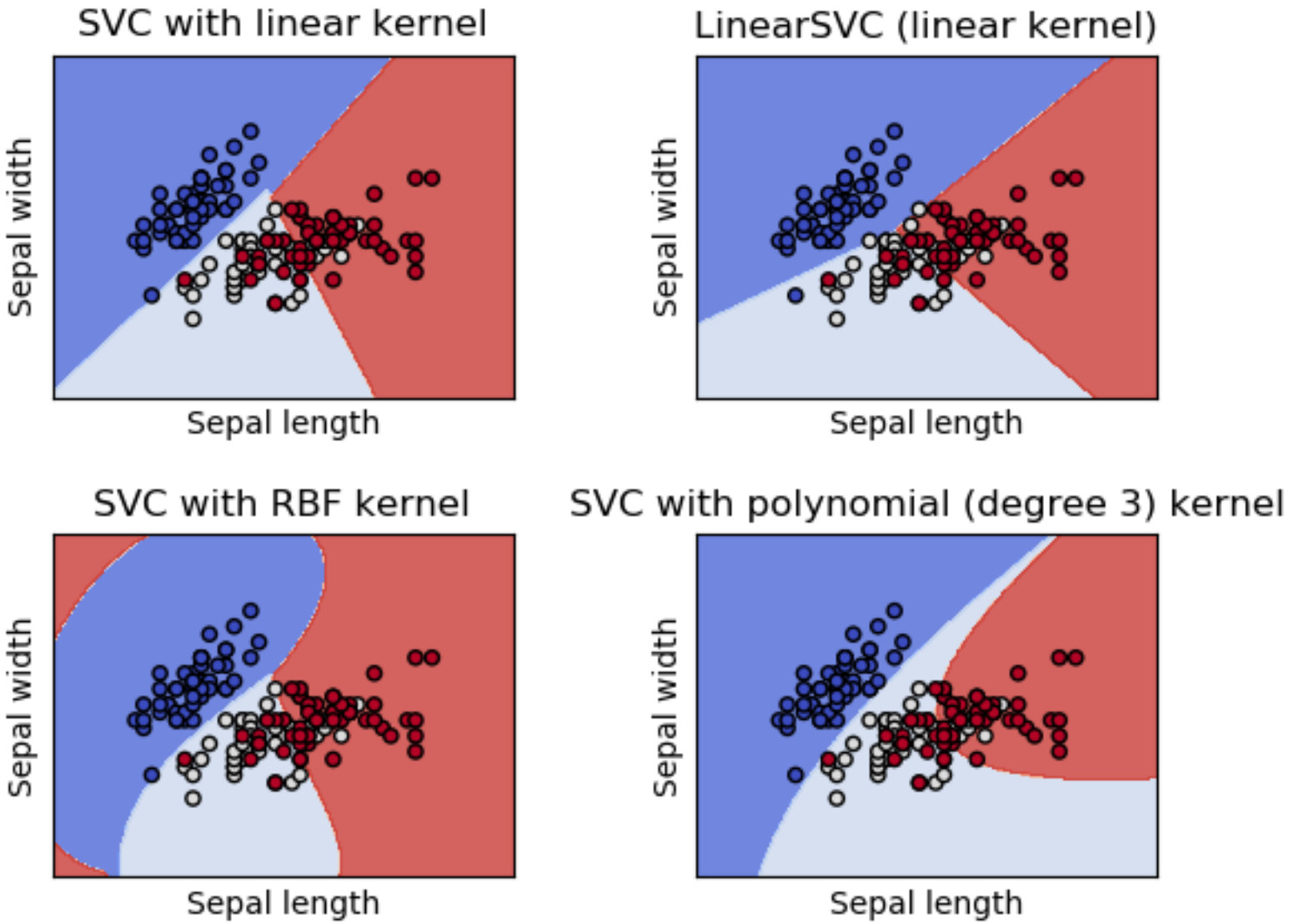
$K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \mathbf{x}_i \cdot \mathbf{x}_j \right\rangle^2$

Other Kernels

- linear $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
- polynomial $k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
- Gaussian or radial basis $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$
- sigmoid $k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$

The polynomial kernel
$K(x_i,x_j) = (\gamma\, x_i \bullet x_j + c)^d$, with tunable parameters.
Evaluating K only requires one addition and one exponentiation more than the original dot product but maps a polynomial decision boundary

Gaussian kernels (also called radial basis functions)
$K(x_i,x_j) = \exp(-\gamma\, ||x_i - x_j||^2)$ requires just vector subtract and norm


SVC with linear kernel
LinearSVC (linear kernel)
SVC with RBF kernel
SVC with polynomial (degree 3) kernel
Sepal width
Sepal length

sklearn.svm.SVC

class `sklearn.svm.SVC`(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer or other Kernel Approximation.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

To learn how to tune SVC's hyperparameters, see the following example: Nested versus non-nested cross-validation

Read more in the User Guide.

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`. For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

Algorithm	Use Case	Strengths	Weaknesses	Interpretability	Interpretability Influencers
Logistic Regression	Binary or multi-class classification, linear decision boundary	Simple and interpretable, handles large feature spaces	Assumptions of linearity, sensitive to outliers	High	1. Linearity 2. Simplicity 3. Feature Space Handling
Naive Bayes	Text classification, spam filtering	Fast and scalable, handles high-dimensional data	Assumes independence of features	Moderate	1. Speed 2. High Dimensions 3. Independence
Decision Trees	Data with non-linear relationships, feature importance	Non-linear decision boundaries, interpretable	Prone to overfitting, can be unstable	Moderate	1. Non-linearity 2. Feature Importance 3. Stability
K Nearest Neighbors	Image recognition, recommendation systems	Non-parametric, captures local patterns	Sensitive to irrelevant features, computationally expensive	Low	1.Non-parametric2. Local Patterns 3. Computational expensive
Support Vector machines	Text classification, image recognition	Effective in high-dimensional spaces, handles complex data	Less effective with large datasets, parameter tuning required	Low	1. High Dimensions 2. Complexity 3. Parameter Tuning

Scaling

- You **MUST** scale your data before using SVM.
- **Why?**
  - SVM tries to maximize margin (distance). If one axis has huge numbers, the margin will be biased entirely toward that axis.
- Use StandardScaler or MinMaxScaler in sklearn.

Code Time!