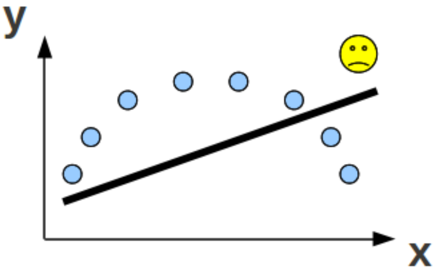
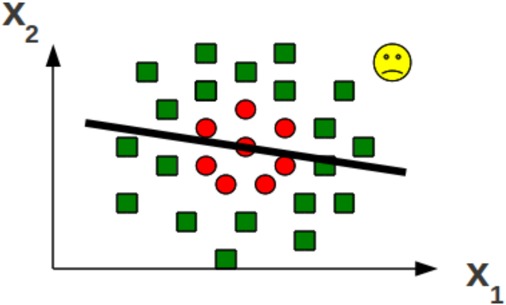
Linear Models

- Nice and interpretable but can't learn "difficult" nonlinear patterns





- So, are linear models useless for such problems?

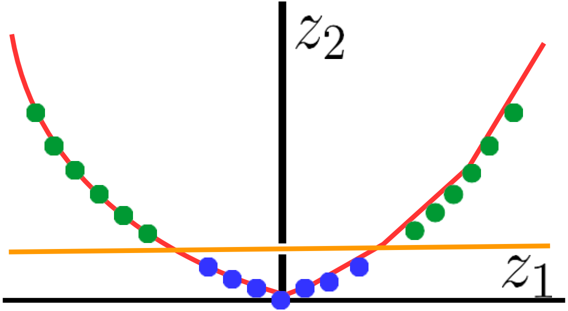
Linear Models for Nonlinear Problems!

- Consider the following one-dimensional inputs from two classes

- Can't separate using a linear hyperplane

Linear Models for Nonlinear Problems!

- Consider mapping each x to two-dimensions as $x \to \mathbf{z} = [z_1, z_2] = [x, x^2]$



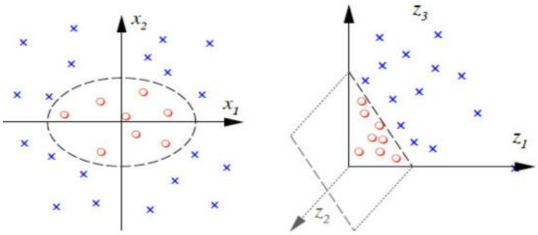
- Data now becomes linearly separable in the two-dimensional space

Linear Models for Nonlinear Problems!

- Essentially, can use some function φ to map/transform inputs to a "nice" space

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

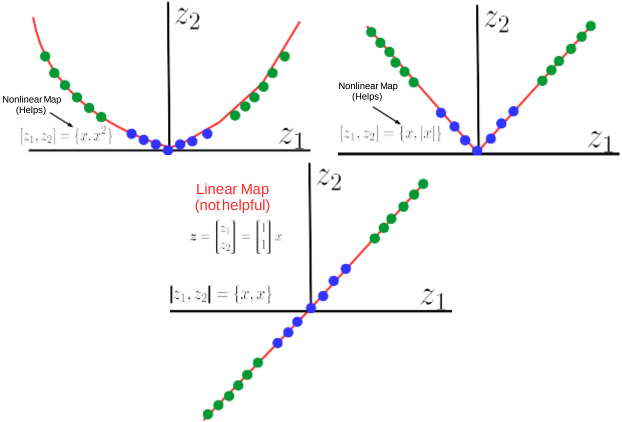
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)}x_1x_2, x_2^2)$$



- .. and then happily apply a linear model in the new space!
- Linear in the new space but nonlinear in the original space!

Not Every Mapping is Helpful

- Not every mapping helps in learning nonlinear patterns. Must at least be nonlinear!
- For the nonlinear classification problem we saw earlier, consider some possible mappings

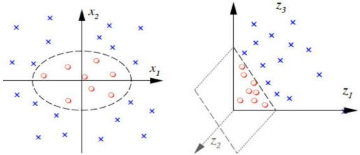


How to get these "good" (nonlinear) mappings?

- Can try to learn the mapping from the data itself (e.g., using deep learning - later)
- There are also pre-defined "good" mappings (e.g., provided by kernel functions - today's topic)

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1x_2, x_2^2)$$



- Looks like I have to compute these mapping using φ. That would be quite expensive!
- Thankfully, not always. For example, when using kernels, you get these for (almost) free
  - A kernel defines an "implicit" mapping for the data

Some Examples of Kernel Functions

- Linear (trivial) Kernel:

$$k(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{x}^\top \boldsymbol{z} \quad \text{(mapping function } \boldsymbol{\varphi} \text{ is identity)}$$

- Quadratic Kernel:

$$k(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2 \quad \text{or } (1 + \boldsymbol{x}^\top \boldsymbol{z})^2$$

- Polynomial Kernel (of degree d):

$$k(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^d \quad \text{or } (1 + \boldsymbol{x}^\top \boldsymbol{z})^d$$

- Radial Basis Function (RBF) of “Gaussian” Kernel:

$$k(\boldsymbol{x},\boldsymbol{z}) = \exp[-\gamma||\boldsymbol{x} - \boldsymbol{z}||^2]$$

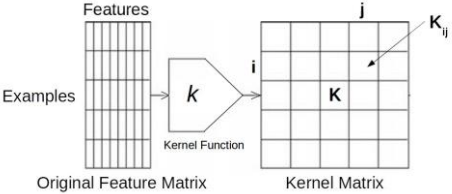
  - γ is a hyperparameter (also called the kernel bandwidth)
  - The RBF kernel corresponds to an infinite dimensional feature space F (i.e., you can’t actually write down or store the map $\boldsymbol{\varphi}(\boldsymbol{x})$ explicitly)
  - Also called “stationary kernel”: only depends on the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$ (translating both by the same amount won’t change the value of $k(\boldsymbol{x},\boldsymbol{z})$)
- Kernel hyperparameters (e.g., d, γ) need to be chosen via cross-validation

The Kernel Matrix

- The kernel function k defines the Kernel Matrix $\mathbf{K}$ over the data Given
- N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the (i, j)-th entry of $\mathbf{K}$ is defined as:

$$K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^\top \boldsymbol{\varphi}(\boldsymbol{x}_j)$$

- K_{ij}: Similarity between the i -th and j-th example in the feature space F
- $\mathbf{K}$: $N \times N$ matrix of pairwise similarities between examples in F space



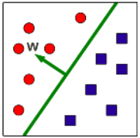
- $\mathbf{K}$ is a symmetric and positive definite matrix
- For a P.D. matrix: $\boldsymbol{z}^\top \mathbf{K} \boldsymbol{z} > 0, \quad \forall \boldsymbol{z} \in \mathbb{R}^N$ (also, all eigenvalues positive)
- The Kernel Matrix $\mathbf{K}$ is also known as the Gram Matrix

Using Kernels

- Kernels can turn a linear model into a nonlinear one
- Recall: Kernel $k(\boldsymbol{x}, \boldsymbol{z})$ represents a dot product in some high dimensional feature space F
- Any learning model in which, during training and test, inputs only appear as dot products ($\boldsymbol{x}_i^\top \boldsymbol{x}_j$) can be kernelized (i.e., non-linearlized)
  - .. by replacing the $\boldsymbol{x}_i^T \boldsymbol{x}_j$ terms by $\boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{\varphi}(\boldsymbol{x}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- Most learning algorithms can be easily kernelized
  - Distance based methods, Perceptron, SVM, linear regression, etc.
  - Many of the unsupervised learning algorithms too can be kernelized (e.g., K -means clustering, Principal Component Analysis, etc. - will see later)
  - Let's look at two examples: Kernelized SVM and Kernelized Ridge Regression

Hyperplane-based Classification

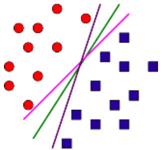
- Basic idea: Learn to separate by a hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$



- Predict the label of a test input $\boldsymbol{x}_*$ as: $\hat{y}_* = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_* + b)$
- The hyperplane may be "implied" by the model, or learned directly
  - Implied: Prototype-based classification, nearest neighbors, generative classification, etc.
  - Directly learned: Logistic regression, Perceptron, Support Vector Machine, etc.
- The "direct" approach defines a model with parameters $\boldsymbol{w}$ (and optionally b) and learns them by minimizing a suitable loss function (and doesn't model $\boldsymbol{x}$, i.e., purely discriminative)
- The hyperplane need not be linear (e.g., can be made nonlinear using kernel methods - next class)

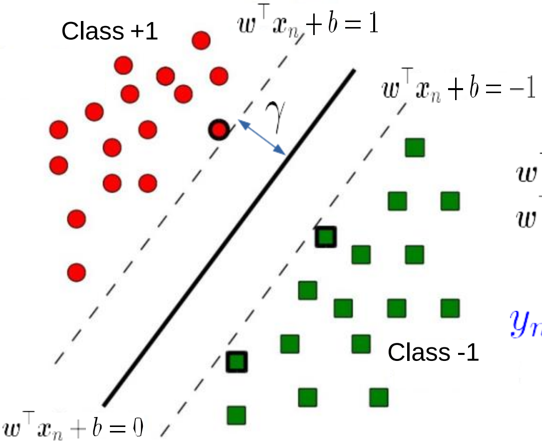
Lack of Margins

- Learns a hyperplane (of many possible) that separates the classes



- The one learned will depend on the initial ***w***
- Doesn't guarantee any "margin" around the hyperplane Note: Possible to
- "artificially" introduce a margin
  - **Support Vector Machine (SVM)** does this directly by learning the maximum margin hyperplane
  -

Hyperplanes and Margin



$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 \text{ for } y_n = +1$$
$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 \text{ for } y_n = -1$$

$$\Downarrow$$

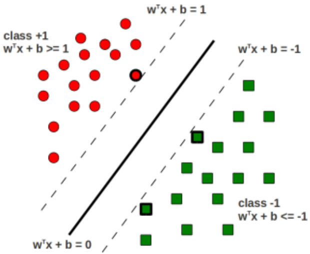
$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$$

$$\gamma = \min_{1 \leq n \leq N} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_n + b|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}$$

(Margin on each side)

Support Vector Machine (SVM)

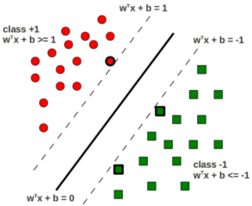
- SVM is a hyperplane based (linear) classifier that ensures a large margin around the hyperplane Note:
- We will assume the hyperplane to be of the form $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ (will keep the bias term b)



- Note: SVMs can also learn nonlinear decision boundaries using **kernel methods** (will see later)
- Reason behind the name "Support Vector Machine"?
  - SVM optimization discovers the most important examples (called "support vectors") in training data
  - These examples act as "balancing" the margin boundaries (hence called "support")

Hard-Margin SVM

- Hard-Margin: Every training example has to fulfil the margin condition $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$



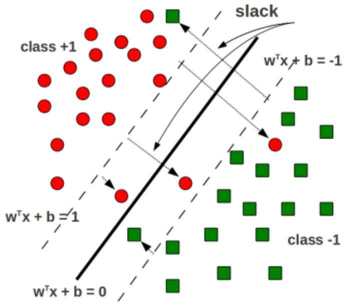
- Also want to maximize the margin $\gamma \propto \frac{1}{||\boldsymbol{w}||}$. Equivalent to minimizing $||\boldsymbol{w}||^2$ or $\frac{||\boldsymbol{w}||^2}{2}$
- The objective for hard-margin SVM

$$\min_{\boldsymbol{w},b} f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$
$$\text{subject to} \quad y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

- Constrained optimization with N inequality constraints (note: function and constraints are convex)

Soft-Margin SVM

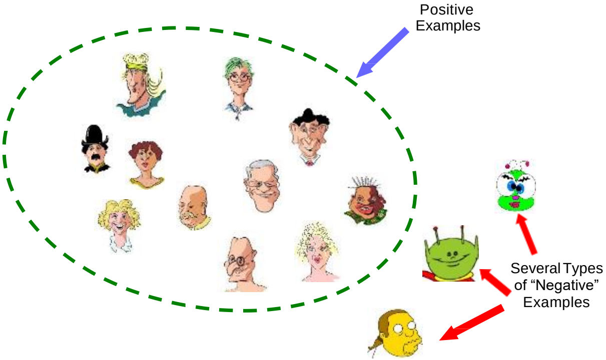
- Allow some training examples to fall **within** the margin region, or be even misclassified (i.e., fall on the wrong side). Preferable if training data is noisy



- Each training example $(\boldsymbol{x}_n, y_n)$ given a "slack" $\xi_n \geq 0$ (distance by which it "violates" the margin). If $\xi_n > 1$ then $\boldsymbol{x}_n$ is totally on the wrong side
  - Basically, we want a soft-margin condition: $y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0$

One-Class Classification

- Can we learn from examples of just one class, say positive examples?
- May be desirable if there are many types of negative examples



- “Outlier/Novelty Detection” problems can also be formulated like this

Figure credit: Refael Chickvashvili

SVM: Some Notes

- A hugely (perhaps the most!) popular classification algorithm
- Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)
  - Some popular ones: libSVM, LIBLINEAR, sklearn also provides SVM
- Lots of work on scaling up SVMs† (both large *N* and large *D*)
- Extensions beyond binary classification (e.g., multiclass, structured outputs)
- Can even be used for regression problems (Support Vector Regression)
- Nonlinear extensions possible via kernels

---

†See: "Support Vector Machine Solvers" by Bottou and Lin