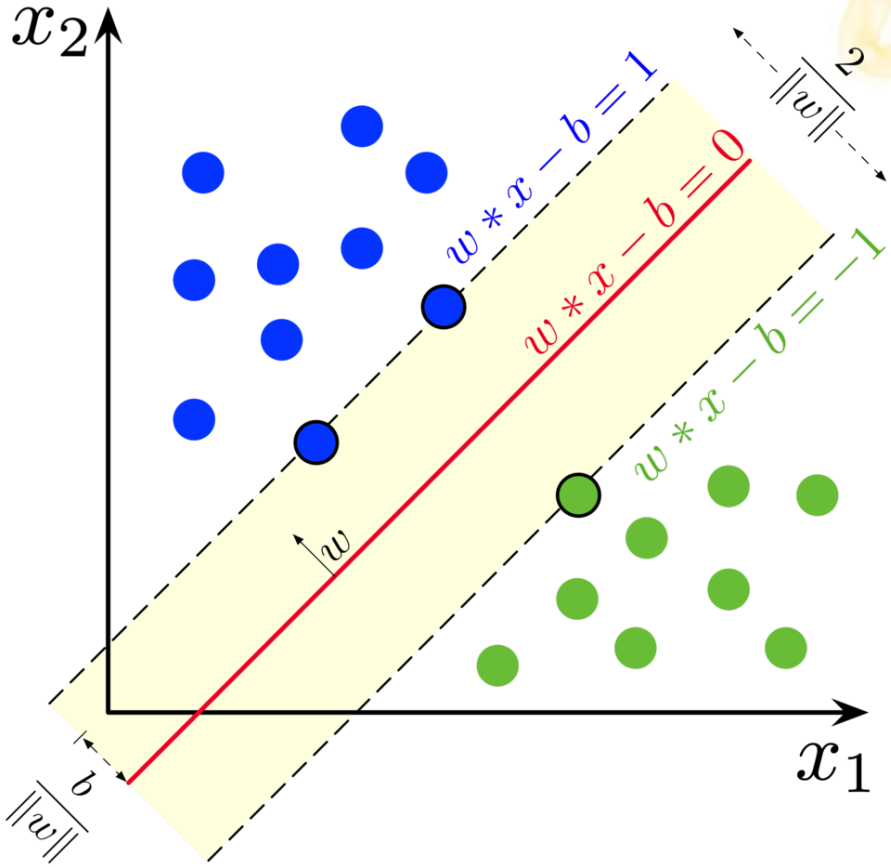
SVM Is a Large Margin Classifier

- Why **large margin** classifier is good?
- Given a linearly separable data set, the optimal separating line maximizes the margin:
  - More robust to noise
  - Large margin reduces VC dimension of hypothesis set



SVM: Notation and Conventions

- Assume that:
  1. Outputs are encoded as $y_i \in \{-1, 1\}$
  2. Pull out w_0 from $\underline{\boldsymbol{w}}$
     - The bias $w_0 = b$ plays a different role
     - $\underline{\boldsymbol{w}} = (w_1, ..., w_d)$ and there is no $x_0 = 1$
     - $\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + b = 0$ is the equation of the separating hyperplane
  3. $\underline{\boldsymbol{x}}_n$ is the closest point to the hyperplane
     - It can be multiple points from different classes
- Normalize $\underline{\boldsymbol{w}}$ and b to get a canonical representation of the hyperplane imposing $|\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_n + b| = 1$

SVM: Original Form of Problem

- The SVM problem is:

$$
\begin{aligned}
&\text{find } \underline{\boldsymbol{w}}, b \\
&\text{maximize } \frac{1}{\|\underline{\boldsymbol{w}}\|} && \text{(max margin)} \\
&\text{subject to } \min_{i=1,\ldots,n} |\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b| = 1 && \text{(hyperplane)}
\end{aligned}
$$

- This problem is not friendly to optimization since it has norm, min, and absolute value

Primal Form of SVM Problem

- You can rewrite it as:

$$\begin{aligned} \text{find } & \underline{\boldsymbol{w}}, b \\ \text{minimize } & \frac{1}{2}\underline{\boldsymbol{w}}^T\underline{\boldsymbol{w}} \\ \text{subject to } & y_i(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b) \geq 1 \; \forall i = 1, ..., n \end{aligned}$$

- Note that under $\underline{\boldsymbol{w}}$ minimal and linear separable classes, it is guaranteed that for at least one $\underline{\boldsymbol{x}}_i$ in the second equation will be equal to 1 (as in the original problem)
  - In fact otherwise we could scale down $\underline{\boldsymbol{w}}$ and b (which does not change the plane) to use the slack, against the hypothesis of minimality of $\underline{\boldsymbol{w}}$

Dual (Lagrangian) Form of SVM Problem

$$\text{minimize with respect to } \underline{\boldsymbol{\alpha}} \qquad \mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \underline{\boldsymbol{x}}_i^T \underline{\boldsymbol{x}}_j$$

$$\text{subject to} \qquad \underline{\boldsymbol{\alpha}} \geq \underline{\mathbf{0}}, \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i$$

- The equation for $\underline{\boldsymbol{w}}$ is not a constraint, but it computes $\underline{\boldsymbol{w}}$ (the plane) given $\underline{\boldsymbol{\alpha}}$, while b is given by $\min|\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b| = 1$

SCIENCE ACADEMY

Dual Form of SVM as QP Problem

- The dual form of SVM problem is a convex quadratic programming problem, in the form:

$$\begin{aligned} \text{minimize with respect to } \underline{\boldsymbol{\alpha}} \quad & \underline{\mathbf{1}}^T \underline{\boldsymbol{\alpha}} - \frac{1}{2} \underline{\boldsymbol{\alpha}}^T \underline{\underline{\boldsymbol{Q}}} \underline{\boldsymbol{\alpha}} \\ \text{subject to} \quad & \underline{\boldsymbol{\alpha}} \geq 0, \underline{\boldsymbol{y}}^T \underline{\boldsymbol{\alpha}} = 0 \end{aligned}$$

where:

- the matrix is $\underline{\underline{\boldsymbol{Q}}} = \{y_i y_j \underline{\boldsymbol{x}}_i^T \underline{\boldsymbol{x}}_j\}_{ij}$
- $\underline{\boldsymbol{\alpha}}$ is the column vector $(\alpha_1, \ldots, \alpha_N)$

Solving Dual Formulation of SVM Problem (1/2)

- **Solving convex problem** for α
  - Feeding this problem to a QP solver, you get the optimal vector $\underline{\boldsymbol{\alpha}}$
- **Compute hyperplane $\underline{\boldsymbol{w}}$**
  - From $\underline{\boldsymbol{\alpha}}$ recover the plane $\underline{\boldsymbol{w}}$ from the equation: $\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i$
  - Looking at the optimal α_i, you can observe that many of them are 0
  - This is because when you applied the Lagrange multipliers to the inequalities: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1$, you got the KKT condition:

$$\alpha_i(y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) - 1) = 0$$

  - From these equations, either
    - $\alpha_i = 0$ and $\underline{\boldsymbol{x}}_i$ is an *interior point* since it has non-null distance from the plane (i.e., slack) from the plane; or
    - $\alpha_i \neq 0$ and the slack is 0, which implies that the $\underline{\boldsymbol{x}}_i$ point touches the margin, i.e., it is a *support vector*

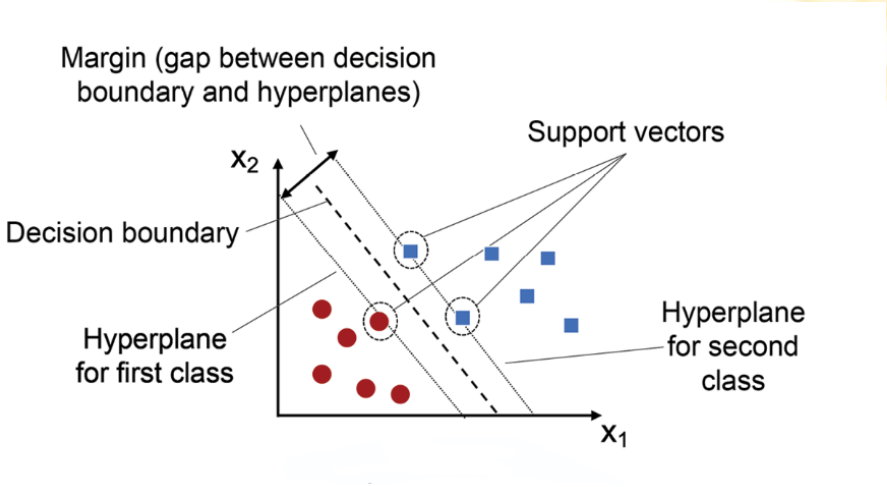
Solving Dual Formulation of SVM Problem (2/2)

- Thus the hyperplane is only function of the support vectors:

$$\underline{\boldsymbol{w}} = \sum_{i=1}^{N} \alpha_i y_i \underline{\boldsymbol{x}}_i = \sum_{\underline{\boldsymbol{x}}_i \in \mathrm{SV}} \alpha_i y_i \underline{\boldsymbol{x}}_i$$

since only for the support vectors $\alpha \neq 0$

  - The $\alpha_i \neq 0$ are the real degree of freedom



- **Compute** b
  - Once $\underline{\boldsymbol{w}}$ is known, you can use any support vector to compute b:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) = 1$$

Support Vectors and Degrees of Freedom for SVM

- The number of support vectors is related to the degrees of freedom of the model
- Because of the VC dimension, you have an in-sample quantity to bound the out-of-sample error:

$$E_{out} \leq E_{in} + c\frac{\text{num of SVs}}{N-1}$$

- You are "guaranteed" to not overfit

Non-Linear Transform for SVM

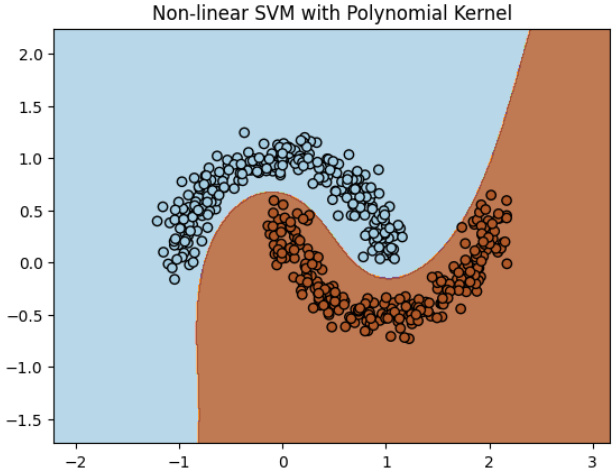
- $\Phi : \mathcal{X} \to \mathcal{Z}$ transforms $\underline{\boldsymbol{x}}_i$ into $\underline{\boldsymbol{z}}_i = \Phi(\underline{\boldsymbol{x}}_i) \in \mathbb{R}^{\tilde{d}}$ with $\tilde{d} > d$
- Transform vectors through Φ and apply SVM machinery
- Dual SVM formulation in $\mathcal{Z}$ space:

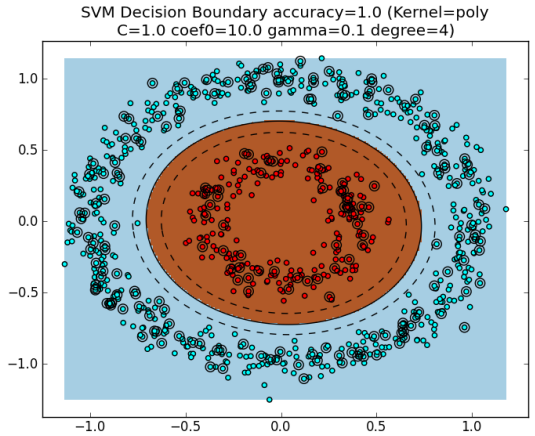
$$\mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \underline{\boldsymbol{z}}_i^T \underline{\boldsymbol{z}}_j$$

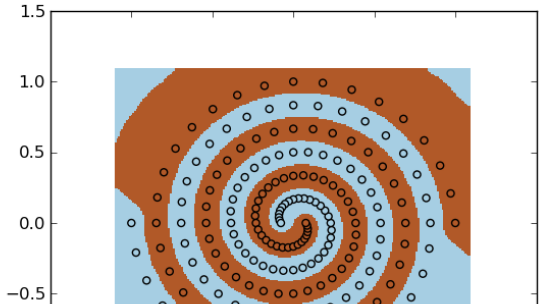
- Note:
  - Optimization problem has same number of unknowns as original space (number of points N)
  - Support vectors live in $\mathcal{Z}$: they have $\alpha = 0$. In $\mathcal{X}$, they are pre-images of support vectors
  - Decision boundary and margin can be represented in original space (not linear)

Non-Linear Transforms for SVM vs Others

- In SVM the non-linear transform does not change the number of unknowns and degrees of freedom of the model
- This is different from transforming the variables in a linear problem, since in that case the number of unknowns changes







SVM in Higher Dimensional Space

- **Pros**
  - You don't pay the price in terms of complexity of optimization problem
    - Number of unknowns is still N (different than a linear problem)
  - You don't pay the price in terms of increased generalization bounds
    - Number of support vectors is $\leq N$
    - This is because each hypothesis h can be complex but the cardinality of the hypothesis set $\mathcal{H}$ is the same
- **Cons**
  - You pay a price to compute $\Phi(\underline{\boldsymbol{x}}_i)^T\Phi(\underline{\boldsymbol{x}}_j)$, since Φ could be very complex
    - The kernel trick will remove this extra complexity by doing $\Phi(\underline{\boldsymbol{x}}_i)^T\Phi(\underline{\boldsymbol{x}}_j) = K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_j)$

SCIENCE ACADEMY

Non-Linear Transform in SVM vs Kernel Trick

- The trivial approach is:
  - Transform vectors with $\Phi(\cdot)$
  - Apply all SVM machinery to the transformed vectors
  - **Cons**: Φ might be very complex, e.g., potentially exponential number of terms
- You can express the SVM problem formulation and the prediction in terms of a kernel

$$K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}') = \Phi(\underline{\boldsymbol{x}})^T \Phi(\underline{\boldsymbol{x}}') = \underline{\boldsymbol{z}}^T \underline{\boldsymbol{z}}'$$

  - You only need the kernel $K_\Phi(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}')$ of the transformation $\Phi(\cdot)$ and not $\Phi(\cdot)$ itself

SCIENCE ACADEMY

SVM in Terms of Kernel: Optimization Step

- When you build the QP formulation for the Lagrangian to compute the α we can use $K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_j)$ instead of $\underline{\boldsymbol{z}}_i^T \underline{\boldsymbol{z}}_j$

$$\mathcal{L}(\underline{\boldsymbol{\alpha}}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m K_\Phi(\underline{\boldsymbol{x}}_n, \underline{\boldsymbol{x}}_m)$$

- $\underline{\boldsymbol{z}}_n$ does not appear in the constraints

$$\underline{\boldsymbol{\alpha}} \geq \underline{\mathbf{0}}, \underline{\boldsymbol{\alpha}}^T \underline{\boldsymbol{y}} = 0$$

SVM in Terms of Kernel: Prediction Step

- You need only inner products to compute a prediction for a given $\underline{\boldsymbol{z}}$
- In fact to make predictions, you replace the expression of $\underline{\tilde{\boldsymbol{w}}} = \sum_{i:\alpha_i>0} \alpha_i y_i \underline{\boldsymbol{z}}_i$ in $h(\underline{\boldsymbol{x}}) = \text{sign}(\underline{\boldsymbol{w}}^T \Phi(\underline{\boldsymbol{x}}) + b)$, yielding:

$$h(\underline{\boldsymbol{x}}) = \text{sign}\Big(\sum_{i:\alpha_i>0} \alpha_i y_i K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}) + b\Big)$$

where b is given by $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{z}}_i + b) = 1$ for any support vector $\underline{\boldsymbol{x}}_m$ and thus

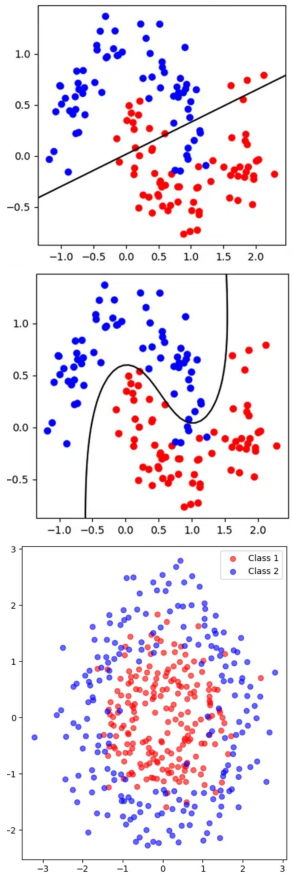
$$b = \frac{1}{y_m} - \sum_{i:\alpha_i>0} \alpha_i y_i K_\Phi(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}_m)$$

Implications of Kernel Trick in SVM

- The "kernel trick" is a computational shortcut:
  - Use the kernel of the transformation instead of the transformation itself
- To use SVMs, compute inner products between transformed vectors $\underline{\boldsymbol{z}}$
- The kernel trick implies:
  - No need to compute $\Phi()$
    - Use the kernel K_Φ, not the transformation Φ
  - No need to know Φ
    - With function K_Φ as an inner product, use SVM machinery without knowing $\mathcal{Z}$ space or transformation Φ
  - Φ can be impossible to compute
    - K_Φ can correspond to a transformation Φ to an infinite dimensional space (e.g., Gaussian kernel)

Non-Linearly Separable SVM Problem

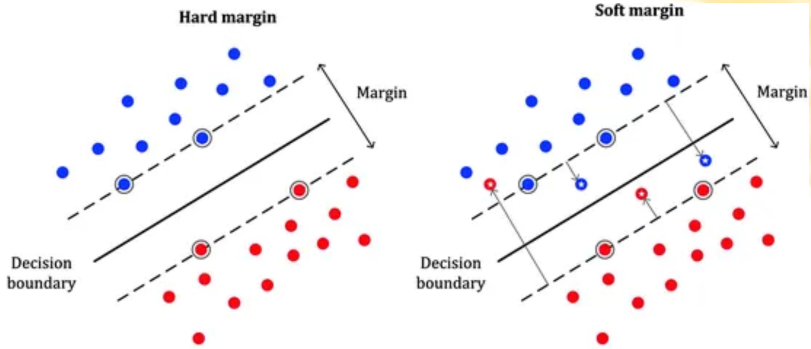
- In general there are different types of **non-linearly separable data sets**
- **Slightly non-separable**
  - Few points crossing the boundary
  - $\implies$ use soft margin SVMs
- **Seriously non-separable**
  - E.g., the class inside the circle
  - $\implies$ use non-linear kernels

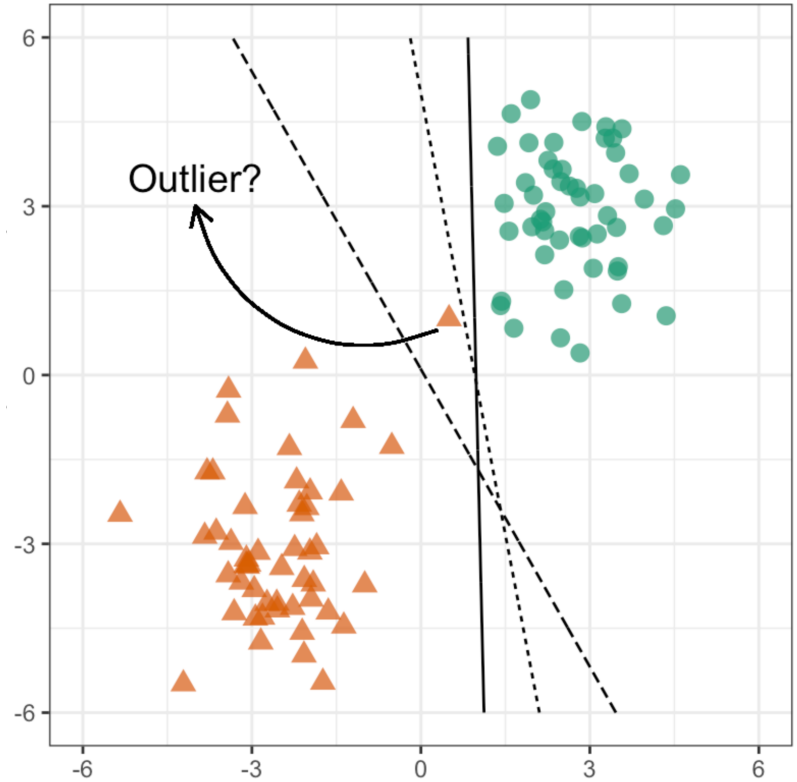


- In practice, both issues are present
  - Combine soft margin SVM and non-linear kernel transforms

Soft-Margin SVM: Advantages

- Even with linearly separable data, improve E_{out} using soft margin SVM at the cost of worse E_{in}
  - Trade-off between in-sample and out-of-sample performance
- E.g., outliers force a smaller margin
  - Ignoring outliers could increase margin
- Large C parameter in SVM requires minimizing error, trading off large margin for correct classification





Primal Formulation for Soft Margin SVM

- You want to introduce an error measure based on the margin violation for each point, so instead of the constraint:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 \text{ (hard margin)}$$

- You can use:

$$y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 - \xi_i, \text{ where } \xi_i \geq 0 \text{ (soft margin)}$$

- The cumulative margin violation is $C \sum_{i=1}^{N} \xi_i$
- The soft margin SVM optimization (primal form) is:

$$\begin{aligned} &\text{find } \underline{\boldsymbol{w}}, b, \underline{\boldsymbol{\xi}} \\ &\text{minimize} \quad \frac{1}{2}\underline{\boldsymbol{w}}^T \underline{\boldsymbol{w}} + C \sum_{i=1}^{N} \xi_i \\ &\text{subject to} \quad y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) \geq 1 - \xi_i \ \forall i \\ &\qquad\qquad\quad \xi_i \geq 0 \end{aligned}$$

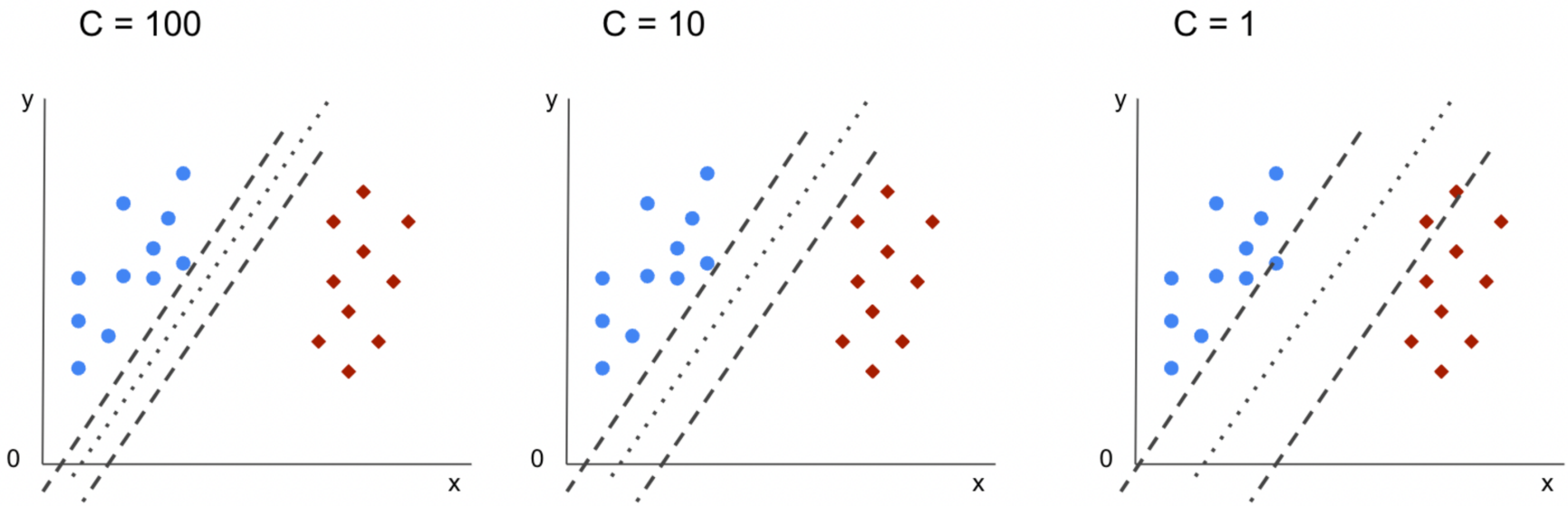
SCIENCE ACADEMY

Classes of Support Vectors for Soft Margin SVM

- There are 3 classes of points:
  - **Margin support vectors**: they are exactly on the margin and define it
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) = 1 \iff \xi_i = 0$
    - In dual form: $0 < \alpha_i < C$
  - **Non-margin support vectors**: they are inside the margin and classified correctly or not
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) < 1 \iff \xi_i > 0$
    - In dual form: $\alpha_i = C$
  - **Non-support vectors**, i.e., interior points:
    - In primal form: $y_i(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}}_i + b) > 1$
    - In dual form: $\alpha_i = 0$

Intuition for C in SVM

- C represents how much penalty you incur for passing the margin
  - If C is large, then SVM will try to fit all the points to avoid being penalized
    - Lower bias / higher variance
    - $C \to \infty$ which yields hard-margin SVM
  - If C is small, then you allow margin violations
    - Higher bias / lower variance
- From another point of view $C \propto \frac{1}{\lambda}$, so large C means small λ and thus small regularization
  - C is chosen through cross validation, like any regularization parameter



Multi-Class Classification for SVM

- Often SVM packages have built-in multi-class classification
- Otherwise use the one-vs-all method:
  - Train K SVMs distinguishing each class from the rest using one-hot encoding
    - Get SVM parameters $(\underline{\boldsymbol{w}}_1, b_1), ..., (\underline{\boldsymbol{w}}_K, b_k)$
  - For a new example $\underline{\boldsymbol{x}}$ compute $\underline{\boldsymbol{w}}_i^T \underline{\boldsymbol{x}} + b_i$ for all the models
  - Pick the model that gives the largest positive value
    - I.e., more confident about its class vs the rest of the classes

- ***Similarity-Based Models***
- Clustering
- Anomaly Detection

Similarity-Based Models: Intuition

- **Idea**: the model evaluated in one point $h(\underline{\boldsymbol{x}})$ is affected by:
  - Other data points in the training set $(\underline{\boldsymbol{x}}_n, y_n) \in \mathcal{D}$
  - The effect is based on the distance $d(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_n) = \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_n\|$
- The model is a **superposition of effects**
  - Sum of the effect of each point in the training set, scaled down by the distance

$$h(\underline{\boldsymbol{x}}) = \sum_i \text{ effect of } h(\underline{\boldsymbol{x}}_i) \text{ scaled by } d(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i)$$

- This approach allows to define complex decision boundaries

Similarity-Based Models: Gaussian Kernels

- Consider a Gaussian kernel with a "landmark" point $\underline{\boldsymbol{x}}_i$ and a similarity distance defined as:

$$K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i) = \exp(-\frac{\|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}{2\sigma^2})$$

- E.g., the hypothesis model has the form:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{3} y_i K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_i) = y_1 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_1) + y_2 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_2) + y_3 K(\underline{\boldsymbol{x}}, \underline{\boldsymbol{x}}_3)$$

  - The response is weighting the responses $y_i = \{0, 1\}$ through the similarity of $\underline{\boldsymbol{x}}$ from the landmark points
  - This can be seen by plotting $h(\underline{\boldsymbol{x}})$ on a plane

Radial Basis Function Model for Regression

- Aka RBF
- The model form for **regression** is:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{N} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2)$$

where:

- If γ is small $\implies$ the exponential falls off slowly, and multiple training points affect a point between them
- If γ is large $\implies$ there are spikes centered in the training points and nothing outside

Radial Basis Function Model for Classification

- For **classification** use a similar approach to "linear regression for classification"
  - Fit a regression model:

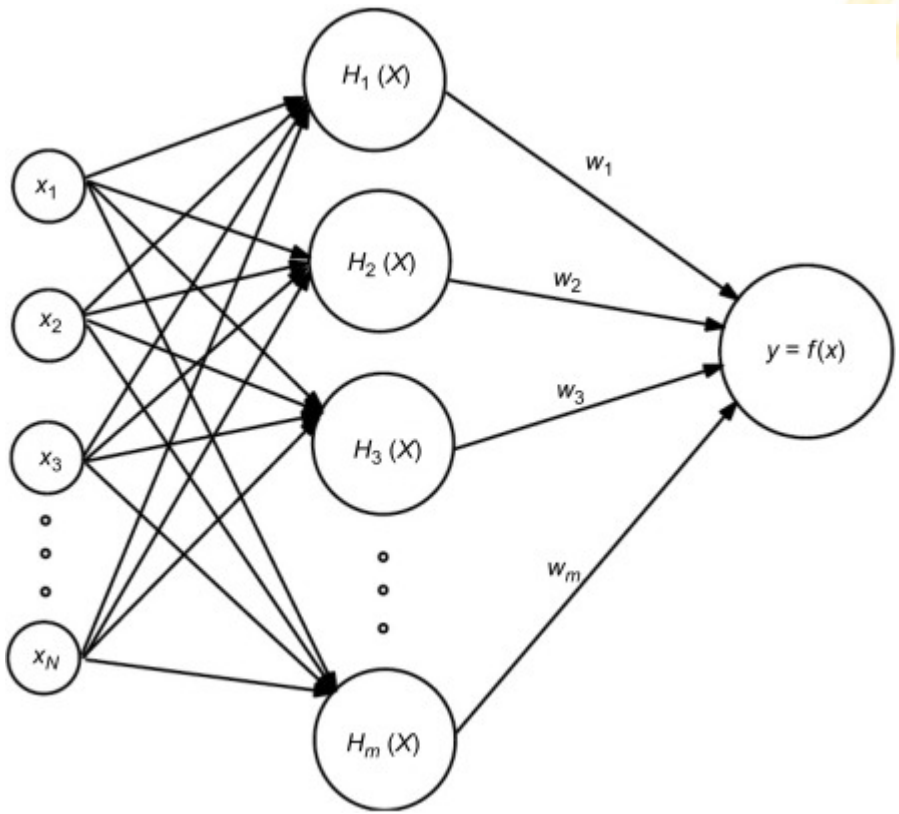
$$s(\underline{\boldsymbol{x}}) = \sum_{i=1}^{N} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2)$$

  - Take the sign to make predictions:

$$h(\underline{\boldsymbol{x}}) = \text{sign}(s(\underline{\boldsymbol{x}}))$$

RBF: Block Diagram

- The params(fixed by learning) are one for each training point
  - The weights depending on the distance of the input to the examples
  - The weighted params are summed together



RBF: Number of Parameters

- Some variants for RBF:
  - Add bias term
  - Use different γ_i for each point
  - Increase degrees of freedom
- RBF has lots of parameters
  - Parameters $\underline{\boldsymbol{w}}$ equal data points N (e.g., $N = 10^9$)
  - One parameter w_i per training point (e.g., $N = 10^6$)
  - **Cons**: Negative impact on generalization error

RBF: Reducing Model VC Dimension

- **To reduce number of parameters**
  - Pick $K \ll N$ centers $\underline{\boldsymbol{\mu}}_1, \ldots, \underline{\boldsymbol{\mu}}_K$ instead of $\underline{\boldsymbol{x}}_1, \ldots, \underline{\boldsymbol{x}}_N$
    - Use k-means clustering to find centers
    - Unsupervised learning; doesn't use labels
  - Same as RBF model using distances from cluster centers:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{K} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}}_i\|^2)$$

- **Still many parameters** because:
  - K (scalar) weights w_k
  - K reference points $\underline{\boldsymbol{\mu}}_k$ (d-dimensional vectors)

RBF: Learning Models (1/2)

- **Learn** w_i, γ, with fixed centers $\underline{\boldsymbol{\mu}}_i$, for an RBF model:

$$h(\underline{\boldsymbol{x}}) = \sum_{i=1}^{K} w_i \exp(-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{\mu}}_i\|^2)$$

- **Minimize:**

$$E_{in} = \sum_i (h_{\underline{\boldsymbol{w}},\gamma}(\underline{\boldsymbol{x}}_i) - y_i)^2 = f(\underline{\boldsymbol{w}}, \gamma)$$

  - Use an iterative approach (e.g., EM, coordinate descent)

RBF: Learning Models (2/2)

- Use iterative approach (similar to EM algorithm):
  - Fix γ, solve for $\underline{\boldsymbol{w}}$ (one-step learning)
  - Fix $\underline{\boldsymbol{w}}$, solve for γ (gradient descent)
- **Step 1**
  - Assume γ is known and fixed
  - Learn $\underline{\boldsymbol{w}}$
- Impose perfect interpolation:

$$E_{in} = \frac{1}{n}\sum(h(\underline{\boldsymbol{x}}_i) - y_i)^2 = 0$$

- **Problem**:

$$h(\underline{\boldsymbol{x}}_j) = \sum_i w_i \exp(-\gamma\|\underline{\boldsymbol{x}}_i - \underline{\boldsymbol{x}}_j\|^2) = \sum_i w_i \phi_{i,j} = \underline{\phi}_j^T \underline{\boldsymbol{w}} = y_i$$

  - N equations (one per point) and N unknowns $\underline{\boldsymbol{w}}$
  - $\underline{\underline{\boldsymbol{\Phi}}}$ is known, function of data set and γ

SCIENCE ACADEMY

Learning RBF Models

- The problem in matrix form is:

$$\underline{\underline{\boldsymbol{\Phi}}} \cdot \underline{\boldsymbol{w}} = \underline{\boldsymbol{y}}$$

  - If $\underline{\underline{\boldsymbol{\Phi}}}$ is invertible, then $\underline{\boldsymbol{w}} = \underline{\underline{\boldsymbol{\Phi}}}^{-1}\underline{\boldsymbol{y}}$
    - Desired values on training points
    - Exponential interpolates other points
  - If $\underline{\underline{\boldsymbol{\Phi}}}$ is not invertible, optimize in least square sense:

$$\text{argmin}_{\underline{\boldsymbol{w}}} E_{in} = \sum_i (h(\underline{\boldsymbol{x}}_j) - y_i)^2$$

    - Compute pseudo-inverse (assuming $\underline{\underline{\boldsymbol{\Phi}}}^T\underline{\underline{\boldsymbol{\Phi}}}$ is invertible)
    - Assign weights:

$$\underline{\boldsymbol{w}} = (\underline{\underline{\boldsymbol{\Phi}}}^T\underline{\underline{\boldsymbol{\Phi}}})^{-1}\underline{\underline{\boldsymbol{\Phi}}}^T\underline{\boldsymbol{y}}$$

- **Step 2**
  - Assume $\underline{\boldsymbol{w}}$ is known and fixed
  - Learn γ

RBF Network vs Neural Networks

- The regression model for Neural Networks and RBF model is **similar**:
  - **RBF**:
$$h(\underline{\boldsymbol{x}}) = \sum_i w_i e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2} = \underline{\boldsymbol{w}}^T \underline{\underline{\phi}}$$
  - **Neural networks**:
$$h(\underline{\boldsymbol{x}}) = \Theta(\underline{\boldsymbol{w}}^{(L)T} \underline{\boldsymbol{x}}^{(L)}) = \Theta(\underline{\boldsymbol{w}}^{(L)T} \underline{\boldsymbol{\Theta}}(\underline{\underline{\boldsymbol{W}}}^{(L-1)} \ldots))$$
- **Difference**:
  - RBF has a single layer
  - Neural networks have multiple layers
- **Similarities**:
  - Combine features with weights using dot product
  - Extract features from inputs
    - RBF features: $e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}$, always > 0
    - NN hidden layers: features can be > 0 or < 0

RBF Network vs SVM

- The model form is the same:
  - RBF:
$$h(\underline{\boldsymbol{x}}) = \text{sign}(\sum_i w_i e^{-\gamma \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_i\|^2}$$
  - SVM:
$$h(\underline{\boldsymbol{x}}) = \text{sign}(\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + b)$$
- The interpretation is completely different (interpolation vs large margin)
  - RBF: all vectors (or centers of few clusters) contribute to the model
  - SVM: only support vectors contribute to the model

K-Nearest Neighbor (KNN) Model

- The model is like:

$$h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}) = \frac{1}{n} \sum_{\underline{\boldsymbol{x}}_i \text{closest to } \underline{\boldsymbol{x}}} w_i$$

- **Idea**:
  - Closeness implies a distance (e.g., euclidean metric) or similarity (e.g., a kernel)
  - Consider the k closest points to the evaluation point $\underline{\boldsymbol{x}}$
  - Take an average of their response

KNN: Intuition of Number Degrees of Freedom

- Nearest neighbor model ($k = 1$)
  - Adopt the response of the closest point to the point we are evaluating $\underline{\boldsymbol{x}}$
  - Like Voronoi tessellations: each point has a region for which it is the closest point and assigns its output to that region
- One could think there is a single parameter for KNN, i.e., k, since this is the hyperparameter we need to learn
  - For $k = 1$ there are N neighborhoods, one around each point of the training set
  - For $k = N$ there is a single neighborhood
  - So the intuition is that the effective number of parameters is $\frac{N}{k}$ since one can imagine there are N/k non-overlapping neighborhoods

KNN: Assumptions on the Data

- KNN makes no assumption on the data
  - This is the opposite of the linear model where there is a strong assumption on the data
- KNN assumes locality in parameter space
  - The model is constant in the neighborhood of an example
  - E.g., $k = 1$ we consider the Voronoi tesselation low-bias / high-variance
  - E.g., $k = N$ we consider the average value (high-bias / low-variance)

Training and Test Error for KNN

- For $k = 1$ a KNN model
  - Makes no error on the training set (assuming a non-noisy target), since it memorizes the training set (low bias / high variance)
  - E_{out} is larger than E_{in}
- If k increases the training error E_{in} increases but the test error E_{out} decreases until it starts increasing again
  - We have the typical behavior of model complexity as in bias-variance diagrams

KNN vs RBF Models

- **Similarities**
  - K-Nearest Neighbor is a *discrete* version of the RBF model
- **Differences:**
  - Consider only the k *closest examples* to the point $\underline{\boldsymbol{x}}$ (not all examples in the training set)
  - Use a *constant kernel* (responses are not weighted by distance)

- Similarity-Based Models
- ***Clustering***
- Anomaly Detection

K-Means Clustering: Problem Formulation

- We have N *unlabeled* points $\{\underline{\boldsymbol{x}}_1, \underline{\boldsymbol{x}}_2, ..., \underline{\boldsymbol{x}}_N\}$
- We want to partition the points into K clusters $S_1, ..., S_K$
  - Each cluster is defined by its center $\underline{\boldsymbol{\mu}}_k$
  - Each point $\underline{\boldsymbol{x}}_i$ is assigned to cluster $c(\underline{\boldsymbol{x}}_i)$
  - The unknowns are: $c(\underline{\boldsymbol{x}}_1), ..., c(\underline{\boldsymbol{x}}_N), \underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$
- We want to minimize the distance between each $\underline{\boldsymbol{x}}_i$ and the assigned center $\underline{\boldsymbol{\mu}}_k$ where $k = c(\underline{\boldsymbol{x}}_i)$:

$$J(c_1, ..., c_N, \mu_1, ..., \mu_K) = \sum_{k=1}^{K} \sum_{\underline{\boldsymbol{x}}_n \in S_k} \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_k\|^2 \text{(scanning the clusters)}$$

$$= \sum_{i=1}^{N} \|\underline{\boldsymbol{x}}_i - \underline{\boldsymbol{\mu}}_{c(\underline{\boldsymbol{x}}_i)}\|^2 \qquad \text{(scanning the points)}$$

- K-means clustering is NP-hard (combinatorial) and thus intractable
  - In fact there are K^N possible assignments

K-Means Clustering: Lloyd'S Algorithm

- We start picking a random assignment of N points to the K clusters
  - Better than picking randomly the centroids of the clusters
- Each iteration does 2 steps
- **Step 1**: Move centroid
  - We move the centroid of each cluster to the mean point of the current cluster
  - This loop iterates over the K clusters
  - $\underline{\boldsymbol{\mu}}_k \leftarrow \frac{1}{|S_k|} \sum_{\underline{\boldsymbol{x}}_n \in S_k} \underline{\boldsymbol{x}}_n$
- **Step 2**: Cluster assignment
  - Each $\underline{\boldsymbol{x}}_n$ is assigned to the closest cluster based on its center
  - This loop iterates over the N points
  - $S_k \leftarrow \{\underline{\boldsymbol{x}}_n : \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_k\| \leq \|\underline{\boldsymbol{x}}_n - \underline{\boldsymbol{\mu}}_l\| \; \forall l \neq k\}$

K-Means Clustering: Convergence

- K-means algorithm converges since:
  - There is a finite (though large K^N) number of possible partitionings, and thus a finite number of possible values of the objective functions
  - The objective function $J(\cdot)$ is always decreased
- The objective function is always decreasing
  - The cost function $J(\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K, c_1, ..., c_N)$ can be seen as a function of:
    - The centroids $c_1, .., c_N$
    - The point assignments $\underline{\boldsymbol{\mu}}_1, ..., \underline{\boldsymbol{\mu}}_K$
  - At each step, K-means minimizes J with respect to:
    - The centroids (keeping the assignments fixed); then
    - The assignments (keeping the centroids fixed)
  - It is like coordinate descent
- Generally, it converges to a local minimum
  - Run K-means multiple times using different random initializations
  - Pick the best result

K-Means Clustering: Non-Separable Clusters

- For simplicity, we imagine a clear separation between clusters
  - In practice, clusters (especially in high dimensions) are not obviously separable
- We can use K-means on data that is not obviously separated
  - E.g., market segmentation
  - E.g., t-shirt sizing
    - Collect height and width of a population of customers
    - Run K-means
    - Find the optimal way to split the population into 3 sizes (S, M, L)

Choosing the Number of Clusters

- It's often unclear how many clusters K exist in the data
  - E.g., visual analysis can be inconclusive with 2D or 3D data
  - Even more difficult in high dimensional spaces

1. Elbow Method
   - Vary the number of clusters K
   - Compute the optimal cost function $J(\cdot)$
   - Choose K at the "elbow" point if visible
   - The elbow is absent if the curve resembles a hyperbole $\approx 1/K$
2. End-to-end approach
   - Choose K to optimize later processing stages
   - E.g., More t-shirt sizes (i.e., more clusters) $\implies$
     - Satisfy customers
     - Complicates manufacturing
     - Increases stocking and inventory management

Clustering: Interpretation of Clusters

- Often we want to give a meaning to clusters
- Cluster meaning is difficult to automate: it must be interpreted manually
  - Examine the cluster centroids
    - Centroid values show the "typical" point in each cluster
    - High, low, or zero feature values highlight key characteristics
  - Analyze the distribution of features per cluster
    - Plot histograms or boxplots for each feature
    - Identify features that vary sharply across clusters
  - Visualize clusters in 2D or 3D
    - E.g., PCA, t-SNE, UMAP
    - Helps understand separation and internal structure
  - Identify common traits in each cluster
    - For categorical features, count dominant categories
  - Compare clusters to external labels if available
    - See if clusters align with known real-world groups
  - Train a classifier like decision tree
    - Important features for predicting cluster reveal their meaning
- Example: Customer Segmentation
  - Features: (Age, Annual Income, Spending Score)
  - Cluster 1
    - (25 yrs, 30K, 90) → "Young Big Spenders"

- Similarity-Based Models
- Clustering
- ***Anomaly Detection***

Anomaly Detection: Problem Formulation

- **Problem**:
  - We have $\{\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N\}$ examples with features $\underline{\boldsymbol{x}} \in \mathbb{R}^P$ for good / non-anomalous instances
  - We want to find a way to detect bad / anomalous instances
- **Algorithm**:
  - We don't know what makes a *"bad instances"*
  - We learn what *"good instances"* have in common using unsupervised learning
    - I.e., find the distribution for *"good instances"* $\underline{\boldsymbol{x}}_i$, $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Pick features
    - The goal is to find "sensitive" features, i.e., features that might take large or small values in case of an anomaly
    - E.g., ratio between CPU load and network traffic
  - Estimate the distribution $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Choose the threshold ε
  - For a new instance $\underline{\boldsymbol{x}}_{new}$, if

$$\Pr(\underline{\boldsymbol{x}}_{new} \text{ is good}) \leq \varepsilon$$

  we flag it as an anomaly

SCIENCE ACADEMY

Anomaly Detection: Example of Aircraft Engines

- **Problem**
  - Test aircraft engines to identify anomalies in a new engine
- **Solution**:
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - Heat generated
    - Vibration intensity
    - ...
  - Collect data for all engines
  - Model a PDF $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Decide if a new engine is acceptable $\Pr(\underline{\boldsymbol{x}}_{good}) \leq \varepsilon$ or needs more testing

SCIENCE ACADEMY

Anomaly Detection: Example of Hacked Account

- **Problem**
  - Find if an account for a given user i was hacked
- **Solution**:
  - Model features that represent "user i activity"
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - How many times s/he logs a day
    - How many times s/he fails to enter the password
    - How fast s/he types
    - How many pages s/he visits
    - How many times s/he posts comments
    - ...
  - Model a PDF $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Identify unusual users by checking $\Pr(\underline{\boldsymbol{x}}_{new}) \leq \varepsilon$

SCIENCE ACADEMY

Anomaly Detection: Example of Computers in Data Center

- **Problem**
  - Monitor servers in a data center to find malfunctioning or hanged servers
- **Solution**:
  - Features $\underline{\boldsymbol{x}}_i$ can be:
    - Memory in use
    - CPU load
    - Network traffic
    - Number of reads/writes per sec
    - CPU load / network activity
    - ...
  - Model a PDF $\Pr(\underline{\boldsymbol{x}}$ is good$)$
  - Identify unusual users by checking $\Pr(\underline{\boldsymbol{x}}_{new}) \leq \varepsilon$

SCIENCE ACADEMY

Using a Gaussian Model for Anomaly Detection

- Aka "density estimation"
- Given N examples $\underline{\boldsymbol{x}}_1, ..., \underline{\boldsymbol{x}}_N \in \mathbb{R}^p$
- Ensure that the features have a Gaussian distribution
  - If not, we can apply some transformations, e.g., $\log(x_i + k)$
- Estimate the parameters of the Gaussian model $f_X(\underline{\boldsymbol{x}})$
- Given a new example $\underline{\boldsymbol{x}}_{new}$, compute:

$$\Pr(\underline{\boldsymbol{x}}_{new} \text{ is good}) \leq \varepsilon$$

to flag an anomaly