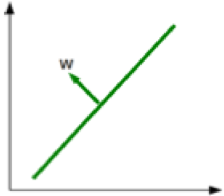
Suport Vector Machines

Acknowledgments: Piyush Rai

June 23, 2016

A Note on Hyperplanes

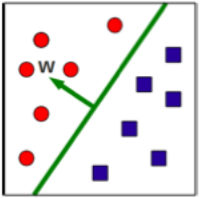
- Separates an n-dimensional space into two half-spaces.



- Defined by an outward pointing normal vector $\mathbf{w} \in \mathbb{R}^n$
- $\mathbf{w}$ is orthogonal to any vector lying on the hyperplane
- Assumption: The hyperplane passes through origin. If not,
  - have a bias term b; we will then need both $\mathbf{w}$ and b to define it

Linear Classification via Hyperplanes

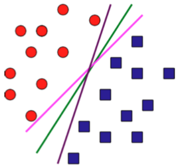
- Linear Classifiers: Represent the decision boundary by a hyperplane **w**



- For binary classification, **w** is assumed to point *towards* the positive class
- Classification rule:
  $y = sgn(\mathbf{w}^T\mathbf{x} + b) = sgn(\sum_{j=1}^{n} w_j x_j + b)$
  - $\mathbf{w}^T\mathbf{x} + b > 0 \Rightarrow y = +1$
  - $\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow y = -1$
- **Goal:** To learn the hyperplane $(\mathbf{w}, b)$ using the training data
  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$.

Linear Classification via Hyperplanes

- Assume that the training data set is linearly separable
- Hence, there exist parameters **w** and b s.t.
  - $\mathbf{w}^T\mathbf{x_i} + b > 0$ for points having $y_i = +1$
  - $\mathbf{w}^T\mathbf{x_i} + b < 0$ for points having $y_i = -1$
  - or equivalently, $y_i(\mathbf{w}^T\mathbf{x_i} + b) > 0$ for all training data points.
- Of the many possible choices, which one is the best?



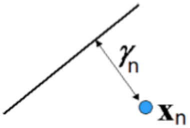
  - Intuitively, we want the hyperplane having the maximum margin (where the margin is defined as the smallest distance between the decision boundary and any of the samples)

The Concept of Margin

- The perpendicular distance of a point $\mathbf{x}$ from a hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ is given by

$$\gamma = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}$$

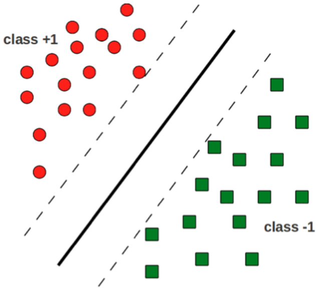
- Margin is given by the perpendicular distance to the closest point $\mathbf{x}_n$ from the data.



- Support Vector Machine finds the hyperplane with the maximum margin.

Support Vector Machine (SVM)

- Probably the most popular/influential classification algorithm
- Backed by solid theoretical groundings (Vapnik and Cortes, 1995)
- A hyperplane based classifier
- Uses the Maximum Margin Principle
  - Finds the hyperplane with maximum separation margin on the training data



Support Vector Machine

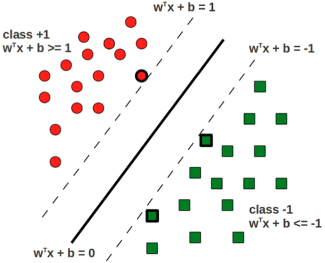
- **Goal:** Find the hyperplane with maximum separation margin on the training
- Interested in solutions for which all data points are correctly classified, s.t. $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$ for all $i = 1, \cdots, N$.
- We wish to optimize the parameters $\mathbf{w}$ and b in order to maximize the margin.
- The maximum margin solution is found by solving:

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_i\left[y_i(\mathbf{w}^T\mathbf{x}_i + b)\right]\right\}$$

- Direct solution - very complex $\Rightarrow$ rescaling

Support Vector Machine

- Assume the hyperplane is such that
  - $\mathbf{w}^T\mathbf{x}_i + b \geq 1$ for $y_i = +1$
  - $\mathbf{w}^T\mathbf{x}_i + b \leq -1$ for $y_i = -1$



- For the point that is closest to the decision surface, set:

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

- All other points satisfy the constraints:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, i = 1, \cdots, N$$

Support Vector Machine: The Optimization Problem

- Hence,

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_i\left[y_i(\mathbf{w}^T\mathbf{x}_i+b)\right]\right\} \text{ becomes } \arg\max_{\mathbf{w},b}\frac{1}{\|\mathbf{w}\|}$$

which is equivalent to the optimization problem:

$$\arg\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2$$

subject to

$$y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1, i=1,\cdots,N$$

- A *quadratic programming* problem - minimizing a quadratic function subject to a set of linear inequality constraints.

Large Margin = Good Generalization

- Large margins intuitively mean good generalization
- Recall: Margin $\gamma = \frac{1}{||\mathbf{w}||}$
- Large margin $\Rightarrow$ small $||\mathbf{w}||$
- Small $||\mathbf{w}||$ $\Rightarrow$ regularized/simple solutions (w_i's don't become too large)
- Simple solutions $\Rightarrow$ good generalization on test data

Solving the SVM Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{\|\mathbf{w}\|^2}{2}$$

$$\text{subject to} \quad 1 \leq y_i(\mathbf{w}^T \mathbf{x}_i + b), i = 1, \cdots, N$$

- Introducing Lagrange Multipliers α_i ($i = \{1, \cdots, N\}$), one for each constraint, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|^2}{2} + \sum_{i=1}^{N} \alpha_i \{1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)\}$$

$$\text{subject to} \quad \alpha_i \geq 0, i = 1, \cdots, N$$

- We can now solve this Lagrangian
  - i.e., optimize $L_P(\mathbf{w}, b, \alpha)$ w.r.t. $\mathbf{w}$, b, and α
  - Making use of the Lagrangian Duality theory.

Solving the SVM Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \; \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0$$

- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

Maximize $L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$

subject to $\sum_{i=1}^{N} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, \cdots, N$

- This is a Quadratic Programming problem in α
  - Several off-the-shelf solvers exist to solve such QPs

Dual vs. Primal

- Generally, the computational complexity of a quadratic programming problem in n variables is $O(n^3)$.
- Going from Primal to Dual: n variables vs N variables.
  - If n (dimension) is smaller than N (number of data points), the move to the dual problem appears disadvantageous.
- However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied *efficiently* to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces.

Support Vector Machine: Prediction

- Prediction rule: $y = sgn(\mathbf{w}^T\mathbf{x} + b)$
- Substituting

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

  - Prediction rule becomes:

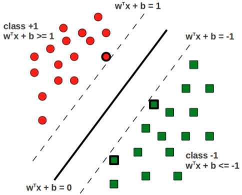
$$y = sgn\left(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}^T \mathbf{x}_i + b\right)$$

- What is b?

Support Vector Machine: Sparse solution

- Most α_i's in the solution are zero (sparse solution)
  - Reason: SVM constrained optimization satisfies Karush-Kuhn-Tucker (KKT) conditions:
    $$\alpha_i \geq 0$$
    $$y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0$$
    $$\alpha_i\{y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\} = 0$$
    Thus for every data point, either $\alpha_i = 0$ or $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$.
  - α_i is non-zero only if $\mathbf{x}_i$ lies on one of the two margin boundaries, i.e., for which $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$.

  - These examples are called support vectors
  - Support vectors “support" the margin boundaries



- Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Solving for b

- Note that any support vector $\mathbf{x}_i$ satisfies $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$, or equivalently

$$y_i\left(\sum_{j\in S}\alpha_j y_j \mathbf{x}_i^T\mathbf{x}_j + b\right) = 1$$

where S denotes the set of indices of the support vectors.

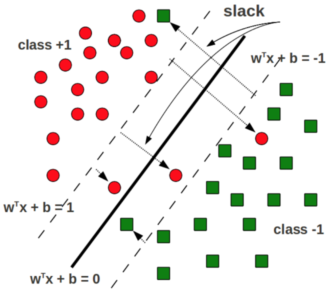
- We can solve this eq. for b using an arbitrarily chosen support vector $\mathbf{x}_i$.
- However, a numerically more stable solution is obtained by first multiplying by y_i, making use of $y_i^2 = 1$, and then averaging these equations over all support vectors and solving for b.
- Solving for b, we obtain:

$$b = \frac{1}{N_S}\sum_{i\in S}\left(y_i - \sum_{j\in S}\alpha_j y_j \mathbf{x}_i^T\mathbf{x}_j\right)$$

where N_S is the total number of support vectors.

SVM - Non-Separable Case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region



SVM - Non-Separable Case

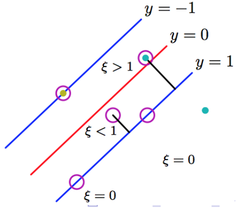
- Recall: For the separable case (training loss = 0), the constraints were:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \ \forall i$$

- For the non-separable case, we relax the above constraints:
  - Data points are allowed to be on the "wrong side" of the margin, but with a penalty that increases with the distance from that margin.
  - Make this penalty a linear function of this distance.
  - Introduce slack variable ξ_i, which represent the distance that each $\mathbf{x}_i$ goes past the margin boundary.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0 \ \forall i$$

- misclassification when $\xi_i > 1$
- A data point that is on the decision boundary will have $\xi_i = 1$



SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - but we want their number to be minimized
    $\Rightarrow$ by minimizing the sum of slack variables ($\sum_{i=1}^{N} \xi_i$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{N} \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \cdots, N$$

- C dictates which term ($\frac{\|\mathbf{w}\|^2}{2}$ or $C\sum_{i=1}^{N} \xi_i$) will dominate the minimization
  - Small $C \Rightarrow \frac{\|\mathbf{w}\|^2}{2}$ dominates $\Rightarrow$ prefer large margins
    - ★ but allow potentially large numbers of misclassified training examples
  - Large $C \Rightarrow C\sum_{i=1}^{N} \xi_i$ dominates $\Rightarrow$ prefer small numbers of misclassified examples
    - ★ at the expense of having a small margin

SVM - Non-separable case: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b, \xi) = \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{N} \xi_i$$

$$\text{subject to} \quad 1 \leq y_i(\mathbf{w}^T\mathbf{x}_i + b) + \xi_i,\ 0 \leq \xi_i,\ i = 1, \cdots, N$$

- Introducing Lagrange Multipliers α_i, β_i $(i = \{1, \cdots, N\})$, for the constraints, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{N} \xi_i + \sum_{i=1}^{N} \alpha_i\{1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) - \xi_i\} - \sum_{i=1}^{N} \beta_i\xi_i$$

$$\text{subject to} \quad \alpha_i, \beta_i \geq 0, i = 1, \cdots, N$$

- Comparison note: Terms in red were not there in the separable case

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_i and set them to zero:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0, \frac{\partial L_P}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \beta_i = 0$$

- Using $C - \alpha_i - \beta_i = 0$ and $\beta_i \geq 0 \Rightarrow \alpha_i \leq C$
- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

Maximize $L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$

subject to $\sum_{i=1}^{N} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, \cdots, N$

- This is a Quadratic Programming problem in α
- Given α, the solution for $\mathbf{w}$, b has the same form as the separable case
- Note: α is again sparse. Nonzero α_i's correspond to the support vectors

Support Vector Machine: Sparse solution

- Karush-Kuhn-Tucker (KKT) conditions:

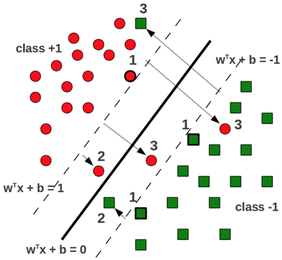
$$\alpha_i \geq 0$$
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i \geq 0$$
$$\alpha_i\{y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\} = 0$$
$$\beta_i \geq 0$$
$$\xi_i \geq 0$$
$$\beta_i\xi_i = 0$$

  Thus for every data point, either $\alpha_i = 0$ or $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i$.

- When $\alpha_i = 0$, the corresponding points do not contribute to the predictive model.
- The remaining points constitute the support vectors, i.e., those for which $\alpha_i > 0$, and hence, they must satisfy $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i$.
  - If $\alpha_i < C \Rightarrow \beta_i > 0 \Rightarrow \xi_i = 0$ ($\mathbf{x}_i$ lies on the margin.)
  - Points $\mathbf{x}_i$ with $\alpha_i = C$ can lie inside the margin and can either be correctly classified if $\xi_i \leq 1$ or misclassified if $\xi_i > 1$.
- Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Support Vectors in the Non-Separable Case

- The separable case has only one type of support vectors
  - ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$
- The non-separable case has three types of support vectors



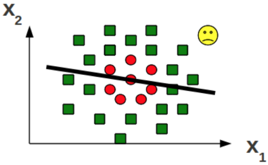
1. Lying on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$ ($\xi_i = 0$)
2. Lying within the margin region ($0 < \xi_i < 1$) but still on the correct side
3. Lying on the wrong side of the hyperplane ($\xi_i \geq 1$)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick
  - More than 2 classes (multiclass classification)
- Popular SVM implementations: libSVM, SVMLight, SVM-struct, etc.

Kernel Methods: Motivation

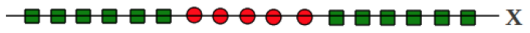
- Often we want to capture nonlinear patterns in the data
  - Nonlinear Classification: Classes may not be separable by a linear boundary
- Linear models (e.g., linear SVM) are not just rich enough



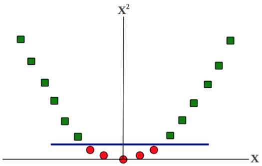
- Kernels: Make linear models work in nonlinear settings
  - By mapping data to higher dimensions where it exhibits linear patterns
  - Apply the linear model in the new input space
  - Mapping means changing the feature representation
- **Note:** Such mappings can be expensive to compute in general
  - Kernels give such mappings for (almost) free

Classifying non-linearly separable data

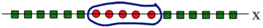
- Consider this binary classification problem



  - Each example represented by a single feature x
  - No linear separator exists for this data
- Now map each example as $x \rightarrow \{x, x^2\}$
  - Each example has now two features ("derived" from the old representation)
- Data now becomes linearly separable in the new representation

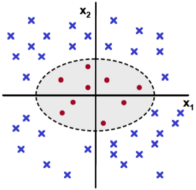


- Linear in the new representation means non-linear in the old representation



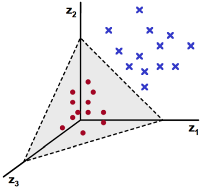
Classifying non-linearly separable data

- Let's look at another example:
  - Each example represented by two features $\mathbf{x} = \{x_1, x_2\}$
  - No linear separator exists for this data



- Now map each example as $\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} \rightarrow \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$
  - Each example has now three features ("derived" from the old representation)

- Data now becomes linearly separable in the new representation



Feature Mapping

- Consider the following mapping ϕ for an example $\mathbf{x} = \{x_1, \cdots, x_n\}$

$$\phi : \mathbf{x} \rightarrow \{x_1^2, x_2^2, \cdots, x_n^2, x_1 x_2, x_1 x_3, \cdots, x_1 x_n, \cdots, x_{n-1} x_n\}$$

- It's an example of a quadratic mapping
  - Each new feature uses a pair of the original features
- Problem: Mapping usually leads to the number of features blow up!
  - Computing the mapping itself can be inefficient in such cases
  - Moreover, using the mapped representation could be inefficient too
    - e.g., imagine computing the similarity between two examples: $\phi(\mathbf{x})^T \phi(\mathbf{z})$
- Thankfully, Kernels help us avoid both these issues!
  - The mapping doesn't have to be explicitly computed
  - Computations with the mapped features remain efficient

Kernels as High Dimensional Feature Mapping

- Consider two examples $\mathbf{x} = \{x_1, x_2\}$ and $\mathbf{z} = \{z_1, z_2\}$
- Let's assume we are given a function k (kernel) that takes as input $\mathbf{x}$ and $\mathbf{z}$

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^\top \mathbf{z})^2 \\
&= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^\top (z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\
&= \phi(\mathbf{x})^\top \phi(\mathbf{z})
\end{aligned}
$$

- The above k implicitly defines a mapping ϕ to a higher dimensional space

$$\phi(\mathbf{x}) = \{x_1^2, \sqrt{2} x_1 x_2, x_2^2\}$$

- Note that we didn't have to define/compute this mapping
- Simply defining the kernel a certain way gives a higher dim. mapping ϕ
- Moreover the kernel $k(x, z)$ also computes the dot product $\phi(\mathbf{x})^T \phi(\mathbf{z})$
  - $\phi(\mathbf{x})^T \phi(\mathbf{z})$ would otherwise be much more expensive to compute explicitly

Kernels: Formally Defined

- Recall: Each kernel k has an associated feature mapping ϕ
- ϕ takes input $x \in \mathcal{X}$ (input space) and maps it to $\mathcal{F}$ ("feature space")
- Kernel $k(\mathbf{x}, \mathbf{z})$ takes two inputs and gives their similarity in $\mathcal{F}$ space

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}, k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

- $\mathcal{F}$ needs to be a vector space with a dot product defined on it
  - Also called a Hilbert Space

Some Examples of Kernels

- The following are the most popular kernels for real-valued vector inputs
  - Linear (trivial) Kernel:

    $$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} \text{ (mapping function } \phi \text{ is identity - no mapping)}$$

  - Quadratic Kernel:

    $$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2 \text{ or } (1 + \mathbf{x}^T \mathbf{z})^2$$

  - Polynomial Kernel (of degree d):

    $$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^d \text{ or } (1 + \mathbf{x}^T \mathbf{z})^d$$

  - Radial Basis Function (RBF) Kernel:

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

    - ⋆ γ is a hyperparameter (also called the kernel bandwidth)

Note: Kernel hyperparameters (e.g., d, γ) chosen via cross-validation

Kernelized SVM Training

- Recall the SVM dual Lagrangian:

$$\text{Maximize } L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, \cdots, N$$

- Replacing $\mathbf{x}_i^T \mathbf{x}_j$ by $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j) = K_{ij}$, where $k(.,.)$ is some suitable kernel function

$$\text{Maximize } L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K_{ij}$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, \cdots, N$$

- SVM learns a linear separator in the kernel defined feature space $\mathcal{F}$
  - This corresponds to a non-linear separator in the original space X

Kernelized SVM Prediction

- Prediction for a test example x (assume b = 0)

$$y = sign(\mathbf{w}^T\mathbf{x}) = sign(\sum_{i \in S} \alpha_i y_i \mathbf{x}^T \mathbf{x}_i)$$

- S is the set of support vectors (i.e., examples for which $\alpha_i > 0$)
- Replacing each example with its feature mapped representation ($\mathbf{x} \rightarrow \phi(\mathbf{x})$)

$$y = sign(\mathbf{w}^T\mathbf{x}) = sign(\sum_{i \in SV} \alpha_i y_i \phi(\mathbf{x})^T \phi(\mathbf{x}_i)) = sign(\sum_{i \in SV} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}))$$