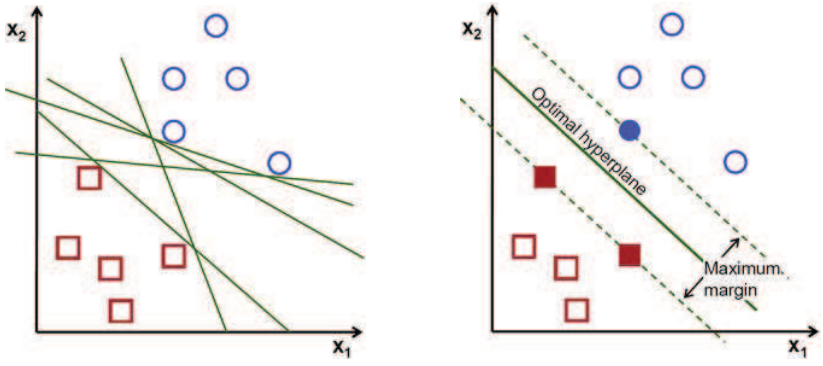
12. SUPPORT VECTOR MACHINES

Support Vector Machines (SVM) are very powerful supervised ML algorithms used in many applications in both linear and nonlinear classification and regression including face detection, text categorization, classification of images, bioinformatics, etc. They are especially used for classification of complex small and medium size data.

- **Maximal Margin Classifier (SVM classification in a linearly separable case)**

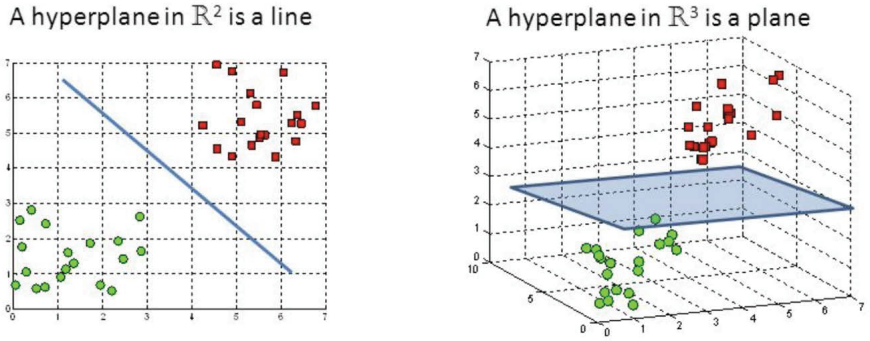
  If we have two classes of data instances that are linearly separable, there are infinitely many lines (or in d-dimensions, infinitely many hyperplanes) that separate them. Some of these separators are worse than the others and might not perform well on the unseen data. The maximal margin classifier is not only separating the two classes, but it also stays as far away from the data instances as possible and is trying to fit the widest possible "street" between the two classes. This width of this street is called the *margin*.



https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

In $\mathbb{R}^d$, a *hyperplane* is a flat affine subspace of dimension $d-1$. It is defined by equation

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d = 0$$



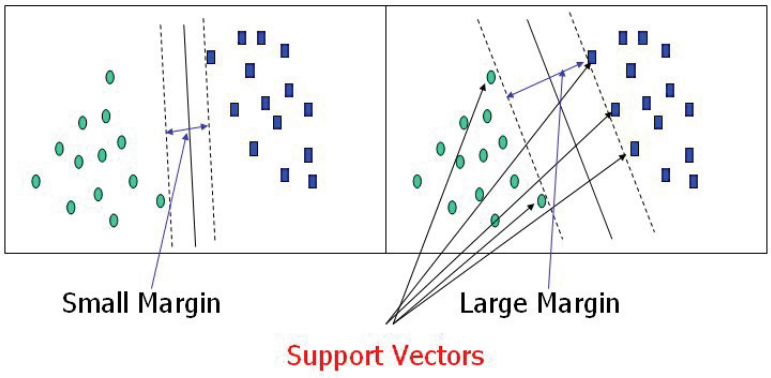
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Assume that we have a training data set consisting of two classes (positive and negative) of linearly separable observations

$$D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$$

separated with a hyperplane that has the widest margin. Notice that adding more data points off the street or moving the points that are off the street (as long as they don't cross the street) will not affect the decision boundary and the street is determined by the data instances on its edges (gutters) which are called *support vectors*.

This classifier is called the *hard margin* classifier since we require all data instances to be on the correct side.



https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

How can we construct the maximal margin classifier?

For simplicity of notation, denote $w := (\theta_1, \ldots, \theta_d)$ and $b := \theta_0$ so that the decision boundary (hyperplane) is given by

$$\boxed{w^T x + b = 0}$$

Our goal is to determine w and b so that this hyperplane has the widest possible margin.

Note that vector w is perpendicular to this hyperplane. Further more, we require

$$w^T x^+ + b \geq 1 \text{ for all data instances in the positive class}$$

and

$$w^T x^- + b \leq -1 \text{ for all data instances in the negative class.}$$

Defining labels $y^{(i)} = +1$ for positive data instances and $y^{(i)} = -1$ for negative data instances, we have

$$\boxed{y^{(i)}(w^T x^{(i)} + b) - 1 \geq 0, \quad i = 1, 2, \ldots, n}$$

for all data instances.

In addition, for data points in the gutters of the classifier, we have

$$y^{(i)}(w^T x^{(i)} + b) - 1 = 0. \tag{1}$$

Next, we express the width of the margin (the distance between the two gutters). Consider two points in the gutters (one positive and one negative). The width of the margin is

$$(x^+ - x^-)^T \frac{w}{\|w\|} = \frac{1 - b + 1 + b}{\|w\|} = \frac{2}{\|w\|}.$$

In order to maximize the width of the margin, we need to minimize $\|w\|$, or equivalently, we minimize $\frac{1}{2}\|w\|^2 = \frac{1}{2} w^T w$. Therefore, to find w and b for the maximum margin classifier, we need to solve the optimization problem

$$\begin{array}{ll} \text{minimize}_{w,b} & \frac{1}{2} w^T w \\ \text{subject to} & y^{(i)}(w^T x^{(i)} + b) - 1 \geq 0, \quad i \in \{1, \ldots, n\}. \end{array}$$

This is an optimization problem with a convex quadratic objective and only linear constraints. It can be solved using commercial quadratic programming code.

However, to understand the full power of support vector machines and use of kernels, we will derive a dual problem for the above (primal) optimization problem (see APPENDIX).

We construct the Lagrangian

$$\mathcal{L}(w, b, \alpha) := \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right]$$

where $\alpha_i \geq 0$, for all $i \in \{1, \ldots, n\}$. To find the dual problem, we need to minimize $\mathcal{L}(w, b, \alpha)$ with respect to w and b, for fixed α. We find partial derivatives

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y^{(i)}$$

and setting them to 0, we get

$$w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0.$$

Plugging this expression for w back to $\mathcal{L}$ we find

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)} + \sum_{i=1}^{n} \alpha_i. \tag{2}$$

Hence, the above primal minimization problem is converted to its dual problem

$$\begin{aligned} \text{maximize}_\alpha \qquad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + \sum_{i=1}^{n}\alpha_i \\ \text{subject to} \qquad & \sum_{i=1}^{n}\alpha_i y^{(i)} = 0 \quad \text{and} \quad \alpha_i \geq 0,\ i \in \{1,\ldots,n\}. \end{aligned}$$

Coefficients α_i are found using a quadratic programming solver. There is a coefficient α_i corresponding for each data point $(x^{(i)}, y^{(i)})$ and, according to the Karush-Kuhn-Tucker conditions, $\alpha_i \neq 0$ only for support vectors.

Therefore, once $\alpha_1, \ldots, \alpha_n$ are found, we have that w and b are given by

$$w = \sum_{\alpha_i > 0} \alpha_i y^{(i)} x^{(i)}$$

and

$$b = \frac{1}{y^{(m)}} - w^T x^{(m)}$$

$$\boxed{b = y^{(m)} - \sum_{\alpha_i > 0} \alpha_i y^{(i)} (x^{(i)})^T x^{(m)}}$$

for any support vector $(x^{(m)}, y^{(m)})$.

Finally, note that the decision boundary $w^T x + b = 0$ is given by

$$\boxed{\sum_{\alpha_i > 0} \alpha_i y^{(i)} (x^{(i)})^T x + b = 0} \tag{3}$$

which together with (2), implies that both the function $\mathcal{L}$ and the decision boundary depend *only on the dot products.*

- **Support Vector Classifier**

  (SVM classification in a non-linearly separable case – soft margin)

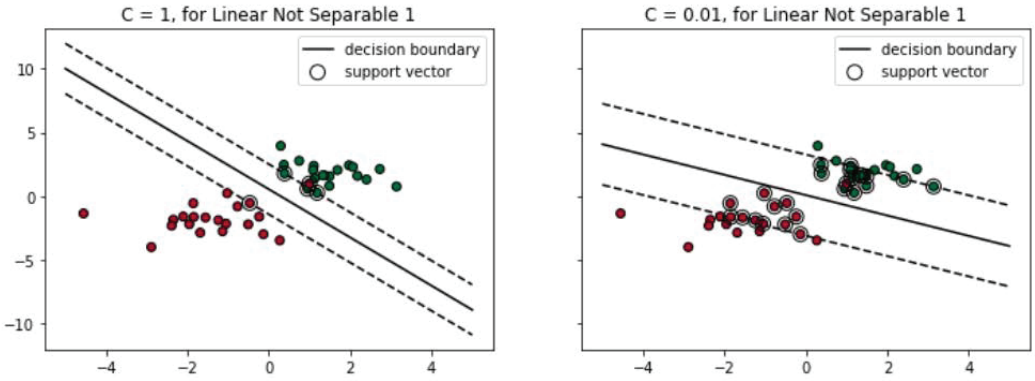
  If we require that all data instances are off the street and on the correct side, this is called hard margin classification. This works only if data is linearly separable, while most of the real-life data is not linearly separable. To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we might want to find a balance between maximizing the margin and tolerating some margin violation. This is called *soft margin classification*.

  We reformulate the minimization problem to

$$\begin{aligned} \text{minimize}_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i \in \{1, \ldots, n\} \\ & \xi_i \geq 0, \quad i \in \{1, \ldots, n\} \end{aligned}$$

  A data instance is now allowed to violate the margin, but will pay a cost of the objective function being increased. Thus, the hyperparameter C allows for margin violation and controls the relative weight between the goal of making the margin as wide as possible (term $\frac{1}{2} w^T w$) and of ensuring that most data points are not violating the margin.

  If the model is overfitting, we can reduce C to regularize the model.

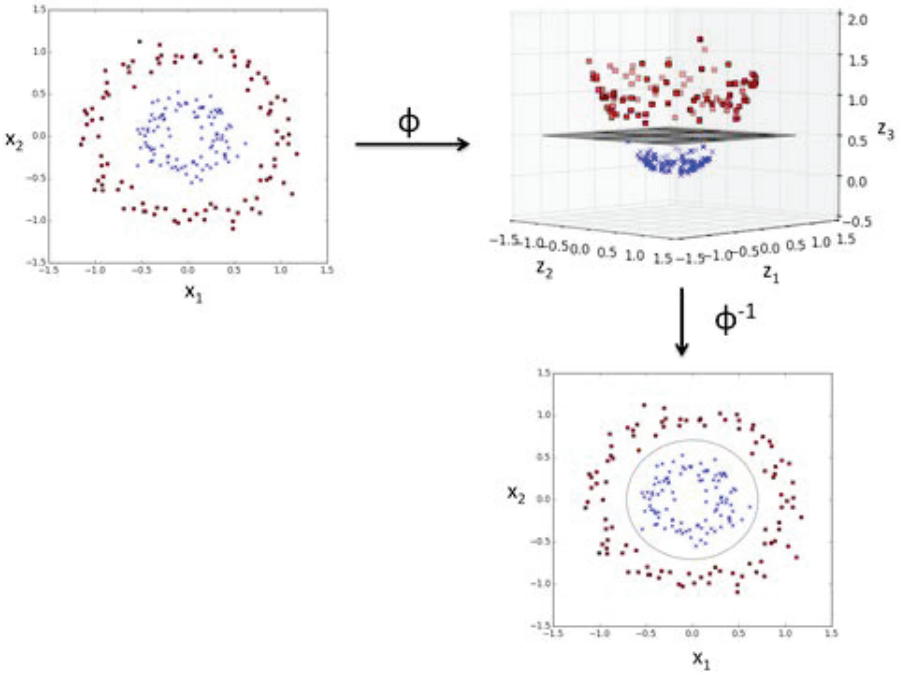


https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

- **Support Vector Machine**

  (Nonlinear SVM classification – kernels)

  The main idea is to create more features by applying some transformation Φ on the existing features. By adding new features (dimensions in the data set), we end up in a higher dimensional space in which the data becomes linearly separable. Then, we can separate the data using SVC and project the decision boundary back to the original space. In the higher dimensional space the decision boundary is linear, but in the original space it will be nonlinear.



https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html

Remark:

– Notice that the function $\mathcal{L}$ and the decision boundary in (2)-(3) depend only the dot products and if we have a transformation

$$x \mapsto \Phi(x)$$

all we need to compute is the dot product in the new space.

In order to make mathematics possible and computation efficient, SVM uses kernel functions. A function K is a kernel if there is a map Φ such that

$$K(a,b) = \Phi(a)^T \Phi(b), \quad \text{for} \quad a, b \in \mathbb{R}^d.$$

Mercer's Theorem (1909) gives sufficient and necessary condition for a function to be a kernel.

Therefore, if we have a kernel function $K(a, b)$, the decision boundary $w^T x + b = 0$ is given by

$$\sum_{\alpha_i > 0} \alpha_i y^{(i)} K(x^{(i)}, x) + b = 0$$

where

$$b = y^{(m)} - \sum_{\alpha_i > 0} \alpha_i y^{(i)} K(x^{(i)}, x^{(m)})$$

where $(x^{(m)}, y^{(m)})$ is any support vector.

In addition, the hypothesis for SVM is given by

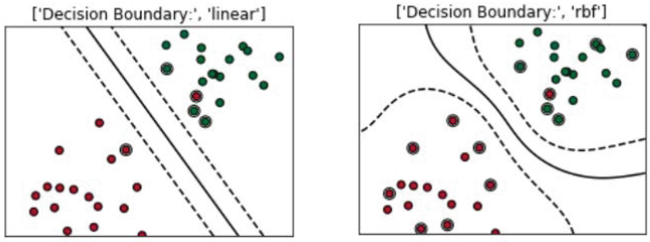
$$\boxed{h(x) = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y^{(i)} K(x^{(i)}, x) + b\right)}$$

Some commonly used kernels are:

* linear kernel: $K(a, b) = a^T b$
* polynomial kernel: $K(a, b) = (1 + a^T b)^r$, where r is the degree of the polynomial
* Gaussian RBF (Radial Basis Function) kernel: $K(a, b) = e^{-\gamma \|a-b\|^2}$
* sigmoid kernel: $K(a, b) = \tanh(\gamma a^T b + r)$

Instead of transforming data points to a higher dimensional space using a transformation Φ and finding dot products in that higher dimensional space (which can be infinitely dimensional), we use the kernel function in the original space and we do not even need to know the exact form of Φ. This method is called the *kernel trick*.

– We can find the optimal kernel and kernel hyper-parameters by using grid search.



https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

Example 1: Consider the polynomial kernel

$$K(a, b) = (1 + a^T b)^2$$

where $a, b \in \mathbb{R}^2$. What is the transformation Φ associated with it?

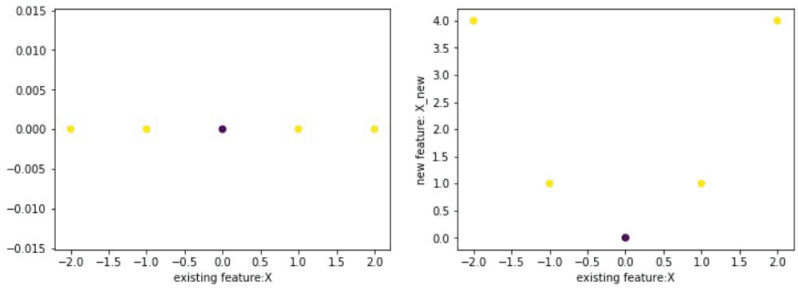
Note that

$$\begin{aligned} K(a,b) &= (1 + a_1b_1 + a_2b_2)^2 \\ &= 1 + a_1^2b_1^2 + a_2^2b_2^2 + 2a_1b_1 + 2a_2b_2 + 2a_1b_1a_2b_2 \\ &= \Phi(a)^T\Phi(b) \end{aligned}$$

Therefore,

$$\Phi(a) = \Phi(a_1, a_2) = (1, a_1^2, a_2^2, \sqrt{2}\,a_1, \sqrt{2}\,a_2, \sqrt{2}\,a_1a_2)$$

Therefore, the polynomial kernel creates new features by applying the polynomial combination on all the existing features.



https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

Example 2: Consider the Gaussian RBF kernel

$$K(a,b) = e^{-(a-b)^2}$$

where $a, b \in \mathbb{R}$ and $\gamma = 1$. What is the transformation Φ associated with it?

Consider

$$\begin{aligned} K(a,b) &= e^{-a^2+2ab-b^2} \\ &= e^{-a^2} \cdot e^{-b^2} \cdot e^{2ab} \\ &= e^{-a^2} \cdot e^{-b^2} \cdot \sum_{k=0}^{\infty} \frac{2^k a^k b^k}{k!} \\ &= e^{-a^2} \cdot e^{-b^2} \cdot \left(1 + 2ab + 2a^2b^2 + \frac{2^3}{3!}a^3b^3 + \frac{2^4}{4!}a^4b^4 + \ldots\right) \\ &= \Phi(a)^T\Phi(b) \end{aligned}$$

Therefore

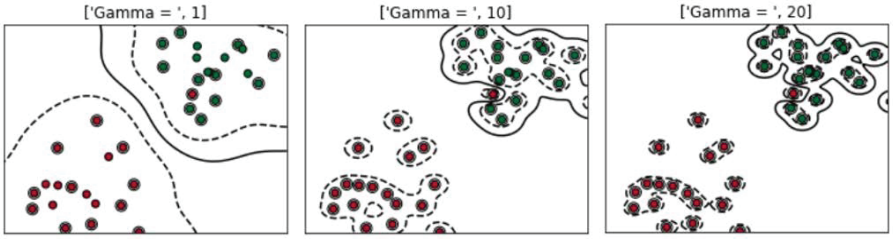
$$\Phi(a) = e^{-a^2}\left(1, \sqrt{2}a, \sqrt{2}a^2, \sqrt{\frac{2^3}{3!}}a^3, \sqrt{\frac{2^4}{4!}}a^4, \ldots\right)$$

Note that $\Phi : \mathbb{R} \to \mathbb{R}^\infty$.

To understand Gaussian RBF kernel, consider a test data point $x^* \in \mathbb{R}^d$. If x^* is far from a training observation $x^{(i)}$, then $\|x^{(i)} - x^*\|$ will be large and

$$K(x^{(i)}, x^*) = e^{-\gamma \|x^{(i)} - x^*\|^2}$$

will be small. This means that $x^{(i)}$ plays virtually no role in classifying x^*. The hyperparameter γ controls the influence of data points on the decision boundary. Increasing γ makes the bell-shape curve narrower and each point's range of influence is smaller making the decision boundary more irregular and wiggling around individual data instances. A small value of γ makes the bell-shape curve wider and instances have a larger influence making the decision boundary smoother. Next graphs show decision boundaries for $\gamma \in \{1, 10, 20\}$.



https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

- **Summary and Remarks on SVM classification**:
  - Note that for a test data observation with input x^*, the classification is made based on the sign of $w^T x^* + b$. However, we can also make use of the *magnitude* of $w^T x^* + b$ as how confident we are in our prediction. For example, if $w^T x^* + b$ is a large positive number, then x^* is far from the hyperplane and we are highly confident in predicting it as a positive data instance.
  - SVM is sensitive to feature scales (for example, use `StandardScaler` for each feature).
  - Hinge loss for a single data point $(x^{(i)}, y^{(i)})$ with prediction $\hat{y}^{(i)}$, where we have labels $y^{(i)} \in \{-1, +1\}$ is defined by

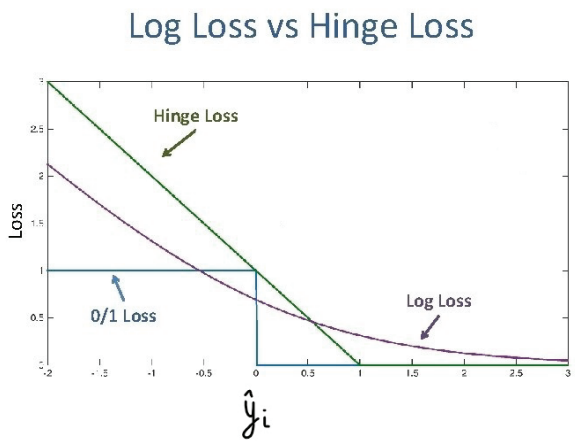
$$\text{Loss}(x^{(i)}, y^{(i)}) = \max\{0, 1 - \hat{y}^{(i)}\, y^{(i)}\}$$

  - It turns out that the minimization problem for SVC (soft-margin classifier) can be rewritten as

$$\text{minimize}_{w,b} \left\{ \sum_{i=1}^{n} \max\{0, 1 - y^{(i)}\, \hat{y}^{(i)}\} + \lambda \|w\|^2 \right\}$$

    where λ is a nonnegative hyperparameter. When λ is larger, $\|w\|$ will be smaller and less violations to the margin are tolerated. Thus, a large value of λ corresponds to the large value of hyperparameter C.

    Also, note that the above expression is of the form "Loss + Penalty", for the hinge loss.
  - Hinge loss is closely related to the negative log loss used in Logistic Regression and this is why SVC and Logistic Regression have similar results. Next graph shows negative log loss and hinge loss when $y^{(i)} = 1$.



https://slidetodoc.com/machine-learning-data-mining-cscnsee-155-lecture-2/

- For linear SVC we can use `sklearn`
  - `LinearSVC(loss="hinge")`
  - `SVC(kernel="linear")`
  - `SGDClassifier(loss="hinge")`
- Hyper-parameters C and γ
  - C controls the trade off between smooth decision boundary and classifying data instances correctly. A large value of C means more data points will be classified correctly.
  - γ defines how far the influence of a single data point reaches. If γ has a very high value, then the decision boundary will depend just on the points that are very close to it which results in ignoring data points that are very far from the decision boundary. This is because the closer points get more weight and the decision becomes a wiggly curve. If γ is low, then even the points that are far away get considerable weight and we get a smoother curve.
- Computational complexity
  - `LinearSVC` class is in the `liblinear` library and has computational complexity $O(n \times d)$, where n is the number of data instances and d is the number of features.
  - `SVC` class is in the `libsvm` library, which supports the kernel trick, and has computational complexity between $O(n^2 \times d)$ and $O(n^3 \times d)$, so it can become very slow if we have a large training data set.

- **SVM for more than two classes**

  Two most popular approaches to extending SVM from binary to K-class classification with $K > 2$ are as follows.

  - *One Vs. One Classification*

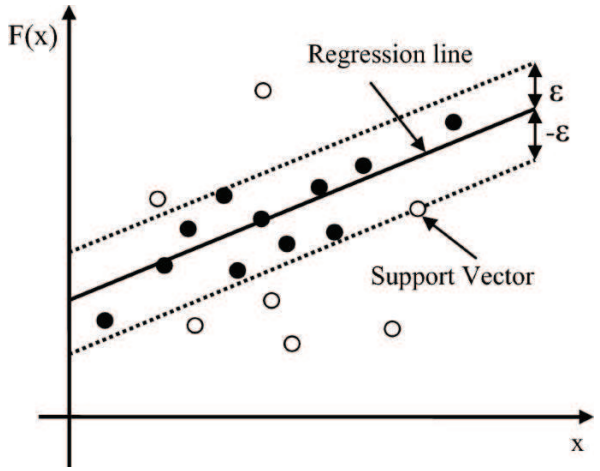
    In this approach we construct $\binom{K}{2}$ SVMs, each of which compares a pair of classes. We classify a test observation using the majority voting among these $\binom{K}{2}$ classifiers.

– *One Vs. All Classification*

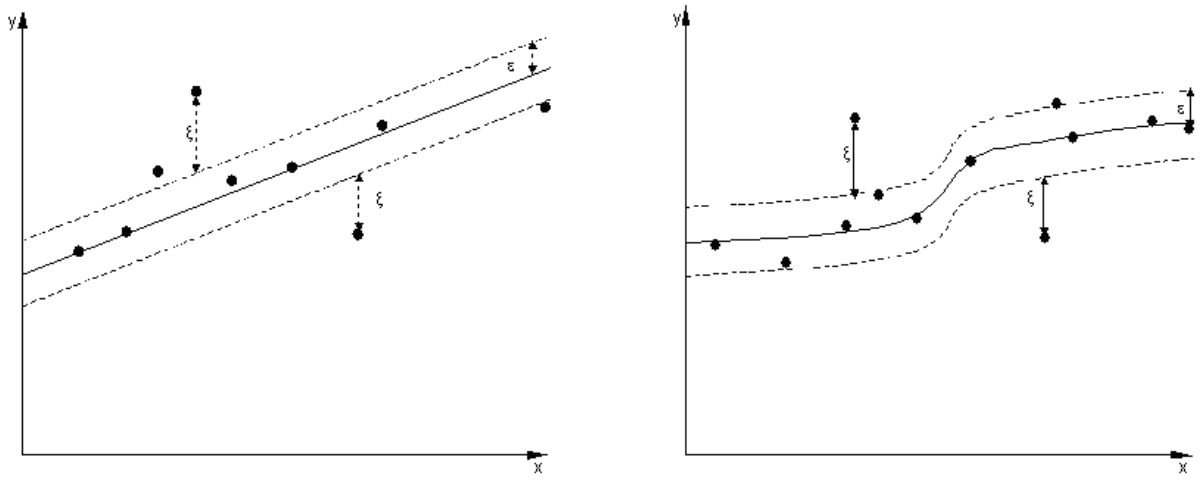
We fit K SVMs, each time comparing one of the classes (coded as $+1$) to the remaining group of $K-1$ classes (coded as -1). Let w_k and b_k be parameters from this classification, $k \in \{1, \ldots, K\}$. If x^* is a test data instance, we assign it to the class for which $(w_k)^T x^* + b_k$ is the largest as this amounts to the high level of confidence that x^* belongs to that class.

- **SVM Regression**

  The objective is to fit as many instances as possible on the "street" while limiting margin violation. `LinearSVR` creates a linear SVM regression, while `SVR` class supports the kernel trick.



https://link.springer.com/article/10.1007/s00500-018-3615-x



https://kernelsvm.tripod.com/

Python code: Lecture_12_SVM.ipynb

Homework 7:

- *Part 1:* Watch Prof. Abu-Mostafa's Lectures [4] and explain what margin and non-margin support vectors are.
- *Part 2:* Create a binary classification problem using `sklearn.datasets.make_moons`. Build a SVM classifier model and investigate the effect of hyper-parameters C, γ, and kernels on the model performance.

APPENDIX ON LAGRANGE DUALITY

Consider the primal constrained optimization problem

$$\begin{aligned} \text{minimize}_w \quad & f(w) \\ \text{subject to} \quad & g_i(w) \leq 0,\ i = 1, \ldots, k \\ & h_i(w) = 0,\ i = 1, \ldots, l \end{aligned}$$

We define the generalized Lagrangian

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

where $\alpha_i \geq 0$ and β_i are Lagrange multipliers. Note that

$$\max_{\alpha,\beta} \mathcal{L}(w, \alpha, \beta) = \begin{cases} f(w) & \text{if } w \text{ satisfies the primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

Therefore, the original primal problem can be reformulated as

$$\boxed{\min_w \max_{\alpha,\beta} \mathcal{L}(w, \alpha, \beta) = \min_w \theta_{\text{primal}}(w)}$$

and the dual optimization problem is posed by

$$\boxed{\max_{\alpha,\beta} \min_w \mathcal{L}(w, \alpha, \beta) = \max_{\alpha,\beta} \theta_{\text{dual}}(\alpha, \beta)}$$

Proposition: (Minimax inequality) For any function $\phi(x, y)$, defined over $x \in X, y \in Y$, we have

$$\max_{y \in Y} \min_{x \in X} \phi(x, y) \leq \min_{x \in X} \max_{y \in Y} \phi(x, y)$$

Proof: Note that for every fixed $x' \in X$ and $y' \in Y$ we have

$$\min_{x \in X} \phi(x, y') \leq \max_{y \in Y} \phi(x', y)$$

Take maximum over $y' \in Y$ on the left side, and take minimum of $x' \in X$ on the right side. □

Therefore,

$$\max_{\alpha,\beta} \min_{w} \mathcal{L}(w, \alpha, \beta) \leq \min_{w} \max_{\alpha,\beta} \mathcal{L}(w, \alpha, \beta)$$

If f and g_i's are convex and h_i's are affine and if g_i's are strictly feasible (i.e., there exists some w such that $g_i(w) < 0$, for all $i = 1, \ldots, k$), there exist w^*, α^*, β^* so that w^* is the solution to the primal problem and α^* and β^* are the solution to the dual problem. In addition, w^*, α^*, β^* satisfy Karush-Kuhn-Tucker conditions:

$$\begin{aligned}
\frac{\partial \mathcal{L}(w^*, \alpha^*, \beta^*)}{\partial w_i} &= 0, \quad i = 1, \ldots, d \\
\frac{\partial \mathcal{L}(w^*, \alpha^*, \beta^*)}{\partial \beta_i} &= 0, \quad i = 1, \ldots, l \\
\alpha_i^* \, g_i(w^*) &= 0, \quad i = 1, \ldots, k \\
g_i(w^*) &\leq 0, \quad i = 1, \ldots, k \\
\alpha_i^* &\geq 0, \quad i = 1, \ldots, k
\end{aligned}$$

In particular, notice that if $\alpha_i^* > 0$ then $g_i(w^*) = 0$. This means that, with respect to hard margin classifier derivation, the coefficients α_i's are not zero only for the support vectors.

References and Reading Material:

[1] To code the SVM classifier from scratch, see

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[2] *An Introduction to Statistical Learning*, James et al., Chapter 9

[3] Andrew Ng's notes on Support Vector Machines and Kernels (pages 45-64)

[4] Youtube videos – Machine Learning Course Caltech CS 156, Yaser Abu-Mostafa's lectures #14 SVM, #15 Kernel Methods, #16 RBF