SL06: Kernel Methods & SVMs

Introduction:

- Given a set of linearly separable data, the goal would be to find the line that has the following characteristics:
  - Defines the decision boundaries.
  - Provides the best fit and least commitment to the data (to avoid overfitting).

Support Vector Machines:

- The linear classifier function would be:

$$y = w^T x + b$$

  $y \in \{-1, +1\}$ → Labels

  w, b → Parameters of the plane.

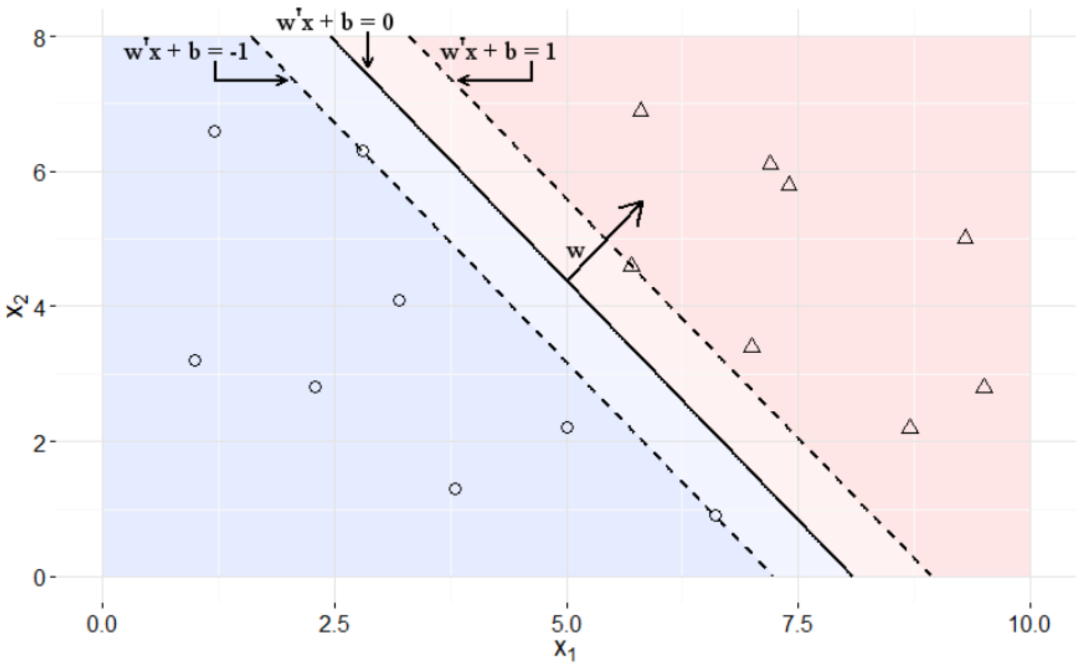
  - The equation for the line we're looking for (best fit and least commitment to the data) will be:

$$w^T x + b = 0$$

    Because that's the decision boundary itself, so we don't have positive or negative classification.
  - If we moved that line towards the positive or the negative values, we'll end up with either of these equations:

$$w^T x_1 + b = 1$$
$$w^T x_2 + b = -1$$

  - We need our boundary decision line to be as far from the data as possible. Because this ensures the least commitment to the data.



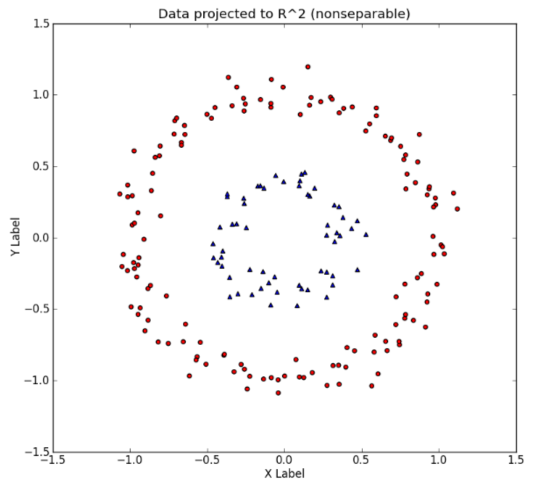
- This means that we need the vector defining the distance between the two boundary lines (the one near positive values, and the one near negative values) to be maximal. Subtracting the two lines' equations will give us an equation for the vector of the distance between the lines.
$$w^T(x_1 - x_2) = 2$$
- To get the values of $(x_1 - x_2)$ we divide both sides by the length of w → $\| w \|$:
$$\frac{w^T}{\| w \|}(x_1 - x_2) = \frac{2}{\| w \|}$$
- $\frac{w^T}{\|w\|}$ is a normalized version of w, which is basically a vector the points in the same direction of w, but sets in the unit sphere (a unit vector in the direction of w). This is effectively 1.
- Furthermore, since w has the property of describing the hyperplane boundary, it itself is perpendicular to the boundary (like $(x_1 - x_2)$). So, $\frac{w^T}{\|w\|}(x_1 - x_2)$ gives us the length of the projection $(x_1 - x_2)$ onto w, and the dot product is $\frac{2}{\|w\|}$ which gives us the length of $(x_1 - x_2)$.
- In order to maximize the length $(x_1 - x_2)$, we want to minimize $\| w \|$.
- $\frac{w^T}{\|w\|}(x_1 - x_2)$ is called the Margin. We want to find a solution that maximizes the Margin, while producing a correct classification.
- Mathematically, classifying everything correctly is equivalent to:
$$y_i(w^T x + b) \geq 1 \quad \forall_i$$
What we need is that $(w^T x + b)$ is ≥ 1 for the positive example, and ≤ 1 for the negative example. Therefore, we have y_i in the equation. If y_i is $+1$, the $(w^T x + b) \geq 1$. If y_i is -1, the $(w^T x + b) < 1$.

• Maximizing $\frac{2}{\|w\|}$ ends up to be hard, but it's equivalent to minimizing $\frac{1}{2} \| w \|^2$ which is easier because it can be transformed into a quadratic programming problem:
$$W(\propto) = \sum_i \propto_i - \frac{1}{2} \sum_{ij} \propto_i \propto_j \, y_i y_j {x_i}^T x_j$$
- This is the sum of alpha (weights on data points) for each data point minus the one-half times for every pair of examples the product of their alphas, labels and their values. Given that:
$$\propto_i \geq 0$$
$$\sum_i \propto_i \, y_i = 0$$
- The sum of the weights on the points categorized as $y_i = -1$ should be equal to those categorized as $y_i = 1$ As for $W(\propto)$, the second term controls the summed weights in the first term from getting too large. The second term takes into account the categories of each pair of points ($y_i y_j = 1$ if they are in the same class, -1 if they differ) and a measure of similarity (evoked by ${x_i}^T x_j$).
- When we obtain the optimal weights $\propto_i$ it turns out that most of them are equal to zero.
- The points that have non-zero weights are the only points that contribute to the calculation of $W(\propto)$, and all of them fall on the margin, satisfying $y_i(w^T x + b) \geq 1 \ \ \forall_i$
- This means that we can have all the "support" we need to find the optimal w from only the vectors that have non-zero alphas. Hence the name "Support Vector Machines".

  - Once we find the alphas that maximizes this equation, we can obtain the parameter values for our dividing hyperplane from:

$$w = \sum_i \propto_i y_i x_i$$

  - Once you recover w, you can have b.
- Optimal learner:
  - The fact that only some points affect finding the optimum w is logical. Because only points near the boundary have influence on the decision line.
  - This is similar to $k - NN$, but it has a hint about which points really matter.
  - The x's in the equation that we're trying to solve are represented by a dot product ${x_i}^T x_j$, which gives us a notion of **similarity** since dot product represents the magnitude of the projection. Orthogonal projection gives dot product of 0 and if the vectors are pointing in a similar direction it gives a larger magnitude.
  - This means that this equation basically tries to
    1. Find all pairs of points.
    2. Figure out which one matters for defining the decision boundary.
    3. Think about how they link to one another in terms of their output labels, with respect to how similar they are.
- Linearly married:
  - In some instances, you won't have linearly separable data. But this can be resulting from noise points that can be easily flipped or ignored, ending up with linearly separable set.
  - However, in other cases, the data is separable but not linearly.



  - One way to deal with this kind of data, is to change the 2-dimensional points to 3-dimensional points, without adding any information:

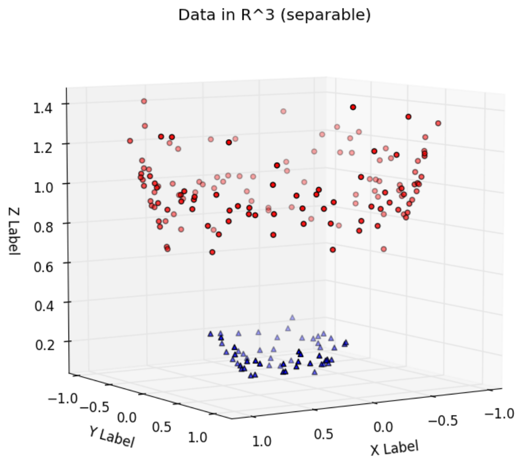
$$\emptyset(q) = \langle {q_1}^2, {q_2}^2, \sqrt{2}\, q_1 q_2 \rangle$$

  - We then utilize the way this function represents the data, in finding the dot product ${x_i}^T x_i$.

- If we have two points (x_1, x_2) and (y_1, y_2) then:

$$\begin{aligned}\emptyset(x)^T\emptyset(y) &= x_1{}^2 y_1{}^2 + 2x_1x_2y_1y_2 + x_2{}^2y_2{}^2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= (x^Ty)^2\end{aligned}$$

- $(x^Ty)^2$ turns out to be the equation of a circle. This means that the notion of falling above of below a decision line, to falling inside or outside a decision circle. Or in other words, falling above or below a decision plane.



- What we observe here is called the "Kernel Trick". The transformation of the original 2D data to the higher dimension representation can be computed in the implicit feature space, never requiring computations on the coordinates to the transformed space. This is typically much more computationally efficient, and therefore desirable for learning endeavors.
- Now the new notion of "Kernel", which can be represented generally using this equation:

$$K(x, y) = (x^Ty + c)^p$$

- The Kernel basically computes the measure of similarity.
- We can inject domain knowledge into SVMs through changing the Kernel form.
- Note that the non-linear map $\emptyset$ exists, such that:

$$K(x_i, x_j) = \langle\emptyset(x_i), \emptyset(x_j)\rangle$$

  only if the kernel K satisfies the **Mercer Condition**. Which mean that $\emptyset$ exists only if K is a positive definite ($K \in \mathbb{R}$).

SVM Summary:

- We need to maximize the margin.
- Finding the maximum margin is a hard optimization problem. Quadratic programming is the solution.
- Support vectors: The data points that have the biggest effect on maximizing the margin.
- SVMs are in some sense eager/lazy learners. They're eager because we define the points that matter the most before starting the lazy learning.
- Kernel trick: We can project data into higher dimensional space and do the comparisons there by turning the dot product $(x^T y)$ into another similarity metric $K(x, y)$.
- Kernels have to satisfy Mercer Condition.

Boosting and Overfitting:

- In Boosting, what we normally keep track of is error:
  - Error: The probability of a prediction that disagrees with the training set.

  However, what we are also keeping track of is **confidence**. How strong our belief in the results is.

$$H_{final}(x) = sgn\left(\sum_t \propto_t h_t(x)\right)$$

- The Boosting function outputs the sign of the sum over the weak learners: $+1$, -1 or 0
- Dividing the above formula by the weights, normalizing the output:

$$H_{final}(x) = sgn\left(\frac{\sum_t \propto_t h_t(x)}{\sum_t \propto_t}\right)$$

- This normalization reduces the term $\frac{\sum_t \propto_t h_t(x)}{\sum_t \propto_t}$ between $+1$ and -1, without affecting the $H_{final}(x)$ itself.
- The reason why boosting is robust to overfitting even with adding more learners is that adding more learners increases the margin (the distance between boundaries), while maintaining constant error.
- Boosting, though, tends to overfit if the underlying weak learner uses an algorithm that tends to overfit (e.g. a Neural Network with a lot of hidden layers).