NYC DATA SCIENCE ACADEMY

Python Machine Learning

Support Vector Machines & Support Vector Regression

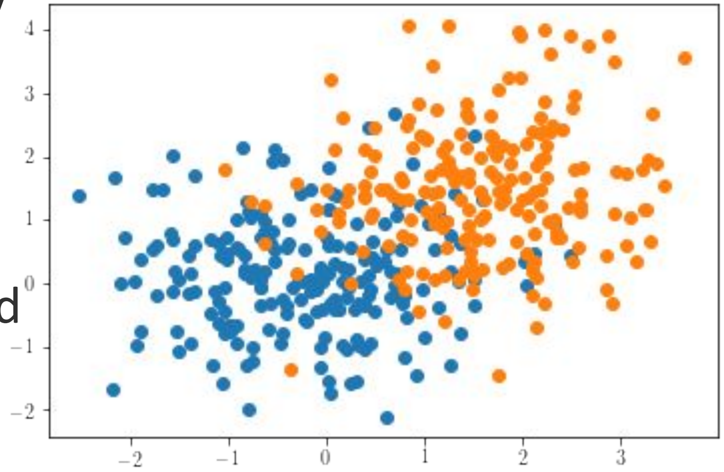
NYC Data Science Academy

Outline

- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- Support Vector Regression

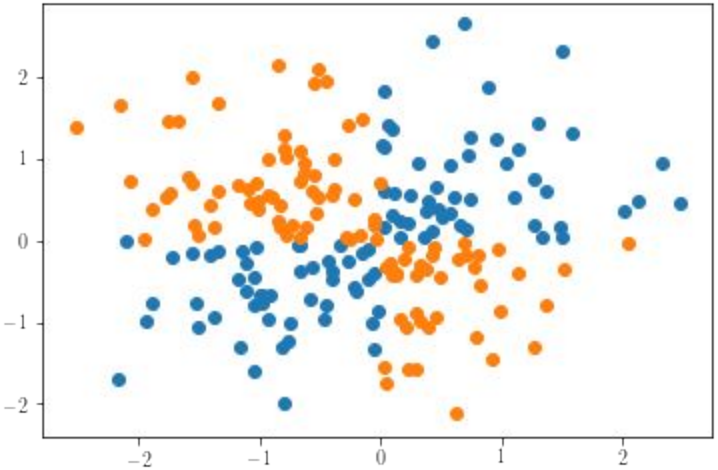
Why Do Classifiers Fail?

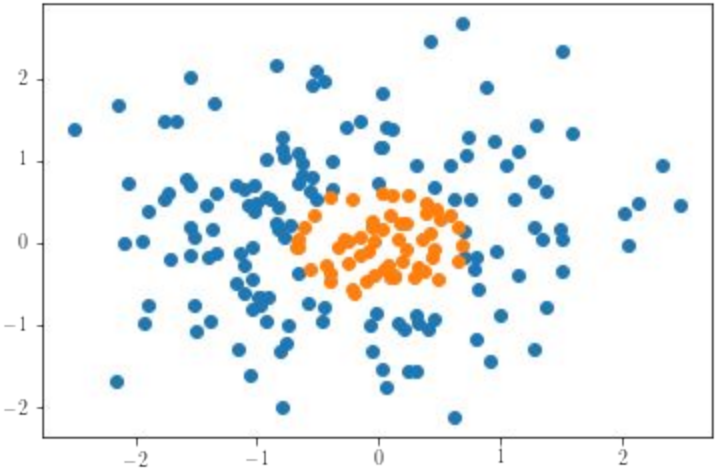
- In a perfect world we can imagine a classifier which always discerns the class labels perfectly. In real life a classifier often encounters classification errors. What is the plausible cause?
- Even for a linear classifier, one issue is that samples of different classes can mingle in the same region of the feature space
- No matter where the decision boundary is placed, it always makes some classification errors!
- LDA and logistic classifier attempt to find optimal decision hyperplanes based on different **probabilistic** model assumptions



The Lack of Suitable Vocabulary

- The issue that any linear classifier faces is the lack of necessary vocabulary to describe the true decision boundary, as can be seen from the following illustrations





- Any choice of a decision line would fail the classification task miserably,

as opposite to the human brain’s success

The Trilogy of SVMs

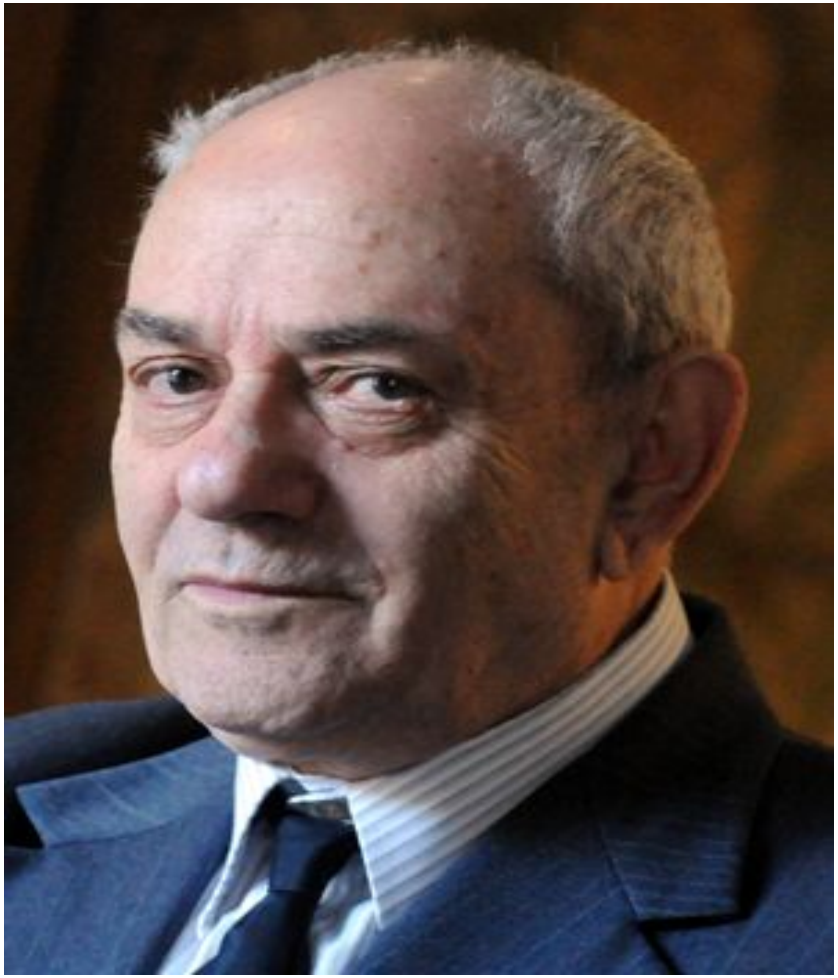
- Different high performance classifier models try to introduce nonlinearity into their models in different ways, which enable them to describe the decision boundaries more accurately
- **Bagging/random forests**, **tree boosting**, **neural networks**, **SVMs** introduce nonlinearity from different angles, which induce different performance trade-off
- **SVMs** stand out in that
  - They are built on top of a linear model called **MMC**, maximal margin classifier
  - It avoids the estimation of probabilities in its formulation

MMC → **SVC** → **SVM**

Support Vector Machines (Vapnik & Cortes 1996)

- **Support vector machines** (**SVMs**) are supervised learning methods used for classification analysis
- The idea of **SVMs** can be modified and can be carried out in the regression setting, called **SVRs**
- Unlike linear discriminant analysis or logistic regression, **SVMs** approach the **two-class** classification problem in a direct way: constructing linear/nonlinear decision boundaries, by explicitly separating the data into two different classes as complete as possible.
- The linear decision boundaries of **SVCs** are called decision **hyperplanes** in the feature space
- The non-linear decision boundaries of the general **SVMs** are called decision hypersurfaces in the feature space

Vapnik, Founder of Statistical Learning Theory and Inventor of SVM

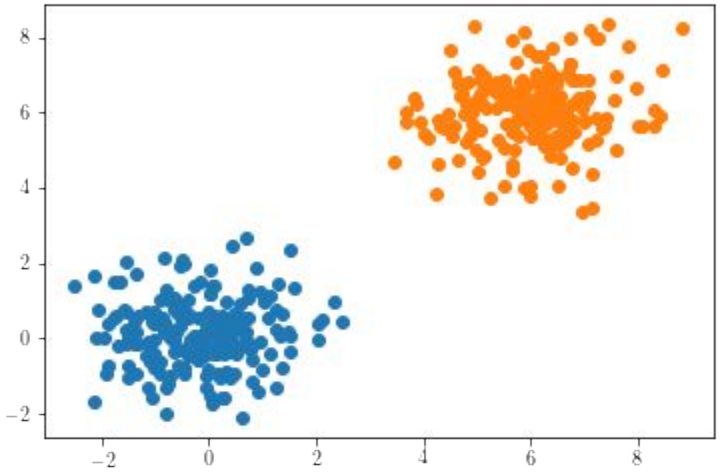


Outline

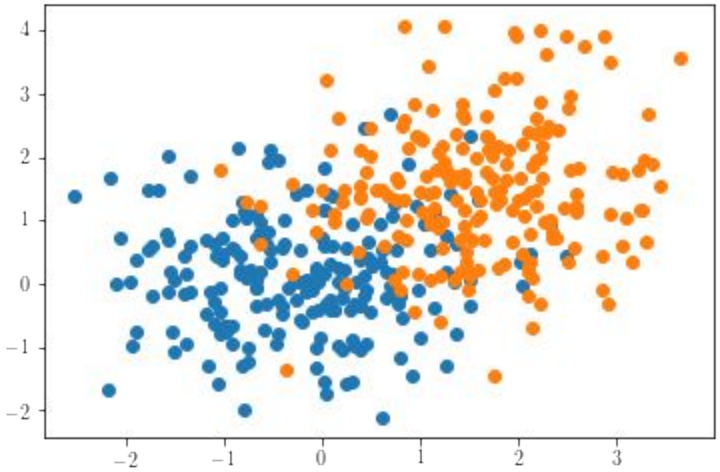
- **Support Vector Machines**
  - **Separating Hyperplanes**
  - **The Support Vector Classifier**
  - **Kernels**
- **Support Vector Regression**
  - epsilon-insensitive loss function
  - Using kernels to approximate functions

Linearly Separable Data

- Not all data is born equal. **SVMs** start by focusing their attention on a particular type of data sets
- A data set is said to be **linearly separable** if there exists a linear decision hypersurface which classifies perfectly



Linearly Separable Data



Linearly Non-Separable Data

The Rationale of SVMs

- As is evident from the above visualization, a linearly separable data set allows uncountable-infinitely many linear classifiers to perform perfect classifications
- Firstly **SVMs** try to pin down some unique **optimal** linear classifier among them, known as **MMC**, maximal margin classifier
- In what sense is this **optimal**?
- What if a data set is no longer linearly separable? How can a linear classifier cope with this situation?
- **SVMs** introduce a relaxed concept called **SVC**, support vector classifier, to handle this slightly more general situation

Support Vectors + Kernel Machines

- The idea of **support vectors** is the core concept fundamental to **SVMs**
- In short, the support vectors are the bad boy data points within the original dataset which obstruct the **MMC/SVC** from doing a better job

- As has been mentioned, linear classifiers lack the vocabulary to describe the true non-linear decision boundaries in the general setting
- **SVMs** make the **SVC** turbo-charged by introducing the concept of **kernel trick** and hop into very high dimensional new feature spaces
- At the end, **SVMs** become the high performance classifiers which are capable of producing highly complex decision boundaries, while making their underlying mathematics tractable
- Before revealing **SVMs**, we go over some basic concepts in geometry

Hyperplanes

- A **hyperplane** of a *p-dimensional* Euclidean space *V* is an affine linear subspace of dimension *p*-1, which can be described by a single linear equation of the form (in Cartesian coordinates):

$$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = 0$$

- It is more convenient to write the equation above in a matrix notation:

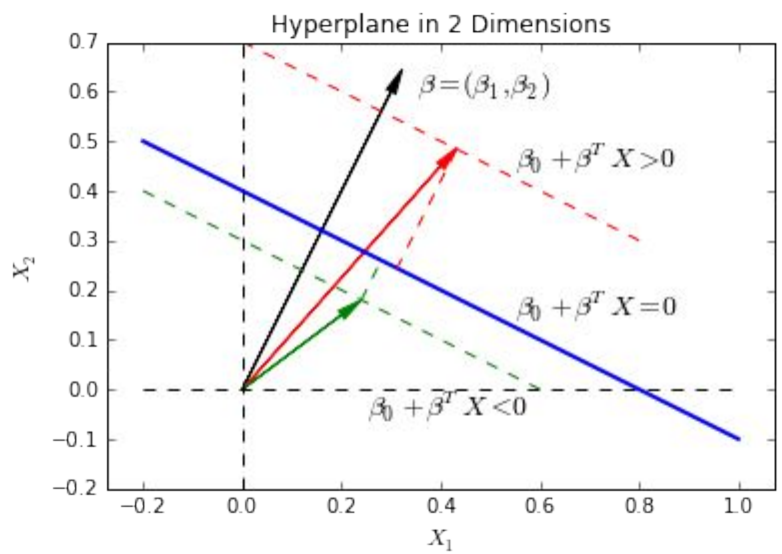
$$\beta_0 + X\beta^T$$

where $\beta = (\beta_1, \ldots \beta_p)$ and $X = (X_1, \ldots X_p)$ are *p-dimensional* vectors.

- In a 2-dimensional space, a hyperplane is nothing but a line and in a 3-dimensional space it is a plane.

Hyperplanes

- The coefficient vector β is called the **normal vector** - a vector orthogonal to all the movements within that hyperplane.
- In some cases we need to work with the normalized form: $\beta^* = \beta / |\beta|$ or equivalently, to require that $\sum_i^p \beta_i^2 = 1$.



Hyperplanes

- Here are some nice properties:
  - For any point x_0 lying in the hyperplane,

$$\beta^T x_0 = -\beta_0$$

  - The signed distance of any given point x to the hyperplane is given by:

$$f(x) = \frac{1}{|\beta|}(\beta^T x + \beta_0)$$

  - The signed distance function f can be used as the decision/discriminative function of the classification (explained below).

Separating Hyperplanes

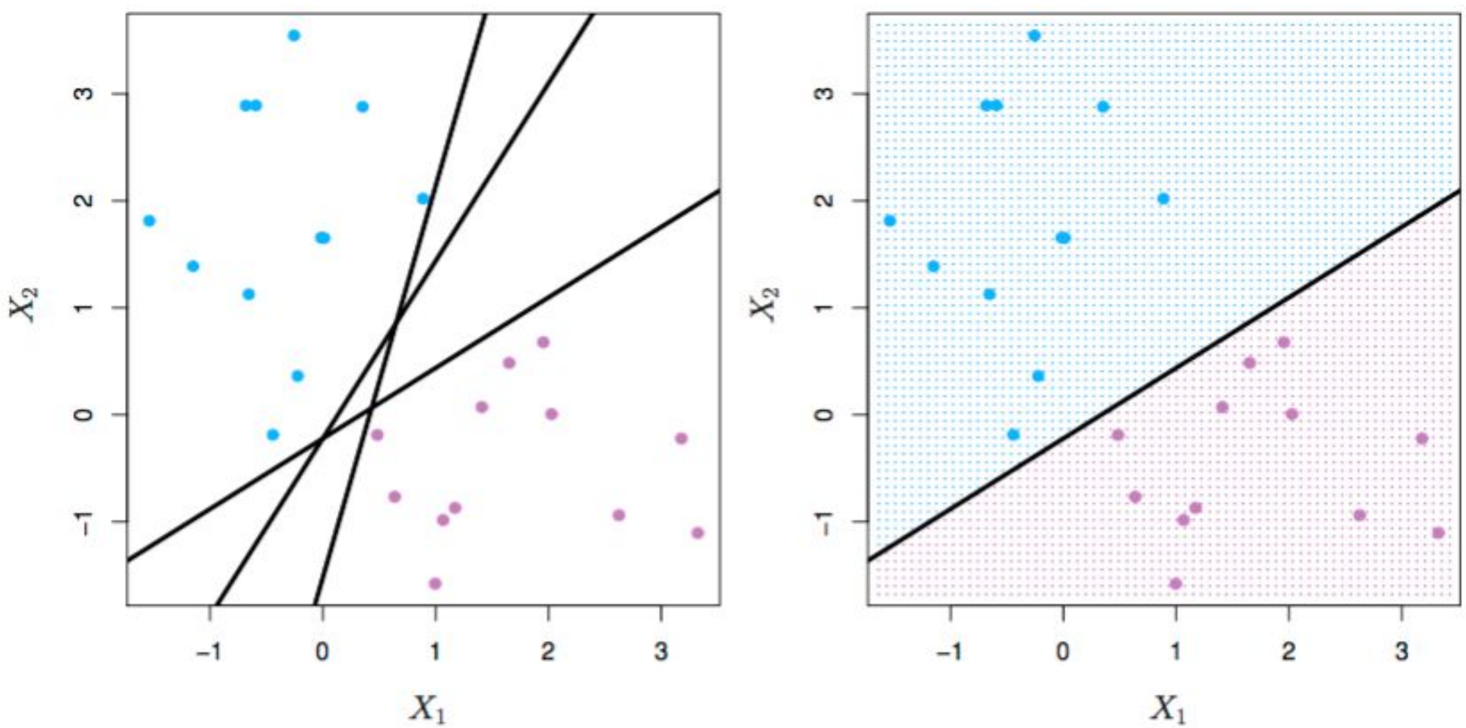
- If we assume that the data can be well *separated by a hyperplane* defined by $f(X) = \beta_0 + \beta^T X = 0$, then:
  - $f(X) > 0$, for points on one side of the hyperplane,
  - $f(X) < 0$, for points on the opposite side.
- If we code the two classes as:
  - $y = 1$, for $f(X) > 0$, and
  - $y = -1$, for $f(X) < 0$,

  then the distance multiplying the class becomes always positive:

$$y_i \cdot f(X_i) > 0$$

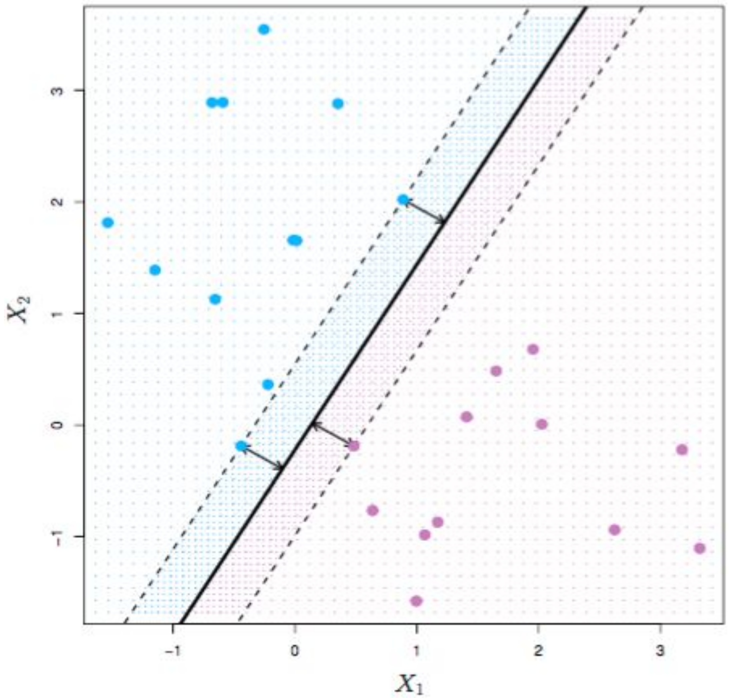
Separating Hyperplanes

- The figure on the left shows three of the infinitely many possible separating lines; the figure on the right shows that the values of function *f*(*X*) on the two sides of the hyperplane are of opposite signs.



Optimal Separating Hyperplanes

- Goal: to maximize the **margin**, defined by the minimal distance from the data points to a hyperplane, between the two classes on the training data.



- The data points that are used to determine the margins are called *support vectors*.

Maximal Margin Classifier

- The optimal separating hyperplane leads to a constrained optimization problem on margin *M*:

$$\max_{\beta_0, |\beta|=1} M$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq M, i = 1, ..., N$$

- The conditions ensure that the distances from all the points to the decision boundary/hyperplane specified by β and β_0 are at least *M*, and we seek the largest *M* by varying the parameters.

- We can get rid of the constraint $|\beta| = 1$ by replacing the inequalities with:

$$y_i(x_i^T \beta + \beta_0) \geq M|\beta|$$

Maximal Margin Classifier

- For any β and β_0 satisfying the inequalities, any positively scaled multiple satisfies them too.
- If we set $|\beta| = 1/M$, we can rephrase the original problem to a more elegant form by dropping the norm constraint on β:

$$\min_{\beta_0, \beta} \frac{1}{2} |\beta|^2$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq 1$$

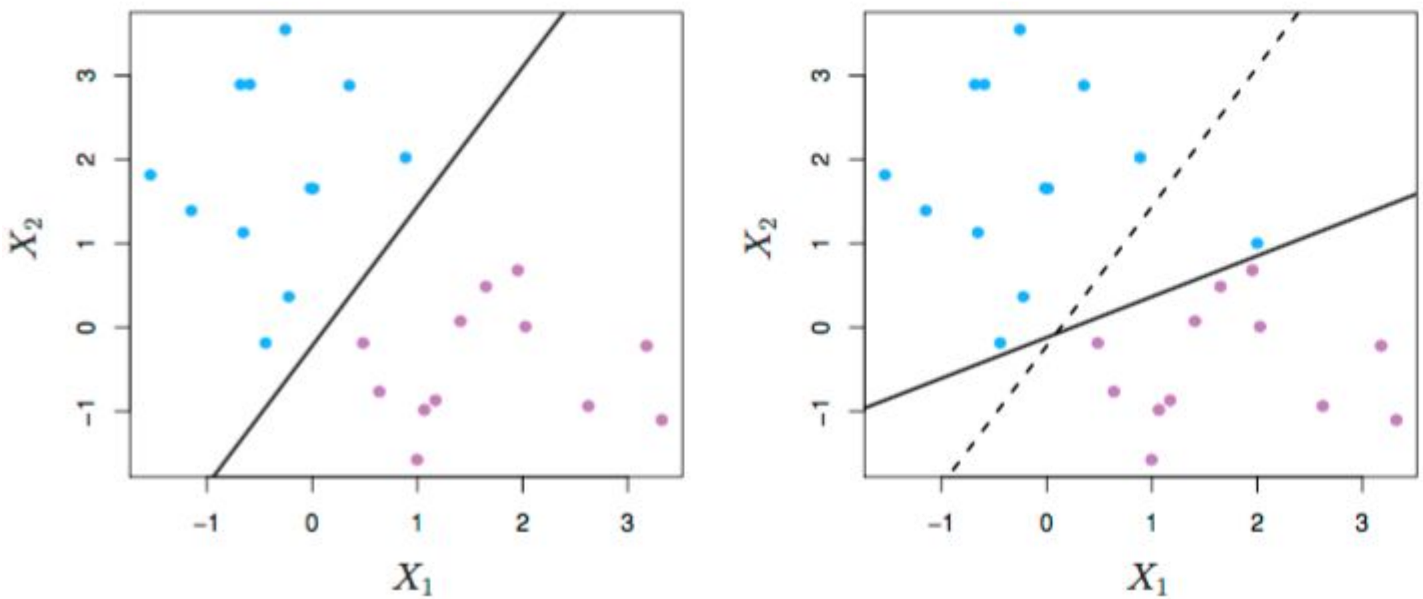
- This is a **convex quadratic optimization** problem, and can be solved efficiently. The same technique has been used in **Lasso** regression.
- It can be shown that the increasing of the maximal margin reduces the **model complexity** of the model.

Outline

- **Support Vector Machines**
  - **Separating Hyperplanes**
  - **The Support Vector Classifier**
  - **Kernels**
- **Support Vector Regression**
  - epsilon-insensitive loss function
  - Using kernels to approximate functions

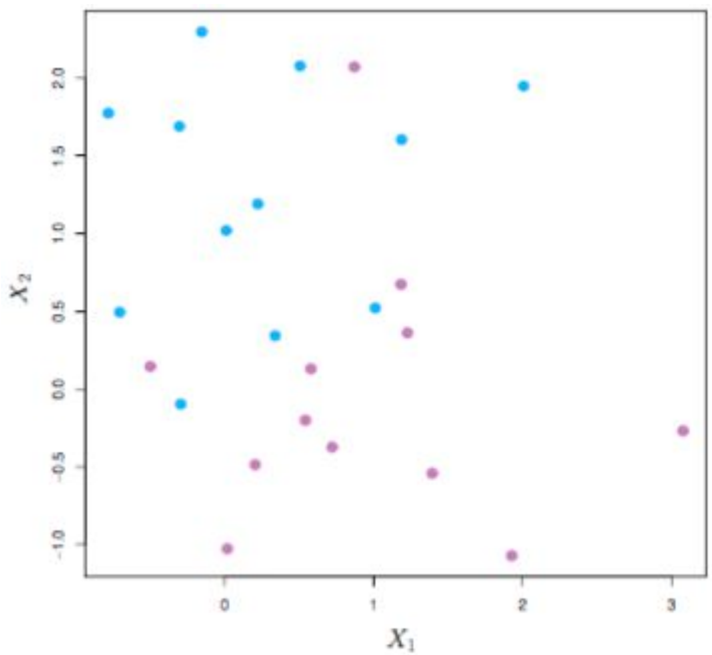
Noisy Data

- The technique for constructing an optimal separating hyperplane can be applied to cases of two perfectly separable classes.
- However, sometimes the data can be noisy, which can lead to a poor solution for the maximal margin classifier. (Note the one additional blue point on the right.)



Non-separable Data

- Even worse, quite often the data is not separable by a linear boundary.



- What shall we do?

The Support Vector Classifier

- To tolerate noise and classification errors, we maximize *M* but allow some points to be on the wrong side of the decision hyperplane.
- We introduce "slack" variables $\epsilon = (\epsilon_1, \ldots \epsilon_N)$ with constraints $\epsilon_i \geq 0$ and $\sum_i \epsilon_i \leq \text{Const}$. We modify the original optimization problem to be:

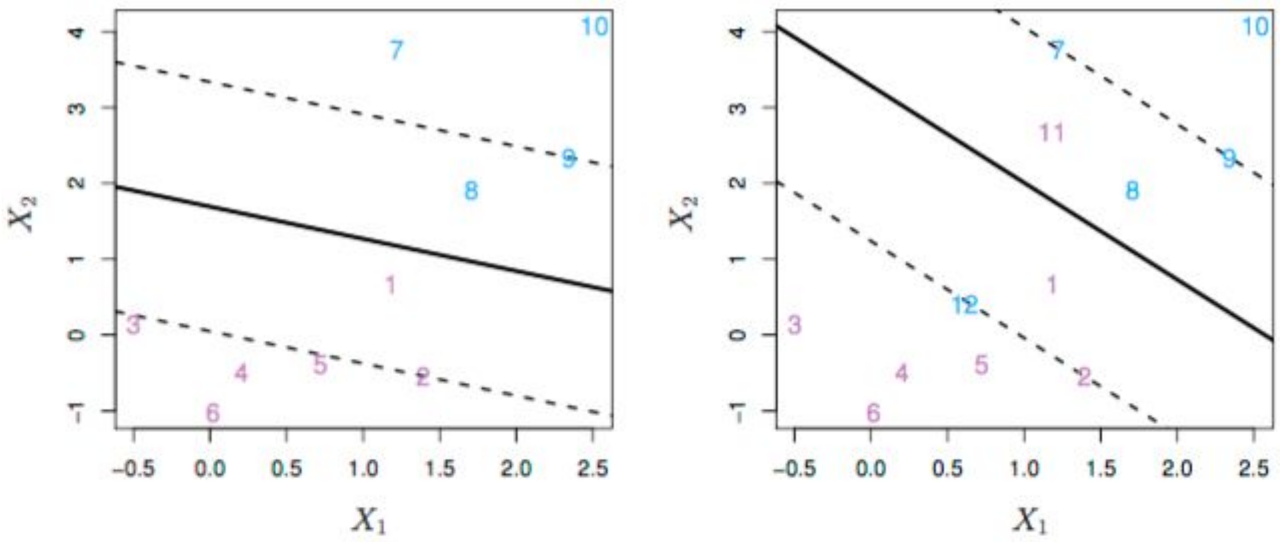
$$\max_{\beta_0, \epsilon, |\beta|=1} M$$

$$\text{subject to } \begin{cases} y_i(x_i^T \beta + \beta_0) \geq M(1 - \epsilon_i) \\ \epsilon_i \geq 0, \text{ and } \sum_i \epsilon_i \leq \text{Const} \end{cases}$$

  - ϵ_i are proportional to the degree by which the prediction is on the wrong side of their margin.
  - Misclassifications occurs when $\epsilon_i > 1$.

The Support Vector Classifier

- The data points that fall within the margin are penalized by the slack variables ϵ.



The Support Vector Classifier

- Computationally, it's convenient to use the following equivalent form:

$$\min_{\beta_0,\beta}\left(\frac{1}{2}|\beta|^2 + C\sum_{i=1}^{N}\epsilon_i\right)$$

$$\text{subject to} \begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \epsilon_i \\ \epsilon_i \geq 0 \end{cases}$$

where *C* is the penalty parameter of the total error term.

- The maximum margin classifier corresponds to $C = \infty$

Hinge Loss Function

- The constraint of the slack variables can be re-casted into the following inequalities

$$\begin{aligned} \epsilon_i \geq & \; 1 - y_i \cdot (\beta_0 + x_i^T \beta) \\ \epsilon_i \geq & \; 0 \end{aligned}$$

, which is equivalent to the combined inequalities

$$\epsilon_i \geq \; max(1 - y_i \cdot (\beta_0 + x_i^T \beta), 0)$$

- By setting

$$\begin{aligned} f(t) = & \; max(1 - t, 0) \\ t_i = & \; y_i \cdot (\beta_0 + x + i^T \beta); \forall 1 \leq i \leq N \end{aligned}$$

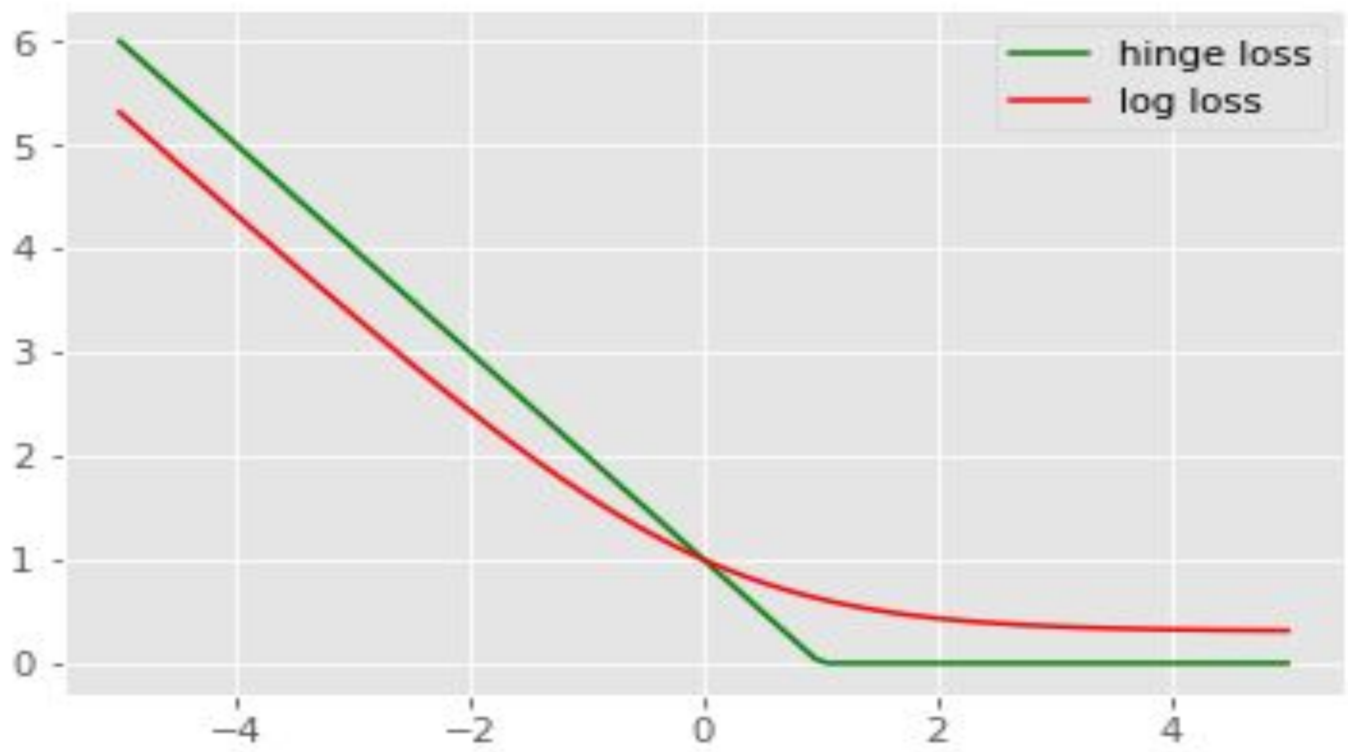
, these are the same as

$$\epsilon_i \geq f(t_i)$$

- f(t) is often called the hinge loss function

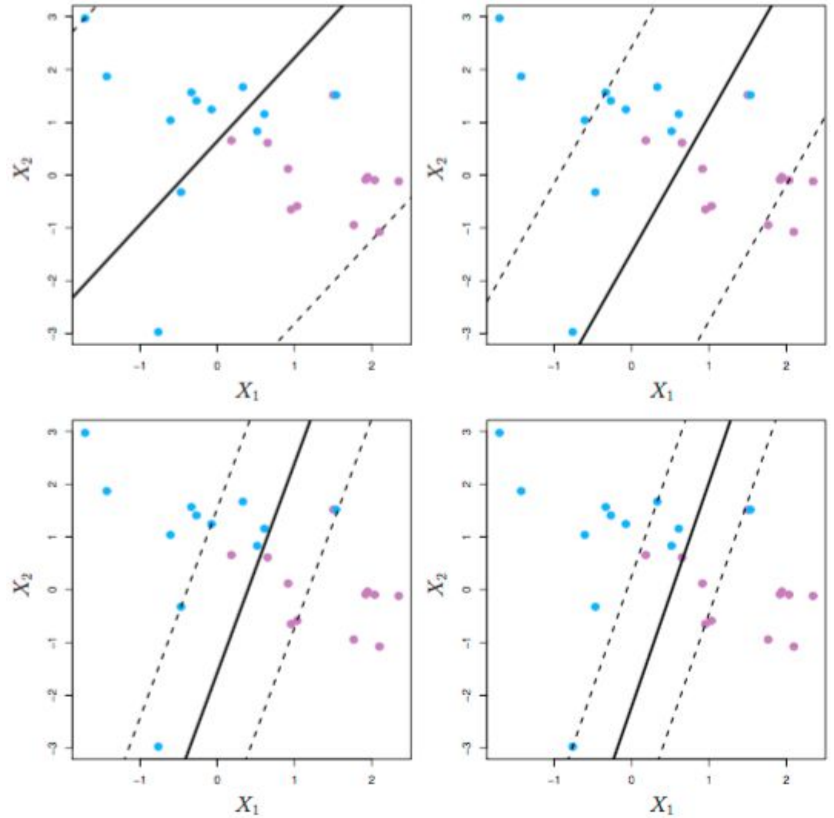
Hinge Loss vs Log Loss

- Hinge loss can be compared directly with log loss, the loss function used for logistic regression, after marking the binary class labels as -1 and 1, respectively
- As can be seen easily, hinge loss and log loss have similar asymptotic behaviors on both ends, but their differences cause the classifers to behave totally differently



Soft vs Hard Margin Classifiers

- If C is close to 0 then we have a *wide, soft margin*.
- If C is large then we are close to the *hard-margin* formulation.

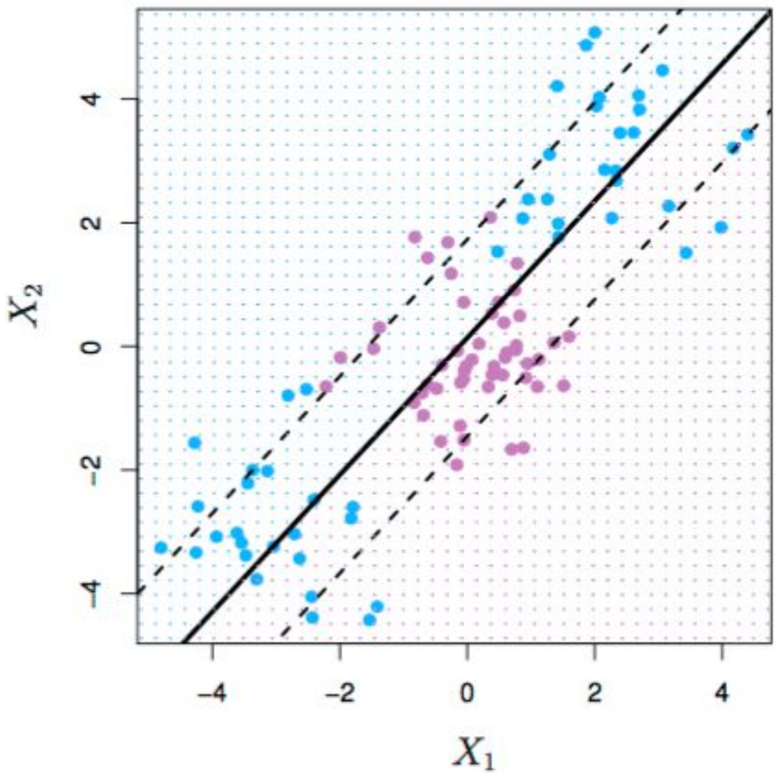


Outline

- **Support Vector Machines**
  - **Separating Hyperplanes**
  - **The Support Vector Classifier**
  - **Kernels**
- **Support Vector Regression**
  - Epsilon-insensitive loss function
  - Using kernels to approximate functions

Beyond Linearity

- The support vector classifier described so far finds linear decision boundaries in the feature space.
- In reality, it's very unlikely that the true boundary is actually linear in *X*.



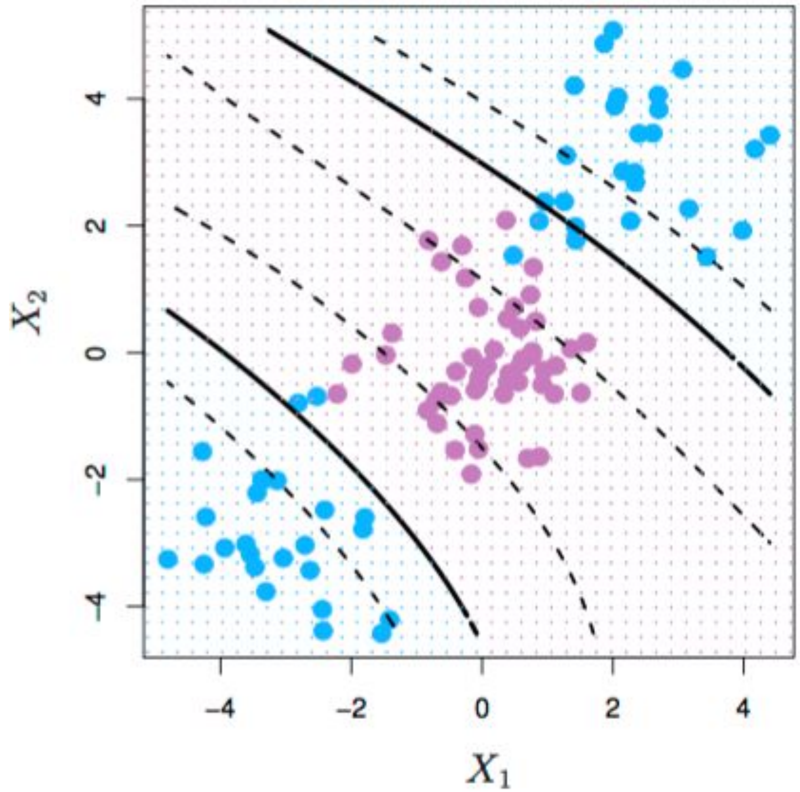
Basis Expansions

- If searching for linear boundary fails, we need to move beyond linearity.
- The core idea can be summarized as:
  - Expand the features *X* using basis expansions (such as polynomials),
  - use **SVC** in the enlarged space of derived input features,
  - then convert to nonlinear boundaries in the original space.
- Example:
  - We enlarge the feature space (X_1, X_2) to $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$
  - The nonlinear decision boundary is then determined by:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2, + \beta_3 X_1^2 + \beta_4 X_1 X_2 + \beta_5 X_2^2 = 0$$

Basis Expansions

- Back to the previous example, a basis expansion of cubic polynomials will create a nonlinear decision boundary in the original feature space.



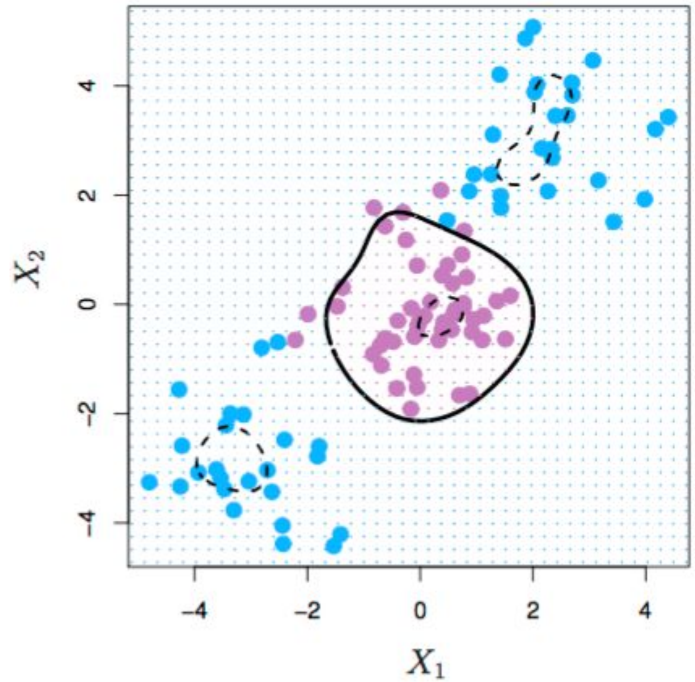
Kernels

- Some popular choices for **SVM** kernel functions are:
  - *d*th-Degree polynomial: $K(x, x') = (1 + \langle x, x' \rangle)^d$
  - Radial basis (**RBF**): $K(x, x') = \exp(-\gamma |x - x'|^2)$
- The higher the polynomial degree, the higher is the implicit target feature space dimension.
- RBF Kernel encodes a feature embedding into an infinite dimensional target linear space.
- By using the kernel function, the classification function can be rewritten as:

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(x, x_i)$$

Kernels

❖ The figure below shows the decision boundary with Radial (**RBF**) Kernel.



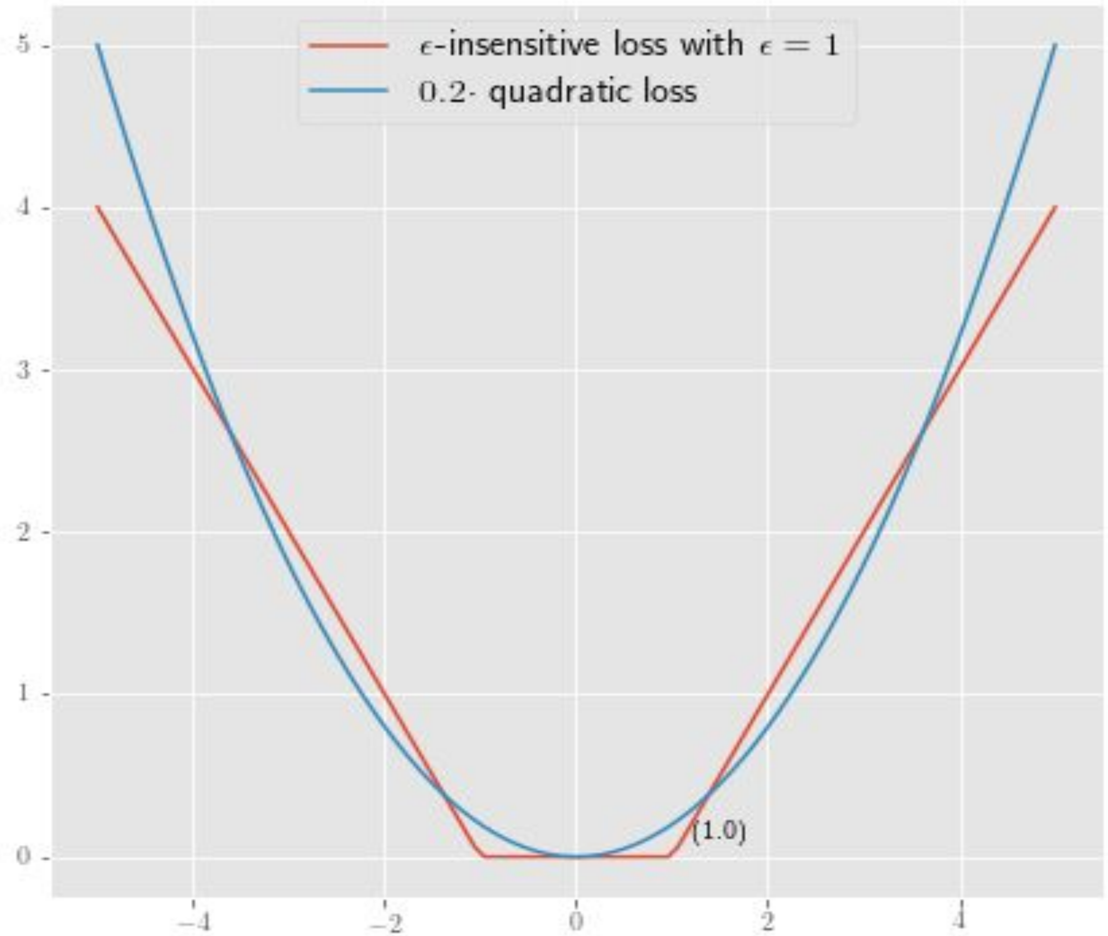
Outline

- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- **Support Vector Regression**
  - epsilon-insensitive loss
  - Using kernels to approximate functions

Support Vector Regression

- The **support vector machines** are advanced classification algorithms in the same ballpark as random forests and neural networks
- The idea can be modified to work with nonlinear regression, approximating a target function by a linear combination of nonlinear kernel functions
- Unlike the traditional regression problems which use **RSS** as the loss function, **SVR** uses the so-called **epsilon-insensitive** loss function to decide the optimal nonlinear approximation

Epsilon-Insensitive Loss Function vs Quadratic Loss



How Does A Sum of Kernels Approximate?

- Recall that in the **SVMs** session the kernel trick allows us to approximate the **decision function** f by

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(x, x_i)$$

, based on the **hinge loss**

- **SVR** follows the same methodology to approximate the **target function** f, based on the **epsilon-insensitive loss**

$$f(x) = \beta_0 + \sum_{i} \alpha_i K(x, x_i)$$

- **SVR's** iterative algorithm selects the support vectors, the optimal coefficients and the intercept for us

What Are SVR Support Vectors?

- Formally the right hand side sums over all the training samples
- Nevertheless the samples can be categorized into two types
- Either the error absolute value between the true target value and the predicted value differs by epsilon or less
- Or the error goes beyond epsilon (the error tolerance)
- For all the samples of the first type, their corresponding coefficients (so-called dual coefficients) vanish
- Only for the second type samples, their coefficients become nonzero. These samples are known as **SVR** support vectors
- **SVR** allows us to pay attention only to those samples whose errors go beyond control (bounded by epsilon)

How Does Epsilon-Insensitive Loss Work?

- The samples where their coefficients are nonzero are called **support vectors**
- The number **epsilon** is a tunable hyperparameter
- The **epsilon-insensitive loss function** penalizes the difference between the kernel-generated function and the true target values such that when their difference is within **epsilon** to each other, there is no loss penalty
- Beyond the epsilon difference, the penalty is by the absolute value of their difference (minus epsilon)
- Schematically we have

  Loss(f, y) = 0   if |f-y| is less than epsilon

  Loss(f, y) = |f-y| - epsilon   if |f-y| is larger than or equal to epsilon

- It is called **epsilon-insensitive** because the loss function ignores any errors smaller than **epsilon**

Comparing with SVMs

- In other words, the locations of the **support vectors** tend to be less sensitive to the samples where the true target and the proposed approximation differ slightly and focus on where they deviate a lot
- For the hard margin limit (large C), a **SVM**'s **support vectors** have to sit **ON** the boundaries of the margins
- For a soft margin **SVM** (lower C), the support vectors can penetrate to the wrong sides of the margins or even to the wrong sides of the decision boundaries
- For a hard margin **SVR** (with a large C), the support vectors lie at the boundaries the epsilon-neighborhood where the absolute difference of true values and predicted values are at most epsilon
- For a soft margin **SVR** (with a smaller C), the **support vectors** begin to penetrate and move out of the epsilon-band of 'small prediction errors' and get into their exterior

Tuning the Hyperparameter C

- As we tune the parameter C, effectively we decide the level of compromise between the flexibility we allow the errors to go beyond the epsilon threshold and the complexity of our **SVR** model
- The number of **support vectors** is a gauge of model complexity
- If we choose a C which turns out to be too large, the constraint on the function approximation becomes too rigid--sometimes there is no such solution

- On the other hand for a very small C, there can be a lot of **support vectors** away from the epsilon-band, which results in highly complicated regression models

Comparing with the RSS Loss

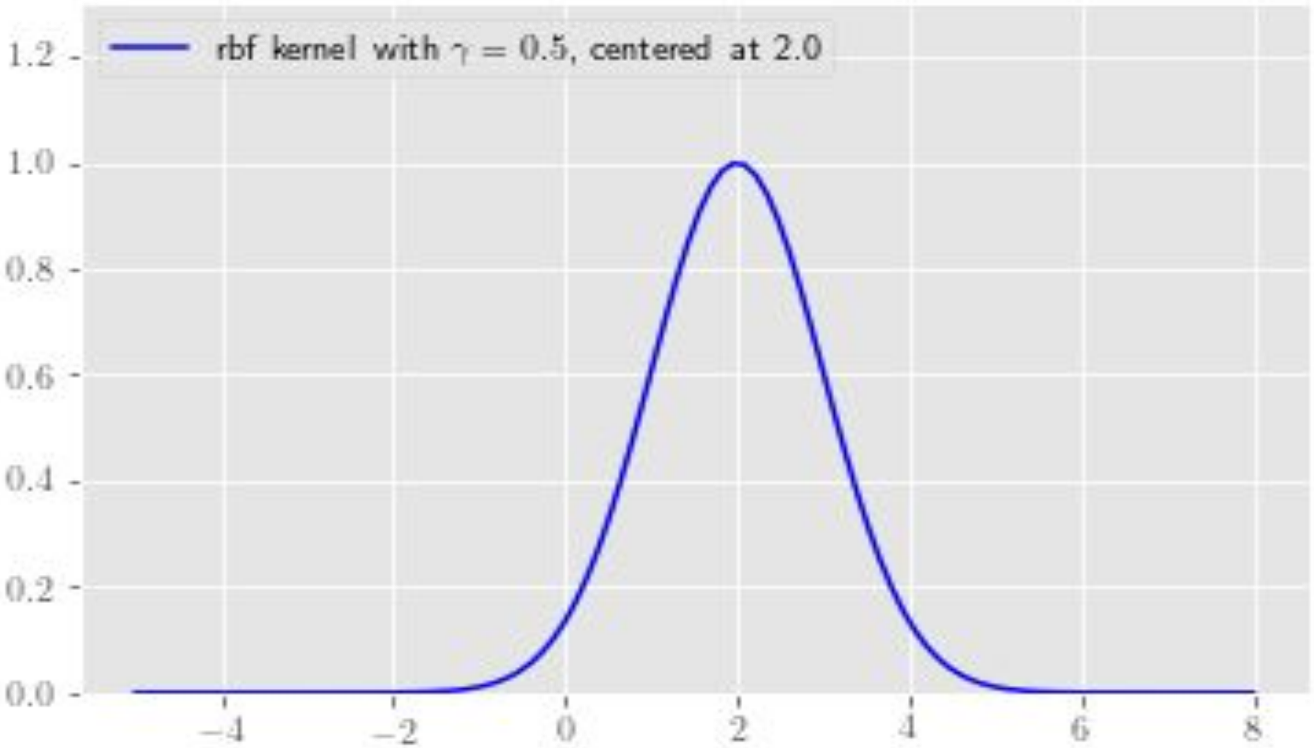
- The key difference between **epsilon-insensitive** loss and **RSS** loss is their asymptotic behaviors for large errors
- When the errors go large, the **RSS** loss penalizes quadratically while the epsilon-insensitive loss penalizes the errors linearly
- This difference of behavior makes **SVR** less sensitive to outliers than **RSS** loss based algorithms (like multiple linear regressions, trees)
- This large residual behavior is similar to a Huber linear regression

Outline

- **Support Vector Machines**
  - **Separating Hyperplanes**
  - **The Support Vector Classifier**
  - **Kernels**
- **Support Vector Regression**
  - epsilon-insensitive loss
  - Using kernels to approximate functions

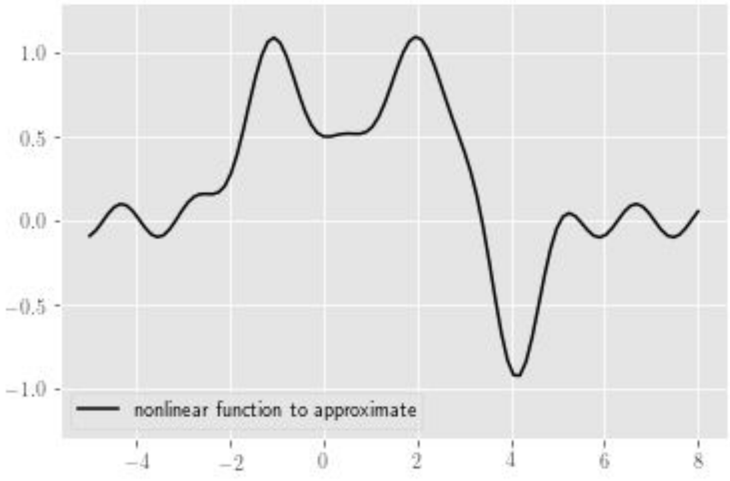
An Visualized Example of RBF Kernel

- An rbf kernel function is essentially proportional to a Gaussian pdf
- A larger gamma produces a tall-peaked bell curve
- The inverse square root of gamma measures the bell curve width

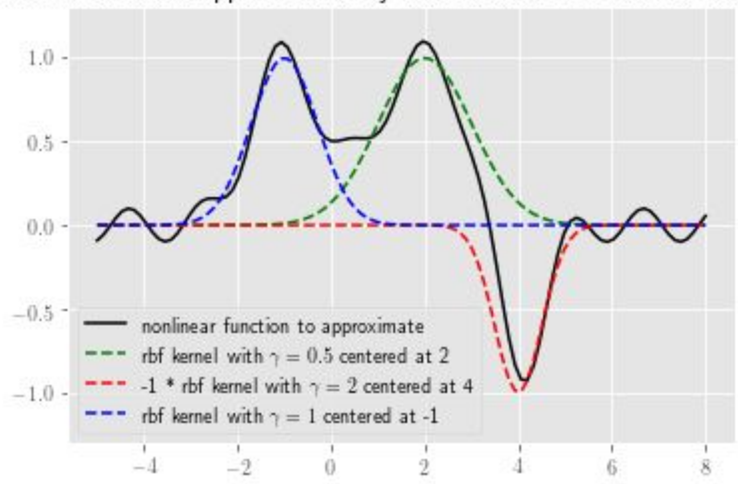


Using Kernels to Approximate Functions

- **SVR** is a special example of **kernel regression** where the model chooses the centers (**support vectors**) and amplitude (dual coefficients and intercept) to approximate a potentially highly nonlinear function



A nonlinear function approximated by the linear combination of 3 rbf kernels



Overfitting with SVR

- The larger the gamma hyperparameter, the more centers (support vectors) we use, the more expressive the model can be
- Then why don't we make the gamma of rbf kernel as big as possible, and add more **support vectors** to the kernel sum?
- Often an over-expressive model can fit the train set very well, as the **support vectors** are selected from the train set. But it can easily fit poorly on the unseen test set. This is a special case of the overfitting phenomenon
- Thus an **SVR** with **epsilon-insensitive loss** needs to avoid using too many support vectors and/or very large gamma to generate a overly complicated function

Hands-on Session

- Please go to the **"SVMs and SVRs"** in the lecture code.