

Machine Learning in Finance
Section I3
Lecture 8

Michael G Sotiropoulos

NYU Tandon
Deutsche Bank

29-Mar-2019

Separating Hyperplanes

- Geometry

- Support Vectors

SVM Classification

- Linear SVM

- Kernel SVM

SVM Extensions

- Multiclass

- Regression

Further Reading

Separating Hyperplanes: Geometry (I)

A hyperplane of a vector space \mathbb{R}^n is an $n - 1$ dimensional subspace containing all points x that satisfy the condition

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0 \quad (1)$$

The vector \mathbf{w} is normal to the hyperplane, and the constant b is called “bias”.

The signed distance of every point x outside the hyperplane is computed as

$$D(\mathbf{x}) = \mathbf{w}^T \cdot (\mathbf{x} - \mathbf{x}_0) / \|\mathbf{w}\| \quad (2)$$

where x_0 is any point inside the hyperplane, i.e. satisfying eq. (1).

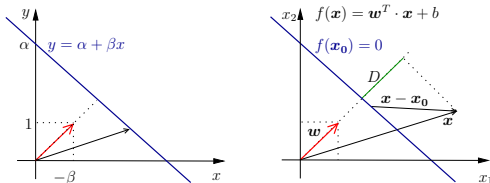


Figure: A 1D hyperplane (left). Signed distance D of x from hyperplane $f(\mathbf{x})$ (right)

A set of points with $D(\mathbf{x}) > 0$ lie on the same side of the hyperplane. This is the side to which \mathbf{w} points to. If $D(\mathbf{x}) < 0$ the points lie on the opposite side. From eq. (2) we see that

$$D(\mathbf{x}) = \frac{f(\mathbf{x})}{\|f'(\mathbf{x})\|} \quad (3)$$

- Given the hyperplane function f or equivalently (\mathbf{w}, b) , the sign of $f(\mathbf{x})$ classifies every point \mathbf{x} as belonging to either side of the hyperplane.

Suppose that the points \mathbf{x} are observations that belong to a binary class $C = \{0, 1\}$. An algorithm that computes a separating hyperplane (\mathbf{w}, b) can generate the fitted/predicted output class \hat{C} using the sign of $f(\mathbf{x})$ as the **decision function**

$$\hat{C} = \begin{cases} 1, & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b \geq 0 \\ 0, & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b < 0 \end{cases} \quad (4)$$

Separating Hyperplanes: Support Vectors (I)

If the points x are really separable, there can be infinitely many separating hyperplanes. We need a criterion for selecting a unique one.

Margin is the smallest distance from the hyperplane where the sign of $f(x)$ becomes either 1 or -1.

- ▶ The slope of the decision function is the length $\|w\|$
- ▶ The margin is inversely proportional to this slope

We want to minimize $\|w\|$ (maximize the margin) while either

1. having no points inside the margin (**hard margin classification**) or
2. minimize the points inside the margin (**soft margin classification**)

The points on the margin boundaries are the **support vectors**.

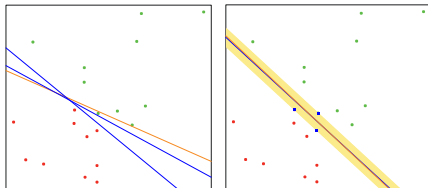


Figure: Separating hyperplanes (left), margin slab and support vectors (right). From [HTF].

Let's encode the positive class $c_i = 1$ for observation i with $t_i = 1$, and the negative class $c_i = 0$ with $t_i = -1$. Then the **hard margin** problem is

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} \quad (5)$$

$$\text{s.t. } t_i \left(\mathbf{w}^T \cdot \mathbf{x}_i + b \right) \geq 1, \quad i = 1, 2 \dots N \quad (6)$$

- ▶ We prefer using $\mathbf{w}^T \cdot \mathbf{w}$ instead of $\|\mathbf{w}\|$ because this makes the algorithm a quadratic optimization (QP) problem, which is solved efficiently
- ▶ The constraints require all points to be on the “correct” side
- ▶ If the points are not separable, the constraints become infeasible
- ▶ If the points are separable, the margin slab is $1/\|\mathbf{w}\|$ thick

The above problem can be expressed in terms of the Lagrange multipliers α_i by defining the Lagrangian function

$$L_p = \frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} - \sum_{i=1}^N \alpha_i \left[t_i \left(\mathbf{w}^T \cdot \mathbf{x}_i + b \right) - 1 \right] \quad (7)$$

To minimize L_p we set its gradient w.r.t. to \mathbf{w} and b to zero.

$$\mathbf{w} = \sum_{i=1}^N \alpha_i t_i \mathbf{x}_i, \quad \sum_{i=1}^N \alpha_i t_i = 0 \quad (8)$$

Substituting the above conditions back into eq. (7) we get the **dual** problem

$$\min_{\alpha_i} L_D = \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^N \alpha_i \alpha_k t_i t_k \mathbf{x}_i^T \cdot \mathbf{x}_k - \sum_{i=1}^N \alpha_i \quad (9)$$

$$\text{s.t. } \alpha_i \geq 0, \quad \sum_{i=1}^N \alpha_i t_i = 0 \quad (10)$$

Once the optimal α_i are found, eq. (8) gives the weights \mathbf{w} .

The solution must satisfy the conditions (Karush-Khun-Tucker)

$$\alpha_i \left[t_i \left(\mathbf{w}^T \cdot \mathbf{x}_i + b \right) - 1 \right] = 0, \quad i = 1, 2 \dots N \quad (11)$$

- ▶ if $\alpha_i > 0$ then $t_i (\mathbf{w}^T \cdot \mathbf{x}_i + b) = 1$, i.e. \mathbf{x}_i is a **support vector**
- ▶ if $t_i (\mathbf{w}^T \cdot \mathbf{x}_i + b) > 1$ then $\alpha_i = 0$, i.e. \mathbf{x}_i not on the margin boundaries

Support Vector Machines (SVM) are ML algorithms for classification. They compute the hyperplane (\mathbf{w} , b) by balancing two competing goals: *largest* possible margin with *smallest* possible number of misclassified points.

The **soft margin** problem is a modification of the hard margin eqs. (5), (6).

1. N “slack variables” ζ_i are introduced, they measure the allowed violation of the margin by the corresponding point \mathbf{x}_i
2. A “hyperparameter C is used to balance the two competing goals

The problem becomes

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} + C \sum_{i=1}^N \zeta_i \quad (12)$$

$$\text{s.t.} \quad t_i \left(\mathbf{w}^T \cdot \mathbf{x}_i + b \right) \geq 1 - \zeta_i, \quad i = 1, 2 \dots N \quad (13)$$

The Lagrange multiplier machinery goes through as before and the dual problem is solved as a constrained QP problem.

In Scikit-Learn, linear SVM classification can be done in three ways

1. Using the `sklearn.svm.LinearSVC` class
The most efficient for data sets that can be processed in batch
 2. Using the `sklearn.svm.SVC(kernel='linear')`
Equivalent to the previous one, but less efficient
 3. Using the `sklearn.linear_model.SGDClassifier(loss='hinge', alpha=1/(N*C))`
Suitable for huge data sets, online and out-of-core learning
The hinge loss function is $\max(1 - t, 0)$ (like put payoff with strike 1)
- ▶ All linear SVM classifiers compute linear boundaries.
 - ▶ SVC models do not output prediction probabilities. SVMs are purely geometric, not probabilistic models.
 - ▶ SVMs are sensitive to feature scaling. Standardize before usage.

For more details see the notebook `L06-LinearSVC-Examples.ipynb`

How do we go beyond linear classification?

We transform the features using basis functions (polynomial, Gaussian, ...). Hyperplanes in the transformed space are curves in the original feature space.

- ▶ Basis function expansion increases the dimensionality of the feature space
- ▶ This adds flexibility (reduces model bias) but becomes computationally expensive for large data sets
- ▶ The cost comes from computing $\phi(\mathbf{x})_i^T \cdot \phi(\mathbf{x})_k$ in the dual problem (9). The transformed feature vector $\phi(\mathbf{x})_i$ can be *much longer* than \mathbf{x}_i

The **kernel method** allows for nonlinear feature transformations while maintaining computational efficiency. The *trick* is to define the transformation in terms of a kernel function $K(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$ instead of the basis $\phi(\mathbf{x})$.

Example: two features, i.e. $\mathbf{x} = (x_1, x_2)$ and a quadratic transformation $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

$$\phi(\mathbf{x})^T \cdot \phi(\mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2 \quad (14)$$

Instead of computing $\phi(\mathbf{x})^T \cdot \phi(\mathbf{x}')$, we compute $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2$.

The kernel method works because of the following theorem (Mercer 1909)

- For any symmetric, positive semi-definite kernel $K(\mathbf{x}, \mathbf{x}')$, there exists an orthonormal basis of functions $\phi(\mathbf{x})$, such that $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \cdot \phi(\mathbf{x}')$

So, instead of specifying basis functions $\phi(\mathbf{x})$, we specify kernel $K(\mathbf{x}, \mathbf{x}')$.

Common choices are

Linear: $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \cdot \mathbf{x}'$

Polynomial: $K(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \cdot \mathbf{x}' + r)^d$

Radial Basis: $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$

Sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh\left(\gamma \mathbf{x}^T \cdot \mathbf{x}' + r\right)$

Predictions with kernel SVM

After solving the dual problem (9) using the transformed features we get the fitted coefficients $\hat{\alpha}_i$. They determine the fitted weights and bias (the separating hyperplane in the transformed space)

$$\hat{\mathbf{w}} = \sum_{i=1}^N \hat{\alpha}_i t_i \phi(\mathbf{x})_i \quad (15)$$

$$\hat{b} = \frac{1}{N} \sum_{i=1}^N \left(1 - t_i \hat{\mathbf{w}}^T \cdot \phi(\mathbf{x})_i \right) = \frac{1}{N} \sum_{i=1}^N \left(1 - t_i \sum_{j=1}^N \hat{\alpha}_j t_j K(\mathbf{x}_j, \mathbf{x}_i) \right) \quad (16)$$

The distance of the new observation \mathbf{x}' from the boundary becomes

$$D' = \hat{\mathbf{w}}^T \cdot \phi(\mathbf{x}') + \hat{b} = \sum_{i=1}^N \hat{\alpha}_i t_i K(\mathbf{x}_i, \mathbf{x}') + \hat{b} \quad (17)$$

Then we use the same decision function as in eq. (4) to forecast the class \hat{C}' .

1. We only need to compute the kernel $K(\mathbf{x}_i, \mathbf{x}_j)$, we never use the $\phi(\mathbf{x})$
2. In eq. (17) only the support vectors ($\hat{\alpha}_i \neq 0$) contribute to the forecast

In Scikit-Learn, kernel SVM classification can be done using `sklearn.svm.SVC(kernel='...')`

Common kernel choices are supported via `kernel='linear', 'poly', 'rbf', 'sigmoid'`

- ▶ Kernel parameters are passed via the arguments `gamma` (γ) and `coef0` (r)
- ▶ Custom kernels are supported by passing a callback function that takes an $N \times N$ matrix as input and returns a scalar

Computational complexity for N observations and M features scales as

class	complexity	kernel
LinearSVC	$\mathcal{O}(NM)$	NO
SVC	$\mathcal{O}(N^2M) - \mathcal{O}(N^3M)$	YES
SGDClassifier	$\mathcal{O}(NM)$	NO

So, SVC is suitable for medium sized data sets with large number of features and small number of sparse features.

For more details see the notebook `L06-KernelSVC-Examples.ipynb`

SVM is strictly a binary classifier. To extend from 2 to $K > 2$ classes, the following methods are used:

1. One-vs-one (OVO)

- ▶ $K(K - 1) / 2$ binary classifiers are trained, one for each class pair
- ▶ For prediction, we try all $K(K - 1) / 2$ classifiers and use the one that wins the most one-on-one duels
- ▶ This can be done with Scikit-Learn SVC via the argument `decision_function_shape='ovo'`
- ▶ The number of classifiers grows quadratically with K but each classifier is trained only on the subset of the two classes it tries to distinguish

2. One-vs-rest (OVR)

- ▶ K binary classifiers are trained, one for each class.
- ▶ For prediction, we try all K classifiers and use the one with the highest score
- ▶ This is the default in Scikit-Learn LinearSVC and SVC via the argument `decision_function_shape='ovr'`
- ▶ The number of classifiers grows linearly with K but each classifier is trained on the whole data set

Since SVC scales poorly with the input size N , we prefer the “OVO” method for large data sets.

SVM's can be used for regression. The trick is to reverse the objective.

- ▶ In SVM classification, we want to compute the widest possible margin, while having most observations outside the margin (few are misclassified)
- ▶ In SVM regression we want to compute the narrowest possible margin, such that most observations are inside (few are outliers)

Formally, given the regression function $f(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{w} + b$, we compute the weights by minimizing the loss function

$$L(\mathbf{w}, b) = \sum_{i=1}^N V(y_i - f(\mathbf{x}_i)) + \frac{\alpha}{2} \|\mathbf{w}\|^2 \quad (18)$$

where the “ ϵ -insensitive” error function is defined as

$$V_{\epsilon}(z) = \begin{cases} 0 & \text{if } |z| < \epsilon \\ |z| - \epsilon & \text{if } |z| \geq \epsilon \end{cases} \quad (19)$$

- ▶ It turns out that the loss minimizing weights (\mathbf{w}, b) depend on the features only via the inner products $\mathbf{x}_i^T \cdot \mathbf{x}_{i'}$ (the kernel trick works!)
- ▶ We can do non-linear SVM regression by using the previous kernels

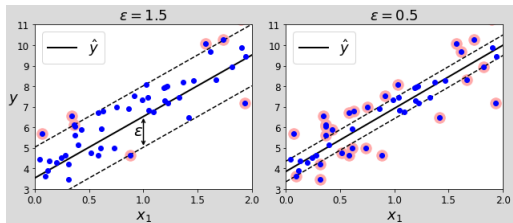


Figure: SVM Regression for two values of ϵ . From [GA]

In Scikit-Learn:

1. For Linear SVM regression use the `sklearn.svm.LinearSVR(epsilon=...)` class
 - ▶ The epsilon parameter defines the “ ϵ -insensitive” error function
2. For Kernel SVM regression use the `sklearn.svm.SVR(kernel=C, epsilon)` class
 - ▶ Common kernel choices are `kernel='linear', 'poly', 'rbf', 'sigmoid'`
 - ▶ The `C` parameter controls the soft margin, eq. (12), and the epsilon parameter controls the error function, eq. (19)

1. Chapters 4 and 12 in [HTF] have the mathematical details
2. Chapter 5 in [GA] has examples and practical advice
3. Chapter 4 in [VP] section “In-Depth: Support Vector Machines” has an interesting face recognition example

References

- [HTF] Hastie T., Tibshirani R. and Friedman J. “The Elements of Statistical Learning” 2ed. Springer, 2009
- [GA] Géron Aurélien. “Hands-On Machine Learning with Scikit-Learn and TensorFlow”. O'Reilly, 2017
- [VP] VanderPlas Jake. “Python Data Science Handbook”. O'Reilly, 2017