Machine Learning in Finance
Section I3
Lecture 8

Michael G Sotiropoulos

NYU Tandon
Deutsche Bank

29-Mar-2019

Contents

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

$$\boldsymbol{w}^T \cdot \boldsymbol{x} + b = 0 \tag{1}$$

The vector $\boldsymbol{w}$ is normal to the hyperplane, and the constant b is called "bias".

The signed distance of every point $\boldsymbol{x}$ outside the hyperplane is computed as

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

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

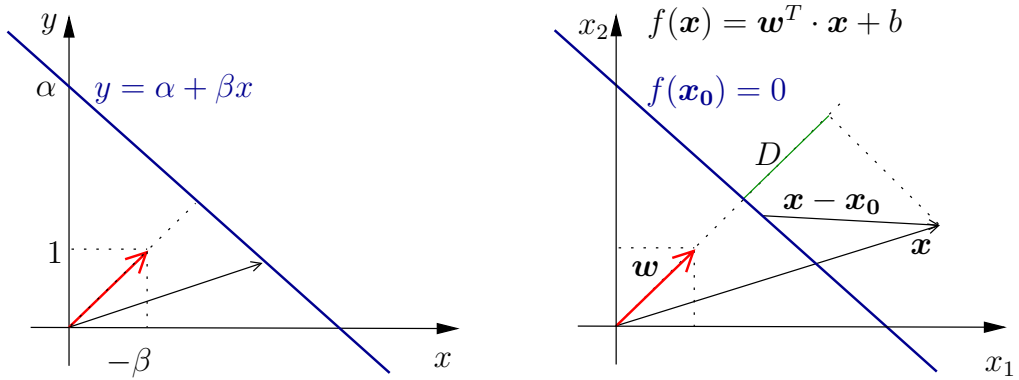

Figure: A 1D hyperplane (left). Signed distance D of $\boldsymbol{x}$ from hyperplane $f(\boldsymbol{x})$ (right)

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

$$D(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{\|f'(\boldsymbol{x})\|} \tag{3}$$

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

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

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

Separating Hyperplanes: Support Vectors (I)

If the points $\boldsymbol{x}$ 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 $f(\boldsymbol{x})$ becomes either 1 or -1.

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

We want to minimize $\|\boldsymbol{w}\|$ (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**.

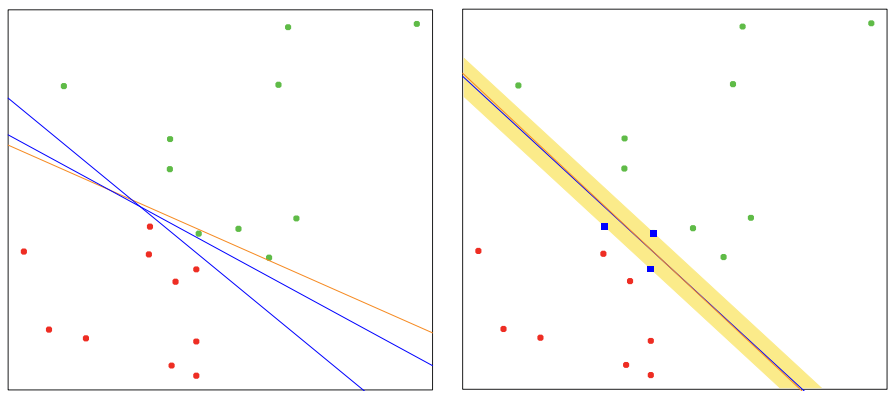

Figure: Separating hyperplanes (left), margin slab and support vectors (right). From [HTF].

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

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T \cdot \boldsymbol{w} \tag{5}$$

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

- We prefer using $\boldsymbol{w}^T \cdot \boldsymbol{w}$ instead of $\|\boldsymbol{w}\|$ 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 $1/\|\boldsymbol{w}\|$ thick

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

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

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

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i t_i \boldsymbol{x}_i, \quad \sum_{i=1}^{N} \alpha_i t_i = 0 \tag{8}$$

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

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

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

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

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

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

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

Support **V**ector **M**achines (SVM) are ML algorithms for classification.
They compute the hyperplane $(\boldsymbol{w}, b)$ 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 $\boldsymbol{x}_i$
2. A "hyperparameter C is used to balance the two competing goals

The problem becomes

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T \cdot \boldsymbol{w} + C\sum_{i=1}^{N}\zeta_i \tag{12}$$

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

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 $\phi(\boldsymbol{x})_i^T \cdot \phi(\boldsymbol{x})_k$ in the dual problem (9). The transformed feature vector $\phi(\boldsymbol{x})_i$ can be *much longer* than $\boldsymbol{x}_i$

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 $K(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}$ instead of the basis $\phi(\boldsymbol{x})$.

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

$$\phi(\boldsymbol{x})^T \cdot \phi(\boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^2 \tag{14}$$

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

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

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

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

Common choices are

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

$$\text{Polynomial:} \quad K(\boldsymbol{x}, \boldsymbol{x}') = \left(\gamma \boldsymbol{x}^T \cdot \boldsymbol{x}' + r\right)^d$$

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

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

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)

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

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

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

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

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 $K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$, we never use the $\phi\left(\boldsymbol{x}\right)$
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 $f(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \boldsymbol{w} + b$, we compute the weights by minimizing the loss function

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

where the "ϵ-insensitive" error function is defined as

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

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

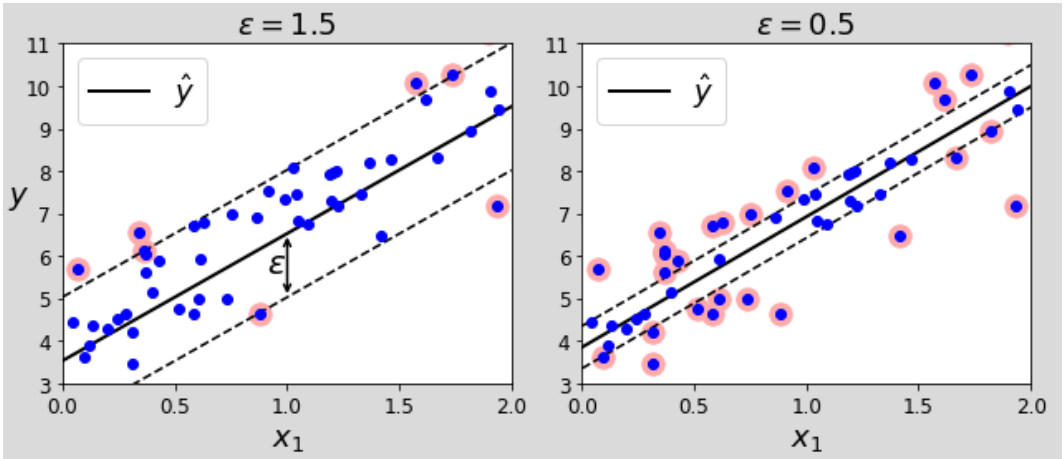

Figure: SVM Regression for two values of ϵ. From [GA]

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)

Further Reading

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