Support Vector Machine (SVM)

Recall from §1: hard-margin SVM

- A hard margin support vector machine (SVM) addresses the case where binary-labeled subsets of examples \( \{(x_i, y_i)\}_{i=1}^{N} \) are linearly separable.
- It finds a linear decision hyperplane \( wx + b = 0 \) (center of a road) that separates +1 examples from −1 examples by minimizing \( ||w|| \) subject to \( y_i(wx_i+b) \geq 1 \) for \( i = 1, \ldots, N \).
- Minimizing \( ||w|| \) maximizes \( \frac{2}{||w||} \), the distance from the +1 margin \( wx_i + b = 1 \) (one edge of road) to the −1 margin \( wx_i + b = -1 \) (other edge).

Two problems and §3 solutions (below):

- Outlying examples may violate the boundary (noise):
  
  A soft margin SVM can include a hyperparameter (parameter controlling learning; not trained) that penalizes misclassification of outliers (positives on the negative side of the boundary or negatives on the positive side).

- For some data, the +1 and −1 examples cannot be separated by a hyperplane but could be separated by a polynomial or other nonlinear boundary: an SVM can include a kernel that allows a nonlinear decision boundary.

---

1Recall: Burkov uses \( wx - b = 0 \). I use \( wx + b = 0 \) to match scikit-learn.

2Burkov and the user guide (linked below) use \( y \in \{-1, 1\} \). scikit-learn seems also to allow \( y \in \{0, 1\} \).
Soft Margin SVM

To handle noise, use the hinge loss, \( \max(0, 1 - y_i^*(w^T x_i + b)) = \begin{cases} 0, & \text{if constraint } y_i^*(w^T x_i + b) \geq 1 \text{ is satisfied} \\ \text{proportional to distance from constraint edge, otherwise} \end{cases} \)

e.g. Here are a few typical hinge losses (pictured for \( w = 1, b = 0 \)):

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Above the ( w^T x + b = 1 ) margin</td>
<td>( y_i^*(w^T x_i + b) \geq 1 ) satisfied</td>
</tr>
<tr>
<td>On the ( w^T x + b = 1 ) margin</td>
<td>( y_i^*(w^T x_i + b) \geq 1 ) just satisfied</td>
</tr>
<tr>
<td>On the ( w^T x + b = 0 ) decision boundary</td>
<td>( y_i^*(w^T x_i + b) = 1 ) (constraint failed; neutral decision)</td>
</tr>
<tr>
<td>On the ( w^T x + b = -1 ) margin</td>
<td>( y_i^*(w^T x_i + b) = 1 )</td>
</tr>
<tr>
<td>Below the ( w^T x + b = -1 ) margin</td>
<td>( y_i^*(w^T x_i + b) &lt; 1 ) (wrong decision)</td>
</tr>
</tbody>
</table>

Hard margin SVM’s minimizing \( ||w|| \) is equivalent to minimizing \( \frac{1}{2}||w||^2 \).

To get a soft margin SVM, we minimize the cost function

\[
\frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i^*(w^T x_i + b))
\]

where hyperparameter \( C \) decides the tradeoff between increasing the separation between +1 and -1 examples and ensuring that each \( x_i \) is on the correct side.

- Increasing \( C \) yields a smaller margin and classifies training examples better.
- Decreasing \( C \) yields a larger margin suited to noisy data, making more errors on training examples but generalizing better to new (test) examples.

Nonlinear Boundary

- For a nonlinear boundary, we may be able to use some \( \phi : \mathbf{x} \mapsto \phi(\mathbf{x}) \) to transform the feature space into a higher-dimension space in which we can make a linear separation.

e.g. In the right-hand figure above, the concentric red and blue rings are not linearly separable. However, we can use \( \phi(\mathbf{x}) = \phi(p, q) = (p^2, \sqrt{2pq}, q^2) \) to map each 2D point to a 3D point:

\[3\text{Burkov puts the } C \text{ on the } \frac{1}{2}||w||^2 \text{ term. I put it on the hinge loss term to match scikit-learn.}\]
In 3D, the red and blue points are linearly separable by a plane. Transforming data and trying several $\phi(x)$ functions while seeking linear separation in a higher dimension can be computationally expensive.

- Using a function to make a transformation implicitly is called the kernel trick and can be much less expensive. It facilitates a nonlinear boundary in the original feature space.

SVM’s (hard-margin) minimization is implemented with the help of Lagrange multipliers, finding $\max_{\alpha_1, \ldots, \alpha_N} \left( \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} y_i \alpha_i (x_i x_k) y_k \alpha_k \right)$ subject to $\sum_{i=1}^{N} \alpha_i y_i = 0$ and $\alpha_i \geq 0$ for $i = 1, \ldots, N$, which can be done with a quadratic programming algorithm.

Here the feature vectors appear only as the dot product $x_i x_k$.

- Instead of finding $\phi(x_i)$ and $\phi(x_k)$ and then $\phi(x_i) \phi(x_k)$ (expensive), we can use some kernel function $k(x_i, x_k)$ that is a function only of $x_i$ and $x_k$ (untransformed) and gives the same result, with no explicit application of $\phi()$ necessary.

e.g. Instead of mapping $x_i$ to $\phi(x_i) = \phi(p_i, q_i) = (p_i^2, \sqrt{2} p_i q_i, q_i^2)$ and $x_k$ to $\phi(x_k) = \phi(p_k, q_k) = (p_k^2, \sqrt{2} p_k q_k, q_k^2)$ and then finding their 3D dot product $x_i x_k = p_i^2 p_k^2 + 2 p_i p_k q_i q_k + q_i^2 q_k^2$, we can find the 2D dot product $p_i p_k + q_i q_k$ and square it to get the same result. This is an example of the kernel trick using a quadratic kernel $k(x_i, x_k) = (x_i x_k)^2$.

The most widely used of the many kernel functions is the radial basis function (RBF) kernel,

$$k(x, x') = \exp \left( -\frac{||x - x'||^2}{2\sigma^2} \right) \quad \text{(Burkov’s notation)}$$

$$= \exp \left( -\gamma ||x - x'||^2 \right) \quad \text{(scikit-learn’s notation, with } \gamma > 0),$$

where the fraction’s numerator is the squared Euclidean distance between the two vectors. Burkov asserts the space for the RBF kernel has $\infty$ dimensions. Varying the hyperparameter $\gamma = \frac{1}{2\sigma^2}$ chooses between a smooth or curvy boundary in the original feature space.
Python

- `from sklearn import svm` (as in §01)
- `clf = svm.SVC(kernel='linear', C=1)` (as in §01)

High C gives a hard-margin SVM (as in §01). Low C allows a larger margin on noisy data. Start with the default $C = 1$. Decrease to generalize better from noisy data. Increase for a narrower margin and better training performance.

- `clf = svm.SVC(kernel="rbf", C=1, gamma='scale')` uses the (RBF) kernel trick to allow a nonlinear decision boundary in the feature space.
  - `gamma='scale'` uses $\gamma = \frac{1}{D \cdot X.\text{var}()}$
  - `gamma='auto'` uses $\gamma = \frac{1}{D}$
  - or set `gamma` to a float

The user guide says “$\text{gamma}$ defines how much influence a single training example has. The larger $\text{gamma}$ is, the closer other examples must be to be affected.” A larger $\text{gamma}$ allows more boundary curvature and more overfitting of training data.

- `clf.fit(X, y)` fits the model to array $X_{N \times D}$ and $y_{N \times 1}$ (as in §1)
- `clf.coef_` gives $w^*$ and `clf.intercept_` gives $b^*$ (as in §1)
- `clf.predict(X)` does classification on examples in $X$ (as in §1)
- `clf.predict_proba(X)[:, 1]` gives probabilities $\{P(y_i = 1)\}$ ($[:, 0]$ gives $\{P(y_i = 0)\}$)
- `clf.score(X, y)` gives the average accuracy on $X$ with respect to $y$ (as in §1)

To learn more:

- Advice on setting $C$ and $\text{gamma}$ for a nonlinear boundary is at [https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html](https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html)
- The kernel trick is described at [https://en.wikipedia.org/wiki/Kernel_method](https://en.wikipedia.org/wiki/Kernel_method)

---

4The guide says SVM is not scale invariant, so we should scale each feature to $[0, 1]$ or $[-1, 1]$ or standardize each feature to have mean 0 and variance 1. We will discuss scaling in §05.