A survey of some Machine Learning models

Chris Cornwell

April 1, 2025

Outline

Support Vector Machines

Neural Networks

Setup

Similar to a logistic model, a **support vector machine** is a model for binary classification, using a hyperplane, of the form $\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{w} \cdot \mathbf{x} + b = 0\}$, as decision boundary.

However, the optimization goal is different.

[1]The multiple $\frac{1}{2}$ is non-essential here, but the convention is to include it.

[2]Recall, we pointed out how this is possible when the data is linearly separable.

Setup

Similar to a logistic model, a **support vector machine** is a model for binary classification, using a hyperplane, of the form $\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{w} \cdot \mathbf{x} + b = 0\}$, as decision boundary.

However, the optimization goal is different.

Given sample data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the goal is to minimize the value of $\frac{1}{2}|\mathbf{w}|^2$, the vector norm[1], subject to the condition that for all $1 \leq i \leq n$, $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ is satisfied.[2]

To work with a data set that is not linearly separable, one introduces so-called "slack variables" $\xi_i \geq 0$, $i = 1, \ldots, n$ into the inequalities. They change to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$.

The reason for wanting to minimize $\frac{1}{2}|\mathbf{w}|^2$?

- Supposing no $\mathbf{x}_i$ passes through hyperplane with parameters $\mathbf{w}, b$, we can scale both the normal vector and b so that $\min_i |\mathbf{w} \cdot \mathbf{x}_i + b| = 1$.
- The distance from any $\mathbf{x} \in \mathbb{R}^d$ to the hyperplane is $\frac{|\mathbf{w} \cdot \mathbf{x} + b|}{|\mathbf{w}|}$. So, if $\mathbf{x}_i \in \mathcal{S}$ is such that $|\mathbf{w} \cdot \mathbf{x}_i + b| = 1$, then its distance to decision boundary is $\rho = \frac{1}{|\mathbf{w}|}$.
- Want to maximize distance to decision boundary, so want to minimize $|\mathbf{w}|$.

[1]The multiple $\frac{1}{2}$ is non-essential here, but the convention is to include it.

[2]Recall, we pointed out how this is possible when the data is linearly separable.

Constrained minimization and SVM

We can understand minimizing $\frac{1}{2}|\mathbf{w}|^2$ subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ with a Lagrangian. (Method of Lagrange multipliers; see Section 7.2 in the Mathematics for Machine Learning book.)

For $\underline{\alpha} = (\alpha_1, \ldots, \alpha_n)$, with $\alpha_i \in \mathbb{R}$, Lagrangian is

$$L(\mathbf{w}, b, \underline{\alpha}) = \frac{1}{2}|\mathbf{w}|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right).$$

It is minimized when

$$\begin{aligned}
\nabla_{\mathbf{w}} L = 0 \quad &\Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i; \\
\nabla_b L = 0 \quad &\Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0; \\
\alpha_i \left(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right) = 0 \quad &\Rightarrow \quad \alpha_i = 0 \quad \text{OR} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1.
\end{aligned}$$

Support vectors are those $\mathbf{x}_i$ for which $\alpha_i \neq 0$, and so $\mathbf{w} \cdot \mathbf{x}_i + b = \pm 1$.

Lagrangian Duality

Something interesting happens when we convert the previous Lagrangian optimization problem into its "Lagrangian dual problem." This means that we take the minimum solution for $\mathbf{w}$, put it into $L(\mathbf{w}, b, \underline{\alpha})$ and want multipliers $\alpha_i \geq 0$ that *maximize* the value of this. That is, maximize

$$\frac{1}{2}\left|\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j \right) \cdot \mathbf{x}_i + y_i b - 1 \right).$$

Rearranged, you can rewrite it:

$$\max_{\underline{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

subject to $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.
This optimization problem only depends on knowing $\mathbf{x}_i \cdot \mathbf{x}_j$ for each (i, j), and this leads to what are called **kernel methods** that are very computationally efficient and allow one to use SVM models that have non-linear decision boundaries.

SVMs via Gradient Descent

An alternative for optimizing an SVM classifier is to do so with a loss function. The loss function has a fair amount of similarity to the log-loss function we used in logistic regression; however, the per-example losses use a piecewise linear function.

When $y_i = 1$ then, writing $z_i = \mathbf{w} \cdot \mathbf{x}_i + b$, the per-example loss is $C \max(1 - z_i, 0)$ for some constant C. Call this $C\mathrm{cost}_1(z_i)$. When $y_i = -1$ (and so $\tilde{y}_i = 0$) then the per-example loss is $C\mathrm{cost}_0(z_i) = C \max(1 + z_i, 0)$.

However, we also include the norm of $\mathbf{w}$ in the loss:

$$\mathcal{L}_{\mathcal{S}}(\mathbf{w}, b) = \frac{1}{2}|\mathbf{w}|^2 + \frac{1}{n}\sum_{i=1}^{n} C\left(\tilde{y}_i \mathrm{cost}_1(\mathbf{w} \cdot \mathbf{x}_i + b) + (1 - \tilde{y}_i)\mathrm{cost}_0(\mathbf{w} \cdot \mathbf{x}_i + b)\right).$$

Outline

Single Layer

Fix a dimension d and a function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ (feel free to think of this as the logistic function, for now). For any $\omega = (\mathbf{w}, b) \in \mathbb{R}^{d+1}$, set $f_\omega(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$.

Single Layer

Fix a dimension d and a function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ (feel free to think of this as the logistic function, for now). For any $\omega = (\mathbf{w}, b) \in \mathbb{R}^{d+1}$, set
$f_\omega(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$.
Given some collection $\omega_1 = (\mathbf{w}_1, b_1), \omega_2 = (\mathbf{w}_2, b_2), \ldots, \omega_m = (\mathbf{w}_m, b_m)$, define $F : \mathbb{R}^d \longrightarrow \mathbb{R}^m$ by setting

$$F(\mathbf{x}) = (f_{\omega_1}(\mathbf{x}), f_{\omega_2}(\mathbf{x}), \ldots, f_{\omega_m}(\mathbf{x})).$$

Such a function F represents a single “layer map” of a neural network, with **activation function** σ.

Single Layer

Fix a dimension d and a function $\sigma : \mathbb{R} \to \mathbb{R}$ (feel free to think of this as the logistic function, for now). For any $\omega = (\mathbf{w}, b) \in \mathbb{R}^{d+1}$, set $f_\omega(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$.
Given some collection $\omega_1 = (\mathbf{w}_1, b_1), \omega_2 = (\mathbf{w}_2, b_2), \ldots, \omega_m = (\mathbf{w}_m, b_m)$, define $F : \mathbb{R}^d \to \mathbb{R}^m$ by setting

$$F(\mathbf{x}) = (f_{\omega_1}(\mathbf{x}), f_{\omega_2}(\mathbf{x}), \ldots, f_{\omega_m}(\mathbf{x})).$$

Such a function F represents a single "layer map" of a neural network, with **activation function** σ.
If we use W to denote the $m \times d$ matrix with $\mathbf{w}_i$ as row i, and $\mathbf{b} = (b_1, \ldots, b_m)$, then we can write $F(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b})$ (*if we allow σ to be applied to a vector by applying the function to each component of the vector*).
W: called the "weight matrix" for the layer map; $\mathbf{b}$ is called the "bias vector."

Composing layers

A neural network is the result of composing some number of layer maps. That is, let $m_0, m_1, \ldots, m_L$ be positive integers and say that $F_1, F_2, \ldots, F_L$ are each layer maps (as in the previous slide), with $F_i : \mathbb{R}^{m_{i-1}} \longrightarrow \mathbb{R}^{m_i}$.

- so, for each $1 \leq i \leq L$, there is a weight matrix W_i (which is $m_i \times m_{i-1}$) and bias vector $\mathbf{b}_i \in \mathbb{R}^{m_i}$.

Composing layers

A neural network is the result of composing some number of layer maps. That is, let $m_0, m_1, \ldots, m_L$ be positive integers and say that $F_1, F_2, \ldots, F_L$ are each layer maps (as in the previous slide), with $F_i : \mathbb{R}^{m_{i-1}} \longrightarrow \mathbb{R}^{m_i}$.

- so, for each $1 \leq i \leq L$, there is a weight matrix W_i (which is $m_i \times m_{i-1}$) and bias vector $\mathbf{b}_i \in \mathbb{R}^{m_i}$.

The *fully connected* neural network, with activation function σ, associated to this collection of weight matrices and bias vectors is the parameterized function

$$f(\mathbf{x}) = F_L \circ F_{L-1} \circ \ldots \circ F_2 \circ F_1(\mathbf{x}).$$

The parameters for the function are $(W_1, \mathbf{b}_1, W_2, \mathbf{b}_2, \ldots, W_L, \mathbf{b}_L)$ (flattened out as a vector).

Example with Logistic model

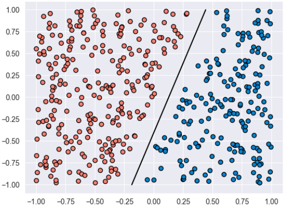
Given ± 1-labeled sample data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we found an expression for each partial derivative of $\mathcal{L}_\mathcal{S}$. In fact, if we call $w_{d+1} = b$ and $x_{i,d+1} = 1$ for every $1 \le i \le n$, then

$$\frac{\partial}{\partial w_j}\mathcal{L}_\mathcal{S} = \frac{1}{n}\sum_{i=1}^{n} -\tilde{y}_i x_{i,j}(1 - f_\omega(\mathbf{x}_i)) + (1-\tilde{y}_i)x_{i,j} f_\omega(\mathbf{x}_i)$$

for every $1 \le j \le d+1$.

Example (cont'd): Batch gradient descent, with learning rate 0.5, stopping with threshhold 0.0005, gives (approximately) parameters for:

$$\hat{H} = \{(x_1, x_2) \in \mathbb{R}^2 \mid 2.01x_1 - 0.63x_2 - 0.26 = 0\}.$$



Logistic regression on the Iris data

Recall the Iris data set: 150 points, 50 from each of three species of Iris flower. Two of the species in the data set, Iris versicolor and Iris virginica, are not linearly separable.

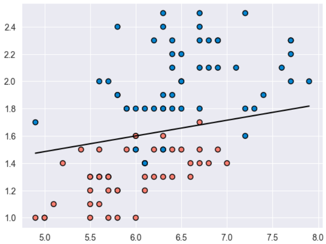
Logistic regression on the Iris data

Recall the Iris data set: 150 points, 50 from each of three species of Iris flower. Two of the species in the data set, Iris versicolor and Iris virginica, are not linearly separable.

We can use gradient descent on the logistic model to find a hyperplane that does *well* in classifying versicolor vs. virginica species.

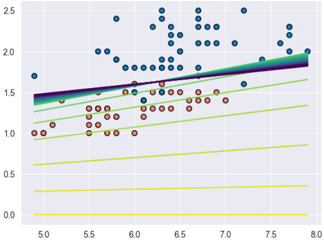
Logistic regression on the Iris data

Recall the Iris data set: 150 points, 50 from each of three species of Iris flower.
Two of the species in the data set, Iris versicolor and Iris virginica, are not linearly separable.
We can use gradient descent on the logistic model to find a hyperplane that does *well* in classifying versicolor vs. virginica species.
For visualization, I first show just the first and fourth coordinates, and the results for logistic model in 2D.



Logistic regression on the Iris data

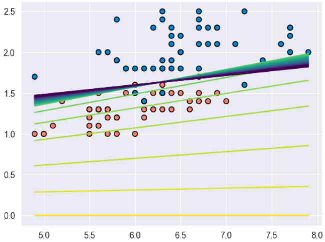
Pictured below are selected lines found during the batch gradient descent.



[3] Recall, the model labels the point with $+1$ when $f_{\omega}(\mathbf{x}) \geq 0.5$.

Logistic regression on the Iris data

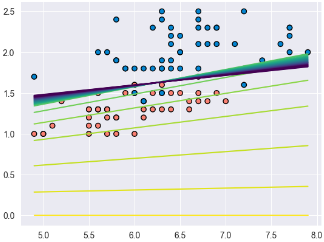
Pictured below are selected lines found during the batch gradient descent. As before, yellow-to-purple is progression through the procedure. Consecutive lines that are shown have 200 updates between them; ≈ 4000 updates in total.



[3]Recall, the model labels the point with $+1$ when $f_\omega(\mathbf{x}) \geq 0.5$.

Logistic regression on the Iris data

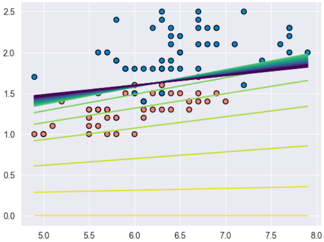
Pictured below are selected lines found during the batch gradient descent. As before, yellow-to-purple is progression through the procedure. Consecutive lines that are shown have 200 updates between them; ≈ 4000 updates in total. The accuracy in this 2D projection is 92% (the final hyperplane correctly labeled 92 out of 100).[3]



[3]Recall, the model labels the point with $+1$ when $f_\omega(\mathbf{x}) \geq 0.5$.

Logistic regression on the Iris data

Pictured below are selected lines found during the batch gradient descent.
As before, yellow-to-purple is progression through the procedure. Consecutive lines that are shown have 200 updates between them; ≈ 4000 updates in total.
The accuracy in this 2D projection is 92% (the final hyperplane correctly labeled 92 out of 100).[3]
The model on the points in $\mathbb{R}^4$ took less updates (just under 3000). It had 97% accuracy on the data.



[3]Recall, the model labels the point with $+1$ when $f_\omega(\mathbf{x}) \geq 0.5$.