Soft-margin SVM, SMO Algorithm, Decision Trees

Milan Straka

November 25, 2019
Kernel Linear Regression

When dimensionality of input is $D$, one step of SGD takes $O(D^3)$.

Surprisingly, we can do better under some circumstances. We start by noting that we can write the parameters $w$ as a linear combination of the input features $\varphi(x_i)$.

By induction, $w = 0 = \sum_i 0 \cdot \varphi(x_i)$, and assuming $w = \sum_i \beta_i \cdot \varphi(x_i)$, after a SGD update we get

$$w \leftarrow w + \alpha \sum_i (t_i - w^T \varphi(x_i)) \varphi(x_i)$$

$$= \sum_i \left( \beta_i + \alpha(t_i - w^T \varphi(x_i)) \right) \varphi(x_i).$$

A individual update is $\beta_i \leftarrow \beta_i + \alpha \left(t_i - w^T \varphi(x_i) \right)$, and substituting for $w$ we get

$$\beta_i \leftarrow \beta_i + \alpha \left(t_i - \sum_j \beta_j \varphi(x_j)^T \varphi(x_i) \right).$$
Kernel Linear Regression

We can formulate the alternative linear regression algorithm (it would be called a *dual formulation*):

**Input**: Dataset \( \mathbf{X} = \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \mathbb{R}^N \), learning rate \( \alpha \in \mathbb{R}^+ \).

- Set \( \beta_i \leftarrow 0 \)
- Compute all values \( K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \)
- Repeat
  - Update the coordinates, either according to a full gradient update:
    - \( \beta \leftarrow \beta + \alpha (\mathbf{t} - K \beta) \)
  - or alternatively use single-batch SGD, arriving at:
    - for \( i \) in random permutation of \( \{1, \ldots, N\} \):
      - \( \beta_i \leftarrow \beta_i + \alpha \left( t_i - \sum_j \beta_j K(\mathbf{x}_i, \mathbf{x}_j) \right) \)

In vector notation, we can write \( \beta \leftarrow \beta + \alpha (\mathbf{t} - K \beta) \).

The predictions are then performed by computing \( y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \sum_i \beta_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) \).
We define a *kernel* corresponding to a feature map $\varphi$ as a function

$$K(\mathbf{x}, \mathbf{z}) \overset{\text{def}}{=} \varphi(\mathbf{x})^t \varphi(\mathbf{z}).$$

There is quite a lot of theory behind kernel construction. The most often used kernels are:

- polynomial kernel or degree $d$

  $$K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z} + 1)^d,$$

  which corresponds to a feature map generating all combinations of up to $d$ input features;

- Gaussian (or RBF) kernel

  $$K(\mathbf{x}, \mathbf{z}) = e^{-\gamma ||\mathbf{x} - \mathbf{z}||^2},$$

  corresponding to a scalar product in an infinite-dimensional space (it is in a sense a combination of polynomial kernels of all degrees).
Assume we have a dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{t} \in \{-1, 1\}^N$, feature map $\varphi$ and model

$$y(x) \stackrel{\text{def}}{=} \varphi(x)^T \mathbf{w} + b.$$ 

We already know that the distance of a point $\mathbf{x}_i$ to the decision boundary is

$$\frac{|y(\mathbf{x}_i)|}{||\mathbf{w}||} = t_i y(\mathbf{x}_i).$$

We therefore want to maximize

$$\arg \max_{\mathbf{w}, \mathbf{b}} \frac{1}{||\mathbf{w}||} \min_i [t_i (\varphi(\mathbf{x})^T \mathbf{w} + \mathbf{b})].$$

However, this problem is difficult to optimize directly.
Support Vector Machines

Because the model is invariant to multiplying \( w \) and \( b \) by a constant, we can say that for the points closest to the decision boundary, it will hold that

\[
t_i y(x_i) = 1.
\]

Then for all the points we will have \( t_i y(x_i) \geq 1 \) and we can simplify

\[
\arg\max_{w,b} \frac{1}{||w||} \min_i \left[ t_i (\varphi(x)^T w + b) \right]
\]

to

\[
\arg\min_{w,b} \frac{1}{2} ||w||^2 \text{ given that } t_i y(x_i) \geq 1.
\]
Support Vector Machines

In order to solve the constrained problem of

$$\arg\min_{w, b} \frac{1}{2}||w||^2 \text{ given that } t_i y(x_i) \geq 1,$$

we write the Lagrangian with multipliers $a = (a_1, \ldots, a_N)$ as

$$L = \frac{1}{2}||w||^2 - \sum_i a_i [t_i y(x_i) - 1].$$

Setting the derivatives with respect to $w$ and $b$ to zero, we get

$$w = \sum_i a_i t_i \varphi(x_i)$$

$$0 = \sum_i a_i t_i$$
Support Vector Machines

Substituting these to the Lagrangian, we get

\[ L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(x_i, x_j) \]

with respect to the constraints \( \forall i : a_i \geq 0, \sum_i a_i t_i = 0 \) and kernel \( K(x, z) = \varphi(x)^T \varphi(z) \).

The solution of this Lagrangian will fulfil the KKT conditions, meaning that

\[
\begin{align*}
    a_i &\geq 0 \\
t_i y(x_i) - 1 &\geq 0 \\
    a_i (t_i y(x_i) - 1) &= 0.
\end{align*}
\]

Therefore, either a point is on a boundary, or \( a_i = 0 \). Given that the predictions for point \( x \) are given by \( y(x) = \sum a_i t_i K(x, x_i) + b \), we need to keep only the points on the boundary, the so-called support vectors.
Support Vector Machines

The dual formulation allows us to use non-linear kernels.

**Figure 7.2** Example of synthetic data from two classes in two dimensions showing contours of constant $y(x)$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

*Figure 7.2 of Pattern Recognition and Machine Learning.*
Until now, we assumed the data to be linearly separable – the hard-margin SVM variant. We now relax this condition to arrive at soft-margin SVM. The idea is to allow points to be in the margin or even on the wrong side of the decision boundary. We introduce slack variables $\xi_i \geq 0$, one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\mathbf{x}_i) \geq 1, \\ |t_i - y(\mathbf{x}_i)| & \text{otherwise.} \end{cases}$$

Therefore, $\xi_i = 0$ signifies a point outside of margin, $0 < \xi_i < 1$ denotes a point inside the margin, $\xi_i = 1$ is a point on the decision boundary and $\xi_i > 1$ indicates the point is on the opposite side of the separating hyperplane.

Therefore, we want to optimize

$$\arg \min_{w,b} C \sum_i \xi_i + \frac{1}{2} ||w||^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$
We again create a Lagrangian, this time with multipliers $a = (a_1, \ldots, a_N)$ and also $\mu = (\mu_1, \ldots, \mu_N)$:

$$L = \frac{1}{2}||w||^2 + C \sum_i \xi_i - \sum_i a_i [t_i y(x_i) - 1 + \xi_i] - \sum_i \mu_i \xi_i.$$ 

Solving for the critical points and substituting for $w$, $b$ and $\xi$ (obtaining an additional constraint $\mu_i = C - a_i$ compared to the previous case), we obtain the Lagrangian in the form

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(x_i, x_j),$$

which is identical to the previous case, but the constraints are a bit different:

$$\forall i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$
Using KKT conditions, we can see that the support vectors (examples with $a_i > 0$) are the ones with $t_i y(x_i) = 1 - \xi_i$. 

*Figure 7.4 of Pattern Recognition and Machine Learning.*
Note that the slack variables can be written as

$$\xi_i = \max \left(0, 1 - t_i y(x_i)\right),$$

so we can reformulate the soft-margin SVM objective using **hinge loss**

$$L_{\text{hinge}}(t, y) \overset{\text{def}}{=} \max(0, 1 - ty)$$

to

$$\arg \min_{w, b} C \sum_i L_{\text{hinge}}(t_i, y(x_i)) + \frac{1}{2} \|w\|^2.$$ 

Such formulation is analogous to a regularized loss, where $C$ is an **inverse** regularization strength, so $C = \infty$ implies no regularization and $C = 0$ ignores the data entirely.
Comparison of Linear and Logistic Regression and SVM

For \( f(\mathbf{x}; \mathbf{w}, b) \stackrel{\text{def}}{=} \varphi(\mathbf{x})^T \mathbf{w} + b \), we have seen the following losses:

<table>
<thead>
<tr>
<th>Model</th>
<th>Objective</th>
<th>Per-Instance Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>( \arg \min_{\mathbf{w}, b} \sum_i L_{\text{MSE}} \left( t_i, f(\mathbf{x}_i) \right) + \frac{1}{2} \lambda | \mathbf{w} |^2 )</td>
<td>( L_{\text{MSE}}(t, y) = \frac{1}{2} (t - y)^2 )</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>( \arg \min_{\mathbf{w}, b} \sum_i L_{\alpha-\text{NLL}} \left( t_i, f(\mathbf{x}_i) \right) + \frac{1}{2} \lambda | \mathbf{w} |^2 )</td>
<td>( L_{\alpha-\text{NLL}}(t, y) = - \log \left( \frac{\sigma(y)^t}{(1 - \sigma(y))^{1-t}} \right) )</td>
</tr>
<tr>
<td>Softmax regression</td>
<td>( \arg \min_{\mathbf{w}, b} \sum_i L_{s-\text{NLL}} \left( t_i, f(\mathbf{x}_i) \right) + \frac{1}{2} \lambda | \mathbf{w} |^2 )</td>
<td>( L_{s-\text{NLL}}(t, \mathbf{y}) = - \log \text{softmax}(\mathbf{y})_t )</td>
</tr>
<tr>
<td>SVM</td>
<td>( \arg \min_{\mathbf{w}, b} C \sum_i L_{\text{hinge}} \left( t_i, f(\mathbf{x}_i) \right) + \frac{1}{2} | \mathbf{w} |^2 )</td>
<td>( L_{\text{hinge}}(t, y) = \max(0, 1 - ty) )</td>
</tr>
</tbody>
</table>

Note that \( L_{\text{MSE}}(t, y) \propto - \log \left( \mathcal{N}(t; \mu = y, \sigma^2 = 1) \right) \) and that \( L_{\alpha-\text{NLL}}(t, y) = L_{s-\text{NLL}}(t, [y, 0]) \).
To compare various functions for binary classification, we need to formulate them all in the same settings, with $t \in \{-1, 1\}$.

- **MSE**: $(ty - 1)^2$, because it is $(y - 1)^2$ for $t = 1$ and $(-y - t)^2$ for $t = -1$
- **LR**: $\sigma(ty)$, because it is $\sigma(y)$ for $t = 1$ and $1 - \sigma(y) = \sigma(-y)$ for $t = -1$
- **SVM**: $\max(0, 1 - ty)$

![Comparison of binary losses with {-1,1} targets](image)
Sequential Minimal Optimization Algorithm

To solve the dual formulation of a SVM, usually Sequential Minimal Optimization (SMO; John Platt, 1998) algorithm is used.

Before we introduce it, we start by introducing coordinate descent optimization algorithm.

Consider solving unconstrained optimization problem

$$\arg \min_w L(w_1, w_2, \ldots, w_D).$$

Instead of the usual SGD approach, we could optimize the weights one by one, using the following algorithm

- loop until convergence
  - for $i$ in $\{1, 2, \ldots, D\}$:
    - $w_i \leftarrow \arg \min_{w_i} L(w_1, w_2, \ldots, w_D)$
Sequential Minimal Optimization Algorithm

- loop until convergence
- for $i$ in $\{1, 2, \ldots, D\}$:
  - $w_i \leftarrow \text{arg min}_{w_i} L(w_1, w_2, \ldots, w_D)$

If the inner $\text{arg min}$ can be performed efficiently, the coordinate descent can be fairly efficient.

Note that we might want to choose $w_i$ in different order, for example by trying to choose $w_i$ providing the largest decrease of $L$. 

NPFL129, Lecture 6  Refresh  Soft-margin SVN  SMO  Primal vs Dual  Decision Tree

In soft-margin SVM, we try to minimize

\[ L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(x_i, x_j), \]

such that

\[ \forall i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0. \]

The KKT conditions for the solution are

- \( a_i = 0 \Rightarrow t_i y(x_i) \geq 1, \) because in this case \( \mu_i = C \) and therefore \( \xi_i = 0 \)
- \( a_i = C \Rightarrow t_i y(x_i) \leq 1, \) because in this case \( t_i y(x_i) - 1 + \xi_i = 0 \) for \( \xi_i \geq 0 \)
- \( 0 < a_i < C \Rightarrow t_i y(x_i) = 1, \) because in this case both above reasonings apply
Sequential Minimal Optimization Algorithm

At its core, the SMO algorithm is just a coordinate descent. It tries to find such $\alpha_i$ fulfilling the KKT conditions – for soft-margin SVM, KKT conditions are sufficient conditions for optimality (the loss is convex and inequality constraints affine).

However, note that because of the $\sum a_i t_i = 0$ constraint we cannot optimize just one $a_i$, because a single $a_i$ is determined from the others.

Therefore, in each step we pick two $a_i, a_j$ coefficients and try to minimize the loss while fulfilling the constraints.

- loop until convergence (until $\forall i : a_i < C \Rightarrow t_i y(x_i) \geq 1$ and $a_i > 0 \Rightarrow t_i y(x_i) \leq 1$)
  - for $i$ in $\{1, 2, \ldots, D\}$, for $j \neq i$ in $\{1, 2, \ldots, D\}$:
    - $a_i, a_j \leftarrow \arg\min_{a_i, a_j} L(a_1, a_2, \ldots, a_D)$ such that $C \geq a_i \geq 0$, $\sum_i a_i t_i = 0$
The SMO is an efficient algorithm, because we can compute the update to $a_i, a_j$ efficiently, because there exists an closed form solution.

Assume that we are updating $a_i$ and $a_j$. Then from the $\sum_k a_k t_k = 0$ condition we can write $a_i t_i = -\sum_{k \neq i} a_k t_k$. Given that $t_i^2 = 1$ and denoting $\zeta = -\sum_{k \neq i, k \neq j} a_k t_k$, we get

$$a_i = t_i (\zeta - a_j t_j).$$

Minimizing $L(a)$ with respect to $a_i$ and $a_j$ then amounts to minimizing a quadratic function of $a_j$, which has an analytical solution.

Note that the real SMO algorithm has several heuristics for choosing $a_i, a_j$ such that the $L$ can be minimized the most.
Sequential Minimal Optimization Algorithm Sketch

**Input**: Dataset \((\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \{-1, 1\}^N)\), kernel \(\mathbf{K}\), regularization parameter \(C\), tolerance \(tol\), \(max\_passes\_without\_a\_changing\) value

- Initialize \(a_i \leftarrow 0\), \(b \leftarrow 0\), \(passes \leftarrow 0\)

- **while** \(passes < max\_passes\_without\_a\_changing\):
  - \(changed\_as \leftarrow 0\)
  - **for** \(i\) in \(1, 2, \ldots, N\):
    - \(E_i \leftarrow y(\mathbf{x}_i) - t_i\)
    - **if** \((a_i < C \text{ and } t_i E_i < -tol) \text{ or } (a_i > 0 \text{ and } t_i E_i > tol)\):
      - Choose \(j \neq i\) randomly
      - Update \(a_i\), \(a_j\) and \(b\)
      - \(changed\_as \leftarrow changed\_as + 1\)
  - **if** \(changed\_as = 0\): \(passes \leftarrow passes + 1\)
  - **else**: \(passes \leftarrow 0\)
Sequential Minimal Optimization Algorithm Sketch

**Input:** Dataset $(X \in \mathbb{R}^{N \times D}, t \in \{-1, 1\}^N)$, kernel $K$, regularization parameter $C$, tolerance $tol$, $max\_passes\_without\_a\_changing$ value

- Update $a_i, a_j, b$:
  - Express $a_i$ using $a_j$
  - Find $a_j$ optimizing the loss $L$ quadratic with respect to $a_j$
  - Clip $a_j$ so that $0 \leq a_i, a_j \leq C$
  - Compute corresponding $a_i$
  - Compute $b$ matching to updated $a_i, a_j$
Assume we have a dataset with $N$ training examples, each with $D$ features. Also assume the used feature map $\varphi$ generates $F$ features.

<table>
<thead>
<tr>
<th>Property</th>
<th>Primal Formulation</th>
<th>Dual Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>$F$</td>
<td>$N$</td>
</tr>
<tr>
<td>Model size</td>
<td>$F$</td>
<td>$s \cdot D$ for $s$ support vectors</td>
</tr>
<tr>
<td>Usual training time</td>
<td>$c \cdot N \cdot F$ for $c$ iterations</td>
<td>between $\Omega(ND)$ and $\mathcal{O}(N^2D)$</td>
</tr>
<tr>
<td>Inference time</td>
<td>$\Theta(F)$</td>
<td>$\Theta(s \cdot D)$ for $s$ support vectors</td>
</tr>
</tbody>
</table>
The idea of decision trees is to partition the input space into usually cuboid regions and solving each region with a simpler model.

We focus on Classification and Regression Trees (CART; Breiman et al., 1984), but there are additional variants like ID3, C4.5, ...

![Decision Tree Diagram](Figure 14.5 of Pattern Recognition and Machine Learning.)

![Decision Tree Diagram](Figure 14.6 of Pattern Recognition and Machine Learning.)
Regression Decision Trees

Assume we have an input dataset \( \mathbf{X} \in \mathbb{R}^{N \times D} \), \( t \in \mathbb{R}^{N} \). At the beginning, the decision tree is just a single node and all input examples belong to this node. We denote \( I_T \) the set of training example indices belonging to a leaf node \( T \).

For each leaf, our model will predict the average of the training examples belonging to that leaf, 
\[
\hat{t}_T = \frac{1}{|I_T|} \sum_{i \in I_T} t_i.
\]

We will use a criterion \( c_T \) telling us how uniform or homogeneous are the training examples belonging to a leaf node \( T \) – for regression, we will employ the MSE between the examples belonging to the node and the predicted value in that node; this is proportional to variance of the training examples belonging to the leaf node \( T \), multiplied by the number of the examples.

\[
c_{\text{MSE}}(T) \overset{\text{def}}{=} \sum_{i \in I_T} (t_i - \hat{t}_T)^2, \quad \text{where} \quad \hat{t}_T = \frac{1}{|I_T|} \sum_{i \in I_T} t_i.
\]
Tree Construction

To split a node, the goal is to find a feature and its value such that when splitting a node $T$ into $T_L$ and $T_R$, the resulting regions decrease the overall criterion value the most, i.e., the difference $c_{T_L} + c_{T_R} - c_T$ is the lowest.

Usually we have several constraints, we mention on the most common ones:

- **maximum tree depth**: we do not split nodes with this depth;
- **minimum examples to split**: we only split nodes with this many training examples;
- **maximum number of leaf nodes**

The tree is usually built in one of two ways:

- if the number of leaf nodes is unlimited, we usually build the tree in a depth-first manner, recursively splitting every leaf until some above constraint is invalidated;
- if the maximum number of leaf nodes is give, we usually split such leaf $T$ where the criterion difference $c_{T_L} + c_{T_R} - c_T$ is the lowest.
Classification Decision Trees

For multi-class classification, we predict such class most frequent in the training examples belonging to a leaf $\mathcal{T}$.

To define the criterions, let us denote the average probability for class $k$ in a region $\mathcal{T}$ at $p_{\mathcal{T}}(k)$.

For classification trees, one of the following two criterions is usually used:

- **Gini index**:

  \[
  c_{\text{Gini}}(\mathcal{T}) \overset{\text{def}}{=} |I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k) (1 - p_{\mathcal{T}}(k))
  \]

- **Entropy Criterion**

  \[
  c_{\text{entropy}}(\mathcal{T}) \overset{\text{def}}{=} |I_{\mathcal{T}}| H(p_{\mathcal{T}}) = -|I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k)
  \]