SVM optimization and Kernel methods

Announcements

• Hw 4 is up. Due in a week.
• Kaggle is up

Outline

• Review SVM optimization
• Non-linear transformations in SVM
• Soft-margin SVM

Goal:

• Find margins that maximize separability of classes
• Margins are defined by data points closest to the hyperplane.
Optimizing

• Maximize geometric (e.g. normalized) margins $\gamma$.
• Without misclassifying a point $y_i(w^T x_i + b) \geq \gamma$ for all $i$.
• And our normalizing constraint $||w|| = 1$.

• This was horribly non-convex so we tried something else instead.
  • Substitute in the definition of functional margins $\gamma = \frac{y}{||w||}$.
  • Gets rid of the normalizing constraint.

Optimizing cont.

• Now because the functional margin can arbitrarily be scaled, we assign it to a value 1.

• Our maximization problem is now
  $$\max_{w,b} \frac{1}{||w||} \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N$$

• Which is equivalent to minimizing
  $$\min_{w,b} w^T w \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N$$

• Note that $\gamma$ is gone from this equation.

Intuition that the problem is convex

• $\min_{w,b} \frac{1}{2} ||w||^2$ s.t. $y_i(w^T x_i + b) \geq 1, \ i = 1, \ldots, N$
Lagrange formulation

- Minimize
  \[ L(w, b, \alpha) = \frac{1}{2} ||w||^2 - \sum_{i=1}^{N} \alpha_i (y_i (w^T x_i + b) - 1) \]
  with respect to \( w \) and \( b \)

- And maximize with respect to each \( \alpha_i \geq 0 \)
  \[ w^T L = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \]
  \[ \frac{\partial L}{\partial \alpha} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \]

Substituting

- Let \( w = \sum_{i=1}^{N} \alpha_i y_i x_i \) and \( \sum_{i=1}^{N} \alpha_i y_i = 0 \)

into the Lagrangian

\[ L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i (y_i (w^T x_i + b) - 1) \]

we get

\[ L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i, x_j) \]

Where \( \alpha_i \geq 0, i = 1, ..., N \) and \( \sum_{i=1}^{N} \alpha_i y_i = 0 \)

New optimization requirements

\[ L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i, x_j) \]

which we maximize with respect to \( \alpha \)

subject to \( \alpha_i \geq 0 \) for \( i = 1, ..., N \) and \( \sum_{i=1}^{N} \alpha_i y_i = 0 \)

- This is something we don’t know how to optimize using our statistical/mathematical knowledge for this class...
- But this is a standard form with a known solution

Extending this idea to SVMs

\[ L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i, x_j) \]

- Let's replace \( (x_i, x_j) \) with \( (z_i, z_j) \) where \( z \) is higher dimension
Extending this idea to SVMs

\[ L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \]

• How much more expensive is this?

Increasing hypothesis complexity

• \( L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \)

• vs \( L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle z_i, z_j \rangle \)

• The number of parameters has not changed.
  • \( \alpha_i \) is based on the number data points
  • \( w = \sum_{i=1}^{N} \alpha_i y_i x_i \) and \( \alpha_i (y_i (w^T x_i + b) - 1) = 0 \)
  • If \( \{z_i, z_j\} \) is easy to calculate, this problem isn’t any harder.
    • Inner products are cheap. So we can go to higher dimensions easily

Higher dimensions: Non-linear hyperplane

• Here our support vectors live in \( Z \) space.
• In \( X \) space we have "pre-images" of our support vectors.
• Margins are maintained in \( Z \) space but to compute the margin we still only need dot products.

Extending this idea to SVMs

• Our algorithm can be written in terms of inner products with the input features
  • E.g. we don’t need the input data (or its transformation) except for the inner product
    \[ L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \]
  • Let’s replace \( \langle x_i, x_j \rangle \) with \( \langle \phi(x_i), \phi(x_j) \rangle \)
    • Computing the inner product is a fairly cheap computation.
    • This allows us to make much more complex models
Kernel trick

- We can easily compute $K(x, z)$ by finding $\phi(x)$ and $\phi(z)$ and taking the inner product.
- But it's easy to imagine how this can get computationally expensive.
- There are some $K(x, z)$ though
  - where we don't actually need to compute the individual $\phi(x)$ and $\phi(z)$'s
  - instead we get the inner product directly.
- This allows us to get an SVM result in high dimensional feature space given by $\phi$ but without explicitly finding or representing $\phi(x)$.
- Effectively free higher dimensional hypotheses.

Generalizing the Kernel $(x^T z + c)^d$

- $\phi$ gives us some intuitions to how Kernels work.
- $c$ controls the relative weighting between $x_i$ (first order) terms and the $x_i x_j$ (second order) terms.
- More broadly the kernel $K(x, z) = (x^T z + c)^d$ corresponds to a feature mapping to a $\binom{n+d}{d}$ feature space, which includes all terms $x, x^*, ..., x^d$.
- Whereas computing $\phi(x)$ takes $O(n^d)$ time, the kernel still takes only $O(n)$ time.
- Don’t explicitly representing feature vectors in our high d space.

Kernels as a similarity measure

- If $\phi(x)$ and $\phi(z)$ are close together, what should $K(x, z)$ be?
- Remember the Kernel is replacing our dot product (which measures the angle between two vectors).
- $K(x, z)$ is some sort of (approximate) distance between $x$ and $z$.
  - If $\phi(x)$ and $\phi(z)$ are close, $K(x, z)$ is larger.
  - If $\phi(x)$ and $\phi(z)$ are far, $K(x, z)$ is small.

Gaussian kernel

$$K(x, z) = \exp \left( -\frac{|x - z|^2}{2\sigma^2} \right)$$

- Captures similarity.
  - It's close to 1 when $x$ and $z$ are close.
  - And close to 0 when $x$ and $z$ are far apart.
By placing the input features in high dimensional space, we’re increasing the likelihood that the data is actually separable. Either because the data is already linearly separable or can be made so by a (potentially non-linear) transformation.

Some cases, we don’t want to find a separating hyperplane. It can be susceptible to outliers. Sometimes we want a robust rather than accurate hypothesis.

Extend optimal margin (linear SVMs) to non-linearly separable data. Makes the margin less sensitive to outliers. Introduce $\ell_1$ regularization. Like ridge regression, penalize the values for being in the margin. Or penalize for being incorrectly classified.

$$\min_{\|w\|^2} \frac{1}{2} \|w\|^2 + C \sum \xi_i$$

s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i$, $i = 1, ..., n$ and $\xi_i \geq 0$
Regularization and SVMs

\[
\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \\
\text{s.t. } y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N \quad \text{and} \quad \xi_i \geq 0
\]

- Training data points can now have a (functional) margin less than 1
- If a data point has a functional margin \(1 - \xi_i\) (with \(\xi_i > 0\))
- There's a cost in or objective fn of \(C\xi_i\)
- \(C\) controls the relative weighting between minimizing \(\|\mathbf{w}\|^2\) and having functional margins at least 1.

Regularization

- As in ridge regression, we can control the penalty \(C\)
- In ridge we called it \(\lambda\)
- And it controlled the size of the \(\beta\)s compared to the ability to fit the training data
- Now we’re trading off between
- Large margins (making \(\|\mathbf{w}\|^2\) small)
- And functional margins of at least 1 (\(\xi_i = 0\))
- As \(C\) goes to 0, we get the optimal margin classifier

Solving for \(\mathbf{w}\) and \(b\)

- Need to construct the Lagrangian again, this time with regularization.
- We have two Lagrange multipliers
  - One for the constraint \(y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i\)
  - And another for \(\xi_i \geq 0\)
- \(\mathcal{L}(\mathbf{w}, b, \xi, r, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{N} \xi_i = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i - \sum_{i=1}^{N} r_i \xi_i\)
- \(\alpha_i\) and \(r_i\) are Lagrange multipliers so \(\alpha_i, r_i \geq 0\)
- We solve for the derivative w.r.t. \(\mathbf{w}\) and \(b\), set it to 0
- Simplifying, we obtain

\[
\max_{\alpha} W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i y_i \alpha_j [\mathbf{x}_i \cdot \mathbf{x}_j] \\
\text{s.t. } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, N \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i y_i = 0
\]
Solving for $\mathbf{w}$ and $b$

\[
\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle
\]

s.t. $0 \leq \alpha_i \leq C, \quad i = 1, \ldots, N \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$

- Note that we now have an upper bound $C$ for $\alpha_i$
- $\mathbf{w}$ can be expressed in terms of $\alpha_i$'s
- The only change from regularization is that we put an upper bound on $\alpha_i$

Support vectors

- Considering the values $\alpha_i$ we could tell if something was a support vector
  - If it was greater than 0, it was a support vector
  - Otherwise, we knew the data point was not near the decision boundary
- Now though, we can have points in the margins.... How do we determine if something is a support vector?

Support vectors

- Considering the values $\alpha_i$ we could tell if something was a support vector
  - $\alpha_i = 0 \Rightarrow y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ “far” from boundary
  - $\alpha_i = C \Rightarrow y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ in the margin/misclassified
  - $0 < \alpha_i < C \Rightarrow y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$ support vector
Algorithms for solving SVMs

- We saw that we needed quadratic programming
- And Lagrange multipliers for standard SVMs
- How about an algorithm for the soft-margin (e.g. non separable) SVMs?

Aside: Coordinate ascent

- Let’s say we have some function \( \max_a W(\alpha_1, \alpha_2, ..., \alpha_N) \) that we’re trying to optimize
  - What methods have we seen to optimize such a problem?
- A new algorithm:
  Loop until convergence:
  
  \[
  \text{For } i = 1, ..., n \{ \\
  a_i := \arg \max_{a_i} W(\alpha_1, ..., \alpha_{i-1}, a_i, \alpha_{i+1}, ..., \alpha_N) \\
  \}
  \]

  - Hold all variables except for \( a_i \) fixed
  - Reoptimize \( W(\alpha) \) with respect to just parameter \( a_i \)
  - Here we’re just visiting \( \alpha \)'s in order
  - But you could imagine other selection algorithms (e.g. one that maximizes the increase for \( W(\alpha) \))

Intermediate step: Coordinate ascent

Loop until convergence:

\[
\text{For } i = 1, ..., n \{ \\
  a_i := \arg \max_{a_i} W(\alpha_1, ..., \alpha_{i-1}, a_i, \alpha_{i+1}, ..., \alpha_N) \\
  \}
\]

Sequential minimal optimization

- SMO algorithm
  - We want to solve
    \[
    \max_a W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i, x_j)
    \]
    s.t. \( 0 \leq \alpha_i \leq C, \quad i = 1, ..., N \) and \( \sum_{i=1}^{N} \alpha_i y_i = 0 \)
  - Let’s say we have a set of \( \alpha_i \)'s that satisfy our constraints
  - What happens if we try coordinate ascent?
We’re stuck

- Because we know that $\sum_{i=1}^{N} \alpha_i y_i = 0$
- We can see that $\alpha_1 y_1 = -\sum_{i=2}^{N} \alpha_i y_i$
- Equivalently, $\alpha_1 = -y_1 \sum_{i=2}^{N} \alpha_i y_i$

- So we can’t get a better estimate for $\alpha_1$ here...

- Instead we’ll consider two $\alpha_i$’s at a time

SMO algorithm

Loop until convergence {
    1. Select some pair $\alpha_i$ and $\alpha_j$ to update next
       (usually we use a heuristic that picks two that will allow the
       largest progress toward the global minimum)
    2. Reoptimize our estimate of $w (W(\alpha))$ with respect to $\alpha_i$
       and $\alpha_j$, while holding all $\alpha_k$ ($k \neq i, j$) fixed.
}

10