Linear Models and Logistic Regression

CIML Chapters 8 and 9
Daniel Lowd
CIS 472/572


Optimization Framework

- **Example:** Minimize training set errors of a linear model

  \[
  \min_{w, b} \sum_n \left[ 1[y_n (w \cdot x_n + b) > 0] \right]
  \]

- **Problem 1:** We want to minimize test error — minimizing training error may lead to overfitting!
- **Problem 2:** This is NP-hard to optimize, even approximately!

Convex Surrogate Loss Functions

- 0-1 loss is non-smooth — a small change in a parameter could lead to a BIG change in accuracy! (How?)

Examples of Loss Functions

| Zero/one: | \(l^{(0)}(y, \hat{y}) = 1[y \leq 0] \) | (6.3) |
| Hinge: | \(l^{(h)}(y, \hat{y}) = \max(0, 1 - y\hat{y}) \) | (6.4) |
| Logistic: | \(l^{(l)}(y, \hat{y}) = \frac{1}{\log 2} \log(1 + \exp(-y\hat{y})) \) | (6.5) |
| Exponential: | \(l^{(e)}(y, \hat{y}) = \exp(-y\hat{y}) \) | (6.6) |
| Squared: | \(l^{(s)}(y, \hat{y}) = (y - \hat{y})^2 \) | (6.7) |

Previous: Perceptron

- Model: hyperplane (linear model)
- Algorithm: mistake-based adjustment
  \(\rightarrow\) Finds a separating hyperplane if one exists!
- Are some separating hyperplanes better than others?
- What if the data isn’t separable?

Figure credit: Krios on Wikipedia (Loss_function_surrogates.png)
Weight Regularization

Q: How does a linear model overfit?
A: Large weights based on little evidence (why?)
Q: How do we prevent overfitting?
A: Adjust our inductive bias with a regularization function which penalizes large weights (or anything else we wish to avoid).

Regularized loss function:
\[
\min_{w,b} \sum_n f(y_n, w \cdot x_n + b) + \lambda R(w, b)
\]

P-Norms

- Common loss functions:
  - L_1 norm of weight vector (square root of sum of squared weights)
  - L_2 norm of weight vector (sum of absolute weights)
- Generalizing these:
\[
||w||_p = \left( \sum |w|^p \right)^{\frac{1}{p}}
\]

Gradient Descent

Q: Suppose you want to avoid a flood. How do you find a high place?
A: Walk uphill!

Q: And if you want to find a low place?
A: Walk downhill!

Q: And if you want to minimize a function?
A: Gradient descent!

[Algorithm for Gradient Descent]

```
Algorithm 24 GRADIENTDESCENT(\epsilon, \eta_1, \eta_2, \ldots)
1. \textbf{initialize variable we are optimizing} \quad g^{(0)} \leftarrow \left[ y(x_1, \ldots, x_K) \right]
2. \textbf{for} i \leftarrow 1, \ldots, N \textbf{do}
3. \quad \textbf{compute gradient at current location} g^{(i)} \leftarrow \nabla F(x_i)
4. \quad \textbf{take a step down the gradient} x^{(i)} \leftarrow x^{(i-1)} - \eta g^{(i)}
5. \textbf{end for}
6. \textbf{return } x^{(N)}
```

L(w, b) = \sum_n \exp \left[ -y_n(w \cdot x_n + b) \right] + \frac{\lambda}{2} ||w||^2
Disadvantages?

• Negative bias terms are not updated.

Therefore, if the current prediction is large, then the exponential term is as large as possible. As this value tends toward negative infinity, the objective value will be as bad as possible. This is true for all our problems, provided the regularized objective function is differentiable everywhere. In particular, the function to be optimized needs to be convex.

Before proceeding, it is worth thinking about what this says. From a practical perspective, the optimization will operate by updating parameters so that the objective value is as small as possible. This is typically done by using gradient descent.

Now that we have done the easy case, let's do the gradient with strongly convex functions.

**Theorem 7 (Gradient Descent Convergence).** Under suitable conditions, for an appropriately chosen constant step size (i.e., \( \eta_1 = \eta_2 = \cdots = \eta \)), the convergence rate of gradient descent is \( O(1/k) \). More specifically, letting \( z^* \) be the global minimum of \( F \), we have:

\[
F(z^*) - F(z) \leq \frac{|z - z^{*}|^2}{2h}
\]

Q: What's an "appropriately chosen constant step size"?

A: \( 1/L \) where \( L \) is the curvature of the function.

Many variants – adaptive learning rates, line search, or momentum terms – speed up convergence.

**Computing Gradients**

\[
\frac{\partial L}{\partial b} = \sum x_i \exp \left[-y_i (w \cdot x_i + b)\right] + \frac{\lambda}{2} \|w\|^2
\]

\[
\frac{\partial L}{\partial w} = \sum x_i y_i \exp \left[-y_i (w \cdot x_i + b)\right] x_i
\]

\[
\frac{\partial L}{\partial x_i} = \exp \left[-y_i (w \cdot x_i + b)\right] y_i x_i - \sum_{j \neq i} \exp \left[-y_j (w \cdot x_j + b)\right] y_j x_j
\]

**Stochastic Gradient**

• **Perceptron algorithm**: Update model immediately after each misprediction.

• **Gradient descent**: Compute gradient over all examples before updating.

What if we instead compute the gradient on a randomly chosen subset of the data?

Advantages?

Disadvantages?

**Predicting Probabilities**

• **0/1 loss**: Minimize mistakes

• What if an email has a 60% chance of being spam?

• What if there is a 30% chance of spam?

• What if a mushroom has a 10% probability of being poisonous?

• Probability can save your life!

**Logistic Regression**

• Predict class probabilities \( P(Y|X) \) with a linear model

• How do we turn \( (w \cdot x + b) \) into a probability?

**The Logistic Function**

The logistic function squashes a real number in the range \((-\infty, +\infty)\) into the range \((0, 1)\).

\[
\sigma(z) = \frac{1}{1 + \exp(-z)}
\]
Understanding Sigmoid

\[ \sigma(w_1 x_1 + b) = \frac{1}{1 + \exp(-(w_1 x_1 + b))} \]

Learning Logistic Regression

Key idea: Choose a model where reality is probable.

Probability of the true labels given the data:

\[ \prod_{(x,y) \in D} \hat{P}(y|x) = \prod_{(x,y) \in D} \frac{1}{1 + \exp(-y(w^T x + b))} \]

\[ \log \prod_{(x,y) \in D} \hat{P}(y|x) = \sum_{(x,y) \in D} - \log(1 + \exp(-y(w^T x + b))) \]

Conditional log-likelihood

(Conditional Log-Likelihood)

Logistic Regression

\[ P(y|x) = \sigma(w^T x + b) = \frac{1}{1 + \exp(-(w^T x + b))} \]

Example: Logistic function applied to linear function of \( x_1 \) and \( x_2 \):

Gradient of Logistic Loss

\[ \frac{\partial L}{\partial w_i} = \sum_{(x,y) \in D} \frac{1}{1 + \exp(y(w^T x + b))} y_i x_i \]

\[ \frac{\partial L}{\partial b} = \sum_{(x,y) \in D} \frac{1}{1 + \exp(y(w^T x + b))} y \]

Large parameters...

- Maximum likelihood solution: prefers higher weights
  - higher likelihood of (properly classified) examples close to decision boundary
  - larger influence of corresponding features on decision
  - can cause overfitting!!!

- Regularization: penalize high weights

Maximum Conditional A Posteriori

\[ p(w \mid Y, X) \propto P(Y \mid X, w) p(w) \]

- One common approach is to define priors on \( w \)
  - Normal distribution, zero mean, diagonal covariance
    \[ p(w) = \prod_{i} \frac{1}{\sqrt{2\pi} \sigma_i^2} e^{-\frac{w_i^2}{2\sigma_i^2}} \]
  - Log p(w) = -w^2/(2 \sigma_i^2) + constants = L2 regularizer
  - Laplace distribution = L1 regularizer.

Summary: To predict probabilities, minimize logistic loss with a regularizer (L1 and/or L2).
Support Vector Machines
Q: Which of these linear separators is optimal?  
A: The one that’s farthest from the other points? (Why?)
   • Separator only depends on nearest points (support vectors).
   • Number of support vectors is an upper bound on LOOCV error.

How do we maximize the margin?
   • Idea 1: Maximize \( y(wx+b) \) (for closest points)
   • Idea 2: Maximize \( y(wx+b)/||w|| \)
   • Idea 3: Maximize \( y(wx+b) \) for \( ||w|| <= 1 \).
   • Idea 4: Maximize \( 1/||w|| \) for \( y(wx+b) >= 1 \)
   • Idea 5: Maximize \( ||w||^2 \) for \( y(wx+b) >= 1 \)

Support Vector Machines
\[
\text{minimize} \quad \frac{1}{2}||w||^2 \\
\text{such that} \quad y_i(w^T x_i + b) \geq 1
\]
(Using subscripts to denote different training examples here.)

Soft-Margin SVM
• What if data is not linear separable? (noisy data, outliers, etc.)
• Slack variables \( \xi \) can be added to allow misclassification of difficult or noisy data points

Soft-Margin SVM
Formulation:
\[
\text{minimize} \quad \frac{1}{2}||w||^2 + C \sum \xi_i \\
\text{such that} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \\
\xi_i \geq 0
\]
Parameter \( C \) can be viewed as a way to control over-fitting.

Solving the Soft-Margin SVM
\[
y_j(w^T x_j + b) \geq 1 - \xi_j
\]
What’s the optimal \( \xi_j \)?
\[
\xi_j \geq y_j(w^T x_j + b) \\
\xi_j \geq 0 \\
\xi_j = \max(0, 1 - y_j(w^T x_j + b))
\]
\text{Hinge loss!}
Soft-Margin SVM

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2}\|w\|^2 + C \sum_{(x,y) \in D} \max(0, 1 - y(w^T x + b)) \\
\text{minimize} & \quad \sum_{(x,y) \in D} \max(0, 1 - y(w^T x + b)) + \frac{\lambda}{2}\|w\|^2
\end{align*}
\]

Maximize (soft) margin = Minimize hinge loss with L2 regularization

Subgradients

- What if the gradient is undefined?
- Example: \(d/dw |w|\)

All of these are valid subgradients at \(w=0\)!

Algorithm 23: HingeRegularizedGD(D, \_lambda, MaxIter)

1. \(w \leftarrow (0, \ldots, 0), \quad b \leftarrow 0\) \hspace{1cm} \(// \text{initialize weights and bias}\)
2. for \(i \leftarrow 1, \ldots, \text{MaxIter}\) do
3.     for all \((x,y) \in D\) do
4.         if \(y(w^T x + b) \leq 1\) then
5.             \(g \leftarrow g + y x\) \hspace{1cm} \(// \text{update weight gradient}\)
6.             \(g \leftarrow g + y\) \hspace{1cm} \(// \text{update bias derivative}\)
7.         end if
8.     end for
9.     \(w \leftarrow w + \lambda g\) \hspace{1cm} \(// \text{add in regularization term}\)
10.    \(b \leftarrow b + \lambda g\) \hspace{1cm} \(// \text{update weights}\)
11. end for
12. return \(w, b\)