15-884/ – Machine Learning 3: Classification

J. Zico Kolter

September 17, 2013
Classification

- **Regression**: predict continuous-valued outputs ($y_i \in \mathbb{R}$)

- **Classification**: predict discrete-valued outputs
  - Common case, binary classification: $y_i \in \{-1, 1\}$
• Binary classification: predictions have yes/no answers
  – Will the peak demand be higher than 2GW tomorrow?
  
  – Will a wind turbine operate at max capacity in the next hour?
  
  – Will this electric line reach its maximum capacity?
  
  – Is the device plugged into this outlet a refrigerator?

• Even when predicting a numerical quantity, what we really care about is often the answer to a yes/no question
Understanding building energy consumption

Buildings (residential and commercial) account for 71% of electricity consumption [Source: US EIA]
• The task: automatically identify appliances from their (individual) power signals

• Feedback about building energy consumption allows users to make more informed decisions

• Modified but similar techniques can be used to identify appliances from just *whole-building* energy signals
Power signal for refrigerator 1
Power signal for refrigerator 2
Constructing inputs from power signal
Classifying fridge 1 vs. fridge 2 using power as input
Classifying fridge 1 vs. fridge 2 using power and duration as inputs
Classification boundary from a linear classifier (here, a support vector machine).
Formal problem setting

• **Input:** \( x_i \in \mathbb{R}^n, \ i = 1, \ldots, m \)

• **Output:** \( y_i \in \{-1, +1\} \) (binary classification task)

• **Feature mapping:** \( \phi : \mathbb{R}^n \rightarrow \mathbb{R}^k \)

• **Model Parameters:** \( \theta \in \mathbb{R}^k \)

• **Predicted output:** \( \hat{y}_i \in \mathbb{R} = \theta^T \phi(x_i) \)
  
  – Intuition: for \( y = +1 \) we want \( \hat{y} > 0 \), for \( y = -1 \), \( \hat{y} < 0 \)
Loss functions

- Loss function: \( \ell : \mathbb{R} \times \{-1, +1\} \rightarrow \mathbb{R}_+ \)
  
  - Again, \( \ell(\hat{y}, y) \) measures how “good” the prediction is

- 0-1 Loss

\[
\ell(\hat{y}, y) = \begin{cases} 
0 & \text{if } y = +1 \text{ and } \hat{y} \geq 0, \text{ or if } y = -1 \text{ and } \hat{y} \leq 0 \\
1 & \text{otherwise}
\end{cases}
\]

\[
= \begin{cases} 
0 & y \cdot \hat{y} \geq 0 \\
1 & y \cdot \hat{y} < 0
\end{cases} \equiv 1\{y \cdot \hat{y} < 0\}
\]
• Unfortunately, hard to optimize 0-1 loss

• Many other loss functions are used in practice

  Hinge loss: \( \ell(\hat{y}, y) = \max\{1 - y \cdot \hat{y}, 0\} \)

  Squared hinge loss: \( \ell(\hat{y}, y) = \max\{1 - y \cdot \hat{y}, 0\}^2 \)

  Logistic loss: \( \ell(\hat{y}, y) = \log(1 + e^{-y \cdot \hat{y}}) \)

  Exponential loss: \( \ell(\hat{y}, y) = e^{-y \cdot \hat{y}} \)
Common loss functions for classification
Some typical classification algorithms

• Logistic Regression: minimize logistic loss

\[
\min_{\theta} \sum_{i=1}^{m} \log (1 + \exp (-y_i \cdot \theta^T \phi(x_i)))
\]

  - Probabilistic interpretation: \( p(y_i = +1|x_i) = \frac{1}{1+\exp(-\theta^T \phi(x_i))} \)

• Support vector machine (SVM): minimize (regularized) hinge loss

\[
\min_{\theta} \sum_{i=1}^{m} \max \{0, 1 - y_i \cdot \theta^T \phi(x_i)\} + \lambda \|\theta\|_2^2
\]

  - If you’ve seen SVMs before, you may have seen them described geometrically in terms of maximizing the margin of the linear classifier; that formulation is equivalent to the above
• YALMIP code for logistic regression

\[
\text{theta} = \text{sdpvar}(n,1);
\text{solvesdp([], sum(log(1+exp(-y.*(Phi*theta)))))};
\]

• YALMIP code for SVM

\[
\text{theta} = \text{sdpvar}(n,1);
\text{solvesdp([], sum(max(0,1-y.*(Phi*theta))) + ... \lambda*norm(\text{theta})^2)};
\]
Classification boundary from a support vector machine
Optimizing loss functions in classification

- YALMIP is great for rapid prototyping, and medium-size problems.

- For larger problems, we might prefer specialized solution methods, like we used for least-squares.

- Logistic regression:

  \[
  J(\theta) = \sum_{i=1}^{m} \log \left( 1 + \exp \left( -y_i \cdot \theta^T \phi(x_i) \right) \right)
  \]

  - Differentiable, but cannot analytically solve for \( \nabla_\theta J(\theta) = 0 \)
Newton’s method

- Newton’s method is a root-finding algorithm
  - i.e., for some vector-input, vector output function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, it finds $z \in \mathbb{R}^n$ such that $f(z) = 0$

- 1D case: $f : \mathbb{R} \rightarrow \mathbb{R}$
  - Given some initial $z$, repeat $z \leftarrow z - f(z)/f'(z)$;
• To extend to the multi-variate case, for a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
we’ll define the *Jacobian* $D_z f(z)$,

$$D_z f(z) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \ldots & \frac{\partial f_1(z)}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(z)}{\partial z_1} & \ldots & \frac{\partial f_m(z)}{\partial z_n} \end{bmatrix}$$

• Jacobian is like the gradient, but also defined for vector-valued functions
  – For scalar valued functions, $D_z f(z) \in \mathbb{R}^{1 \times n} = (\nabla_z f(z))^T$

• Multi-variate Newton’s method: for $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$

Repeat: $z \leftarrow z - (D_z f(z))^{-1} f(z)$
• Newton’s method applied to optimization: apply Newton’s method to find $z$ such that $\nabla_z f(z) = 0$

• The Hessian is a matrix of second derivatives of a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$

$$\nabla^2_z f(z) \in \mathbb{R}^{n \times n} = \begin{bmatrix} \frac{\partial^2 f(z)}{\partial z_1^2} & \cdots & \frac{\partial^2 f(z)}{\partial z_1 \partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(z)}{\partial z_n \partial z_1} & \cdots & \frac{\partial^2 f(z)}{\partial z_n^2} \end{bmatrix} = D_z(\nabla_z f(z)) \quad \text{(Jacobian of the gradient)}$$

• Newton’s method update:

Repeat: $z \leftarrow z - (\nabla^2_z f(z))^{-1} \nabla_z f(z)$
• Logistic regression:

\[ J(\theta) = \sum_{i=1}^{m} \log(1 + \exp(-y_i \cdot \theta^T \phi(x_i))) \]

• Gradient and Hessian given by:

\[ \nabla_\theta J(\theta) = -\Phi^T Z y \]
\[ \nabla^2_\theta J(\theta) = \Phi^T Z (I - Z) \Phi \]

where

\[ Z \in \mathbb{R}^{m \times m} \text{ diagonal, } Z_{ii} = \frac{1}{1 + \exp(y_i \cdot \theta^T \phi(x_i))} \]
• MATLAB code for logistic regression

```matlab
function theta = logreg(Phi,y)
k = size(Phi,2);
theta = zeros(k,1);
g = 1;

while (norm(g) > 1e-10)
    z = 1./(1 + exp(y.*(Phi*theta)));
g = -Phi'*z.*y;
H = Phi'*diag(z.*(1-z))*Phi;
theta = theta - H \ g;
end
```

• YALMIP code: 20.9 seconds, logreg.m: 0.016 seconds
  \((m = 300, n = 3)\)

• SVMs are a bit harder to optimize with custom routines; lots of free libraries available (libsvm, svm-light)
Newton’s method, iteration 1
Newton’s method, iteration 3
Newton's method, iteration 10
Progress of Newton’s method
Non-linear classification

- Same idea as for linear regression: non-linear features, either explicit or using kernels

Classifying refrigerator vs. all other devices
• Key component of kernels is still just the replacement

$$\theta = \sum_{j=1}^{m} \alpha_j \phi(x_j)$$

• YALMIP code for kernelized SVM:

```matlab
K = (X*X' + 1).^d; % polynomial kernel
K = exp(-sqdist(X',X')/(2*sig^2)); % Gaussian kernel
alpha = sdpvar(m,1);  
solvesdp([], lambda*alpha'*K*alpha + ...  
   sum(max(1 - y.*(K*alpha), 0)));
```

• Can derive Newton’s method for kernelized formulation as well
Kernelized SVM, Gaussian kernel
Kernelized SVM, Gaussian kernel (smaller bandwidth)