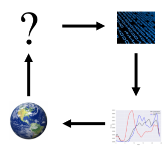
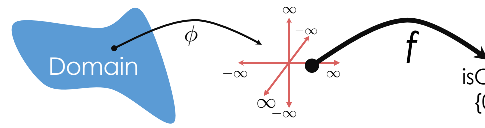
Finishing
Logistic Regression
Classification with Linear Models

Slides by:

Joseph E. Gonzalez
jegonzal@cs.berkeley.edu

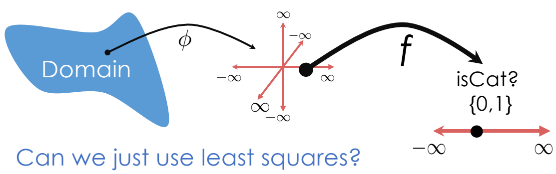


Classification



Predicting categorical responses variables

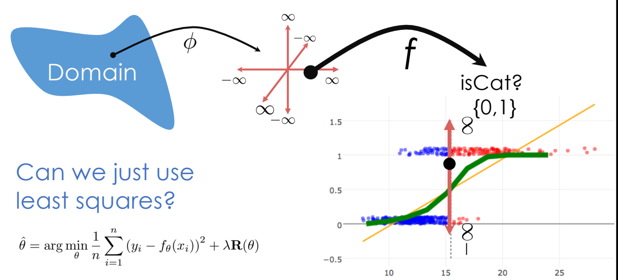
Classification



Can we just use least squares?

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2 + \lambda \mathbf{R}(\theta)$$

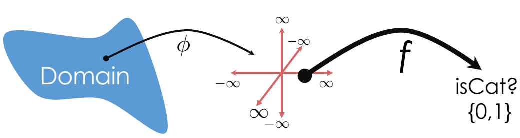
Classification



Can we just use least squares?

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2 + \lambda \mathbf{R}(\theta)$$

Classification



Can we just use least squares?

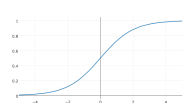
$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2 + \lambda \mathbf{R}(\theta)$$

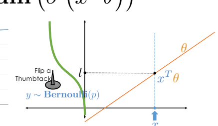
- Yes … (can be easy to compute)
- Need a decision function (e.g., f(x) > 0.5) …
- Difficult to interpret model …

Recap: Logistic Regression

- Defined a model: $y \sim \textbf{Bernoulli}\left(\sigma\left(x^T\theta\right)\right)$

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$





- Constructed a likelihood function:

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} \sigma\left(x_i^T\theta\right)^{y_i} \left(1 - \sigma\left(x_i^T\theta\right)\right)^{(1-y_i)}$$

- Constructed a likelihood function:

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} \sigma\left(x_i^T\theta\right)^{y_i} \left(1 - \sigma\left(x_i^T\theta\right)\right)^{(1-y_i)}$$

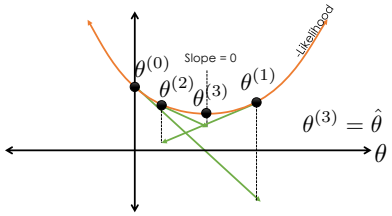
- Computed the gradient of the log-likelihood

$$\nabla_\theta \log \mathcal{L}(\theta) = \sum_{i=1}^{n} \left(y_i - \sigma\left(x_i^T\theta\right)\right) x_i$$

  - Set equal to 0 and solve … no analytic solution ☹

- Gradient Descent

Gradient Descent Intuition



Simple, fast, and works well in high dimensions.

- Gradient Descent

$\theta^{(0)} \leftarrow$ random initial vector

For τ from 0 to convergence

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau)\nabla_\theta(-\log \mathcal{L}(\theta))\Big|_{\theta^{(\tau)}}$$ Evaluated at

$$\nabla_\theta\left(-\log \mathcal{L}(\theta)\right) = \sum_{i=1}^{n} \left(\sigma\left(x_i^T\theta\right) - y_i\right) x_i$$

Maximum Likelihood → Minimize ***Negative*** Log-likelihood

- Gradient Descent

$\theta^{(0)} \leftarrow$ random initial vector

For τ from 0 to convergence

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau)\nabla_\theta(-\log \mathcal{L}(\theta))\Big|_{\theta^{(\tau)}}$$ Evaluated at

- Stochastic Gradient Descent (SGD)

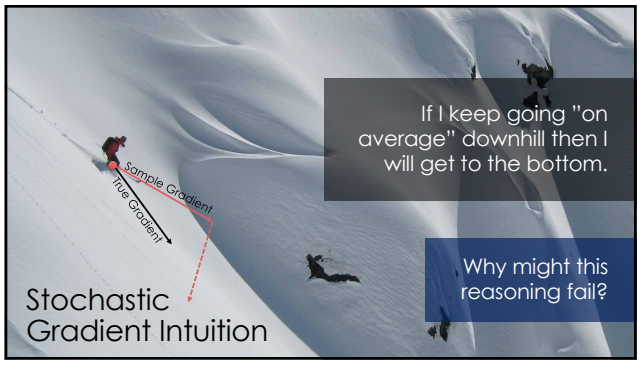
$\theta^{(0)} \leftarrow$ random initial vector

For τ from 0 to convergence

$\mathcal{B} \leftarrow$ select k random indices

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau)\frac{n}{k}\sum_{i\in\mathcal{B}} \nabla_\theta(-\log \mathcal{L}(\theta \mid x_i, y_i))\Big|_{\theta^{(\tau)}}$$ Eval. at

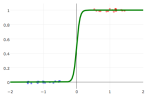
Estimate the gradient by **sampling** the data



Python Demo!

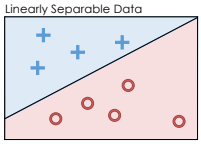
Linearly Separable Data

- A classification dataset is said to be linearly separable if there exists a hyperplane that separates the two classes.
- If data is linearly separable, logistic regression requires regularization

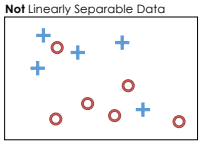


Weights go to infinity!

Linearly Separable Data



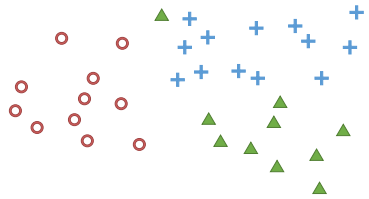
Not Linearly Separable Data



Python Demo!

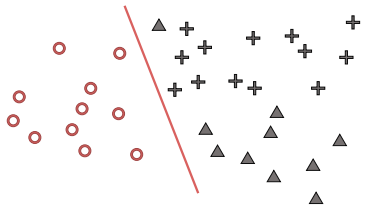
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



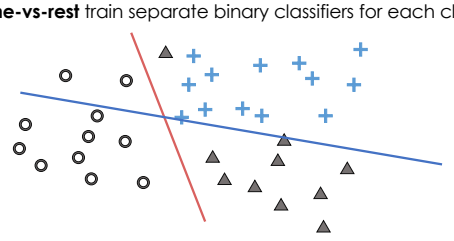
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



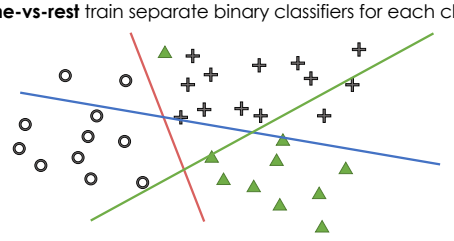
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



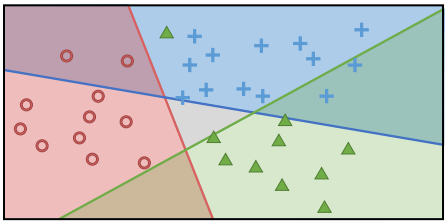
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



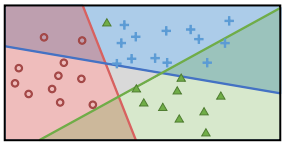
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class
  - Class with highest confidence wins
  - Need to address class imbalance issue



- **Soft-Max** multiclass classification

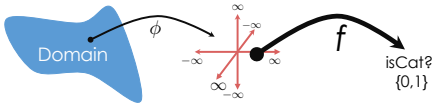
- **Soft-Max** multiclass classification

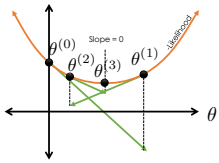
$$\mathbf{P}(Y = j \mid x) = \frac{\exp\left(x^T\theta^{(j)}\right)}{\sum_{m=1}^{k}\exp\left(x^T\theta^{(m)}\right)}$$

- Separate $\theta^{(j)} \in \mathbb{R}^p$ for each class
- Trained using gradient descent methods
- Generates calibrated probabilities

Python Demo!

Summary of Lecture

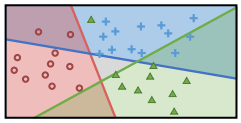




Gradient Descent
& Stochastic Gradient Descent

```
In [28]: def gradient_descent(X, Y, theta0, gradient_function, max_iter = 1000000,
                               epsilon=0.0001):
             theta = theta0
             for t in range(1, max_iter):
                 rho = 1./t
                 grad = gradient_function(theta, X, Y)
                 theta = theta - rho * grad
                 # Track Convergence
                 if np.linalg.norm(grad) < epsilon:
                     return theta
         return theta
```

Python
Logistic Regression



Multiclass
Classification
- One-vs-Rest
- Softmax

Deep Learning Overview

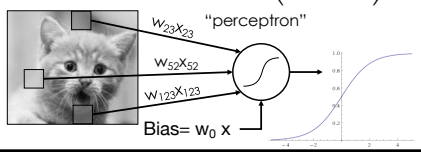
Bonus Material not on Exams!

Borrowed heavily from excellent talks by:
- **Adam Coates:** http://ai.stanford.edu/~acoates/coates_dltutorial_2013.pptx
- **Fei-Fei Li and Andrej Karpathy:** http://cs231n.stanford.edu/syllabus.html

Logistic Regression as a "Neuron"

- Consider the simple function family: $\sigma(u) = \frac{1}{1+\exp(-u)}$

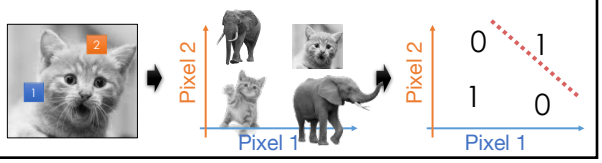
$$f_w(x) = \sigma\left(w^T x\right) = \sigma\left(\sum_{j=1}^{d} w_j x_j\right) = P(y=1 \mid x)$$



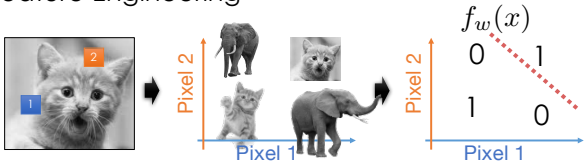
Neuron "fires" if weighted sum of input is greater than zero.

Logistic Regression: Strengths and Limitations

- Widely used machine learning technique
  - convex → efficient to learn
  - easy to interpret model weights
  - works well given good features
- Limitations:
  - Restricted to linear relationships → sensitive to choice of features

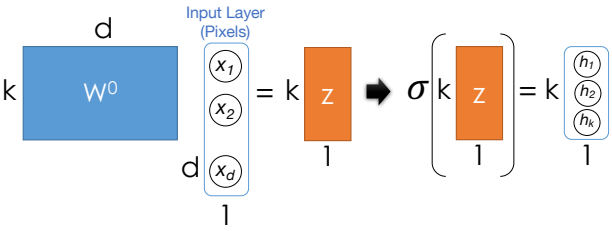


Feature Engineering

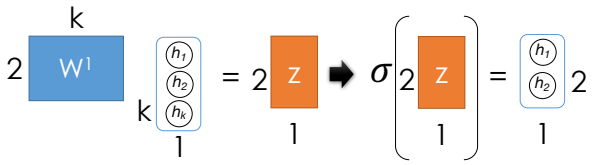


- Rather than use raw **pixels build/train feature functions**:

Composition Linear Models and Nonlinearities



Composition Linear Models and Nonlinearities

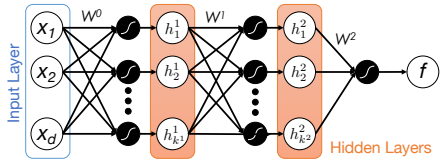


Neural Networks

- Composing "perceptrons"

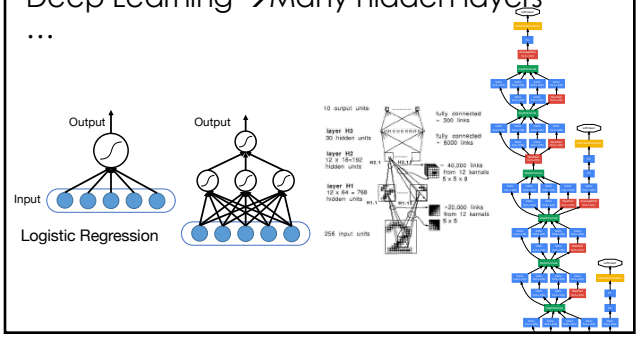
$$x \to \sigma(W^0 x) \to h^1 \to \sigma(W^1 h^1) \to h^2 \to \sigma(W^2 h^2) \to f$$

$$y = f_{W^0,W^1,W^2}(x) = \sigma\left(W^2\, \sigma\left(W^1\, \sigma\left(W^0 x\right)\right)\right)$$



Deep Learning →Many hidden layers

…



Interested in Deep Learning?

- RISE Lab Deep Learning Overview:
  - https://ucbrise.github.io/cs294-rise-fa16/deep_learning.html
- TensorFlow Python Tutorial
- Stanford CS231 Labs
  - http://cs231n.github.io/linear-classify/
  - http://cs231n.github.io/optimization-1/
  - http://cs231n.github.io/optimization-2/