15-780 – Machine Learning

J. Zico Kolter

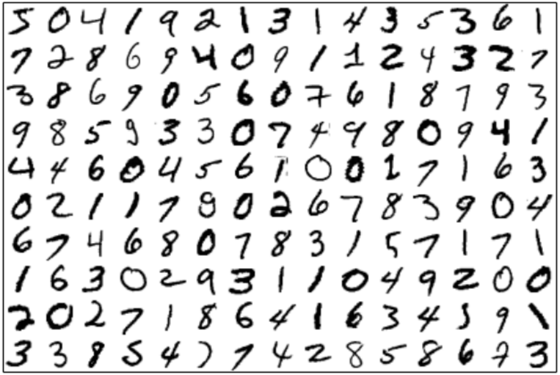
February 19, 2014

Outline

Outline

Introduction: digit classification

- The task: write a program that, given a 28x28 grayscale image of a digit, outputs the string representation



Digits from MNIST dataset
(http://yann.lecun.com/exdb/mnist/)

- One approach: try to write a program by hand that uses your a priori knowledge of digits to properly classify the images
- Alternative method (machine learning): collect a bunch of images and their corresponding digits, write a program that uses this data to build its own method for classifying images
- (This is actually a subclass of ML class **supervised learning**; we will briefly talk about other frameworks)

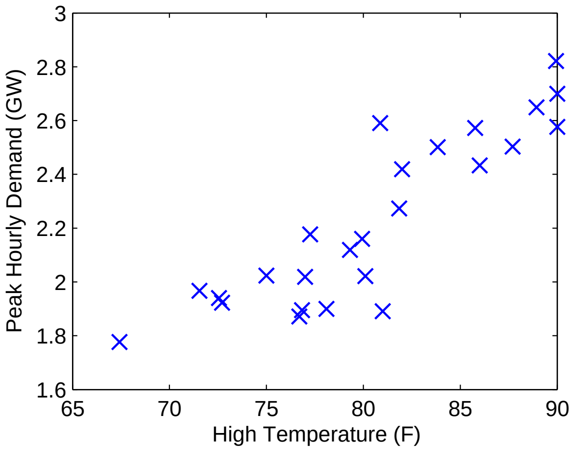
Outline

A simple example: predicting electricity use

- What will peak power consumption be in the Pittsburgh area tomorrow?

- Collect data of past high temperatures and peak demands

High Temperature (F)	**Peak Demand (GW)**
76.7	1.87
72.7	1.92
71.5	1.96
86.0	2.43
90.0	2.69
87.7	2.50
⋮	⋮



Several days of peak demand vs. high temperature in Pittsburgh

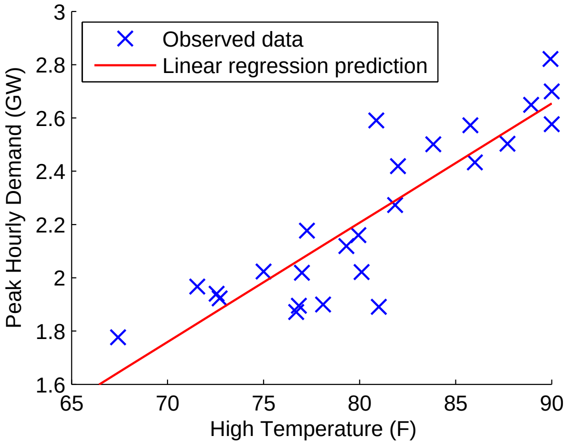
- Hypothesize model

$$\text{Peak demand} \approx \theta_1(\text{High temperature}) + \theta_2$$

  for some numbers θ_1 and θ_2

- Then, given a forecast of tomorrow's high temperature, we can predict the likely peak demand by plugging it into our model

- Equivalent to “drawing a line through the data”



Notation

- **Input features**: $x_i \in \mathbb{R}^n, \;\; i = 1, \ldots, m$
  - E.g.: $x_i \in \mathbb{R}^2 = \begin{bmatrix} \text{high temperature for day } i \\ 1 \end{bmatrix}$

- **Output**: $y_i \in \mathbb{R}$ (*regression* task)
  - E.g.: $y_i \in \mathbb{R} = \{\text{peak demand for day } i\}$

- **Model Parameters**: $\theta \in \mathbb{R}^n$

- **Hypothesis function**: $h_\theta(x) : \mathbb{R}^n \to \mathbb{R}$
  - Hypothesis function $h_\theta(x_i)$ returns a *prediction* of the output y_i, and we will focus initially on *linear predictors*

$$h_\theta(x_i) = \sum_{j=1}^{n} \theta_j (x_i)_j = \theta^T x_i$$

- **Loss function**: $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$
  - $\ell(y_i, h_\theta(x_i))$ is a "penalty" we pay for predicting $h_\theta(x_i)$ when the true output is y_i
  - E.g., squared loss: $\ell(y_i, h_\theta(x_i)) = (h_\theta(x_i) - y_i)^2$

- **Canonical supervised learning problem**: given a collection of input features and outputs (x_i, y_i), $i = 1, \ldots, m$, find parameters that minimize sum of losses over all examples

$$\underset{\theta}{\text{minimize}} \;\; \sum_{i=1}^{m} \ell(y_i, h_\theta(x_i))$$

- Virtually all machine learning algorithms have this form, just different choices of hypothesis and loss functions

Least squares revisited

- A linear hypothesis, $h_\theta(x_i) = \theta^T x_i$, and squared loss function, $\ell(y_i, h_\theta(x_i)) = (h_\theta(x_i) - y_i)^2$, lead to the least-squares problem we saw in the optimization lecture

- Defining the matrices

$$X \in \mathbb{R}^{m \times n} = \begin{bmatrix} — & x_1^T & — \\ — & x_2^T & — \\ & \vdots & \\ — & x_m^T & — \end{bmatrix}, \quad y \in \mathbb{R}^m = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

then

$$\sum_{i=1}^m \ell(y_i, h_\theta(x_i)) = \sum_{i=1}^m (\theta^T x_i - y_i)^2 = \|X\theta - y\|_2^2$$

- Thus, finding the best parameters θ for linear predictor and squared loss is optimization problem

$$\underset{\theta}{\text{minimize}}\ \|X\theta - y\|_2^2$$

- This is a convex optimization problem, so can be solved by a number of methods (e.g., gradient descent, `cvxpy`)

- However, this special case can also be solved analytically by taking gradients

$$\nabla_\theta \|X\theta - y\|_2^2 = 2X^T(X\theta - y)$$

  and setting them equal to zero

$$X^T(X\theta^\star - y) = 0 \implies \theta^\star = (X^TX)^{-1}X^Ty$$

- Python code for solving the least-squares problem

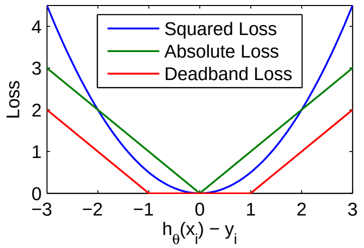
```
# load data files
X = np.mat(np.loadtxt('temperature.txt',ndmin=2))
y = np.mat(np.loadtxt('peak_demand.txt',ndmin=2))

# append a column of ones to X and solve for theta
X = np.hstack((X,np.ones((X.shape[0],1))))
theta = np.linalg.solve(X.T * X, X.T * y)
```

Alternative loss functions

- Why did we choose the squared loss functions $\ell(y_i, h_\theta(x_i)) = (h_\theta(x_i) - y_i)^2$?

- Some other alternatives

Absolute loss:	$\ell(y_i, h_\theta(x_i)) = \lvert h_\theta(x_i) - y_i \rvert$
Deadband loss:	$\ell(y_i, h_\theta(x_i)) = \max\{0, \lvert h_\theta(x_i) - y_i \rvert - \epsilon\},\ \epsilon \in \mathbb{R}_+$



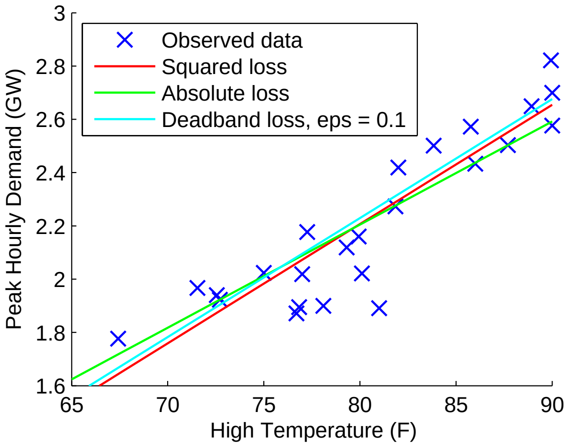
- For most loss functions other than squared loss, can't analytically find optimal θ, but for convex loss, still a convex optimization problem

$$\text{E.g. } \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} |h_\theta(x_i) - y_i|$$

- Intuitively, losses based on absolute loss are less sensitive to outliers (known as *robust* loss functions), analogous to mean vs. median

- Many of the combinations have fancy names: minimizing deadband loss $\equiv$ "support vector regression"

- In many cases (e.g., when there is a “good” model for all the data), different loss functions lead to fairly similar models



Outline

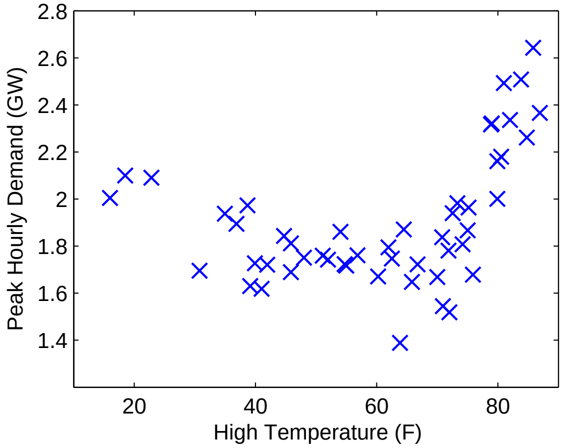
Overfitting

- Though they may seem limited, linear hypothesis classes are very powerful, since the input features can themselves include non-linear features of data

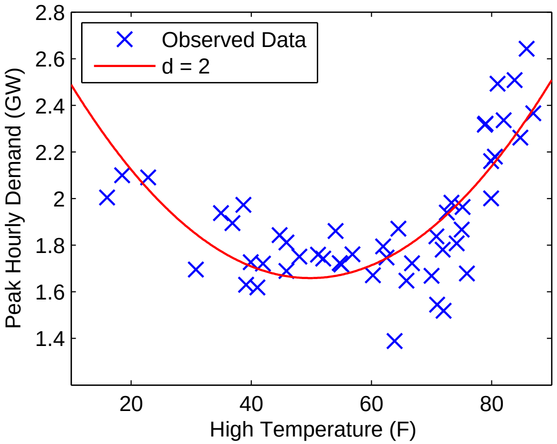
$$x_i \in \mathbb{R}^3 = \left[\begin{array}{c} (\text{high temperature for day } i)^2 \\ \text{high temperature for day } i \\ 1 \end{array} \right]$$

- In this case, $h_\theta(x_i) = \theta^T x_i$ will be a non-linear function of "original" data (i.e., predicted peak power is a a non-linear function of high temperature)

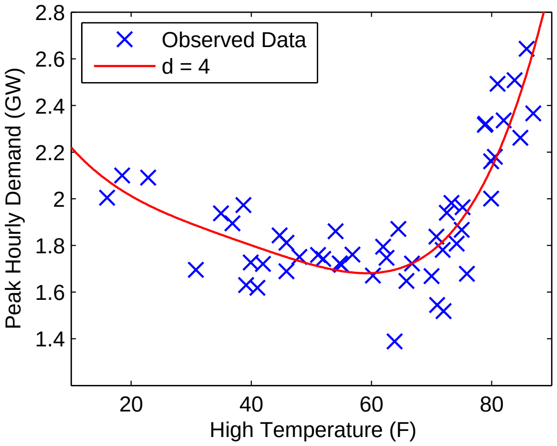
- For least-squares loss, optimal parameters still $\theta^\star = (X^T X)^{-1} X^T y$



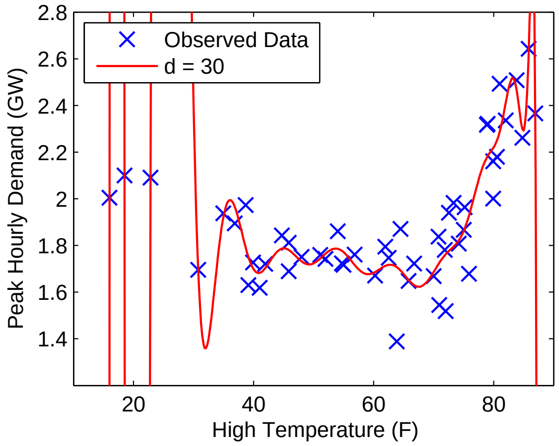
Several days of peak demand vs. high temperature in Pittsburgh over all months



Linear regression with second degree polynomial features



Linear regression with fourth degree polynomial features



Linear regression with 30th degree polynomial features

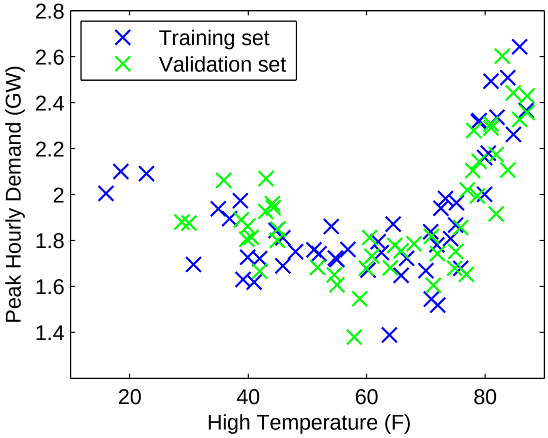
Training and validation loss

- Fundamental problem: we are optimizing parameters to solve

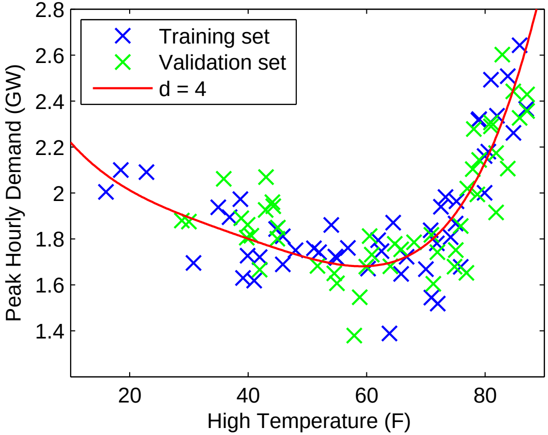
$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \ell(y_i, h_\theta(x_i))$$

  but what we really care about is loss of prediction on *new* examples (x', y') (also called *generalization error*)

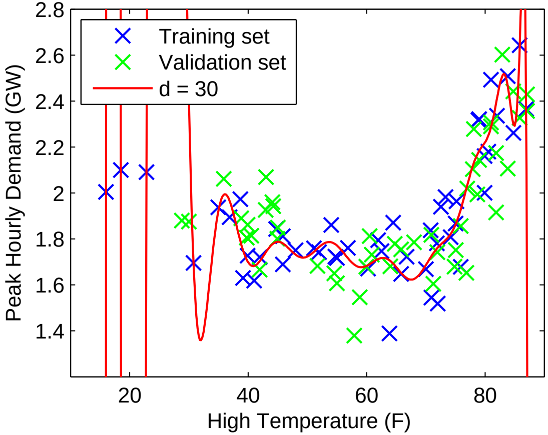
- Divide data into *training set* (used to find parameters for a fixed hypothesis class h_θ), and *validation set* (used to choose hypothesis class)
  - (Slightly abusing notation here, we're going to wrap the "degree" of the input features int the hypothesis class h_θ)



Training set and validation set

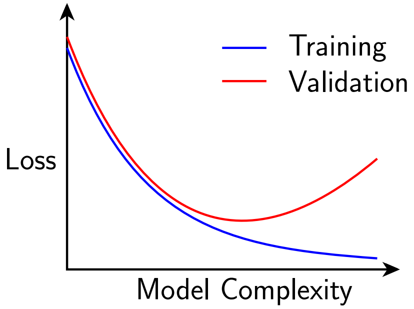


Training set and validation set, fourth degree polynomial

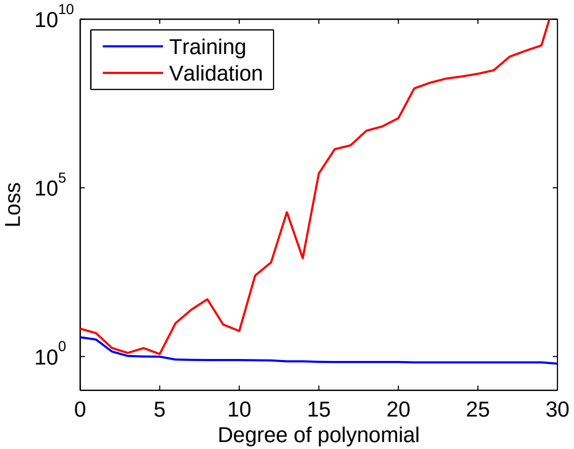


Training set and validation set, 30th degree polynomial

- General intuition for training and validation loss



- We would like to choose hypothesis class that is at the "sweet spot" of minimizing validation loss

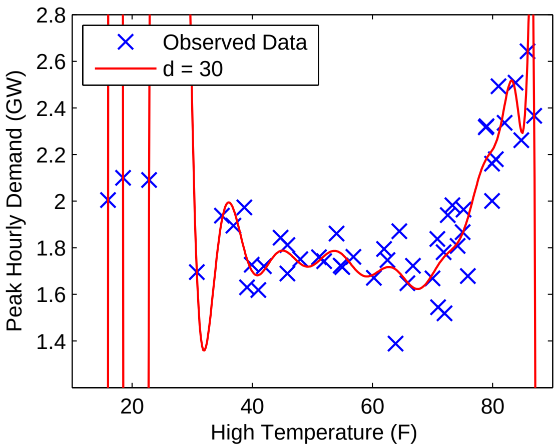


Training and validation loss on peak demand prediction

Model complexity and regularization

- A number of different ways to control “model complexity”
- An obvious one we have just seen: keep the number of features (number of parameters) low
- A less obvious method: keep the *magnitude* of the parameters small

- Intuition: a 30th degree polynomial that passes exactly through many of the data points requires very large entries in θ



- We can directly prevent large entries in θ by penalizing $\|\theta\|_2^2$ in our optimization objective

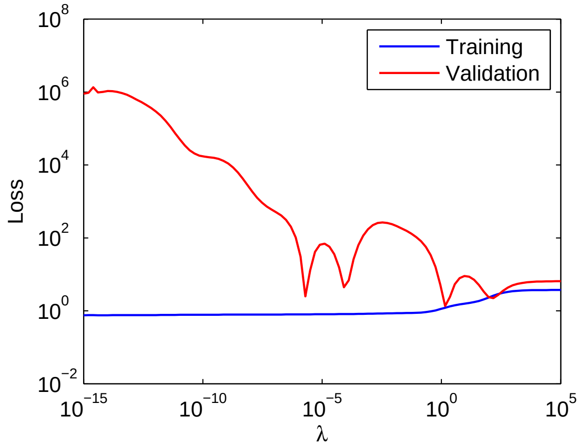
- Leads to *regularized loss minimization* problem

$$\operatorname*{minimize}_{\theta} \ \lambda\|\theta\|_2^2 + \sum_{i=1}^{m} \ell(y_i, h_\theta(x_i))$$

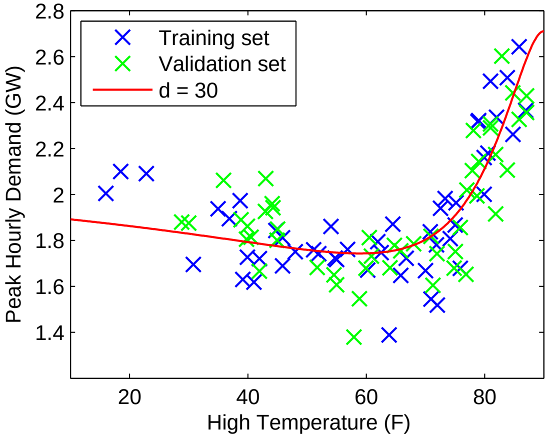
  where $\lambda \in \mathbb{R}_+$ is a *regularization parameter* that weights the relative penalties of the size of θ and the loss

- Example: regularized squared loss

$$\begin{aligned}
&\operatorname*{minimize}_{\theta} \ \lambda\|\theta\|_2^2 + \|X\theta - y\|_2^2 \\
&\implies \nabla_\theta(\|\theta\|_2^2 + \|X\theta - y\|_2^2) = 2\theta + 2X^T(X\theta - y) \\
&\implies \theta^\star = (X^T X + \lambda I)^{-1} X^T y
\end{aligned}$$



Loss for 30 degree polynomial, different values of λ



Degree 30 polynomial, with $\lambda = 1$

- For other convex loss functions, regularized loss minimization is still a convex problem

$$\underset{\theta}{\text{minimize}}\ \lambda\|\theta\|_2^2 + \sum_{i=1}^{m} \ell(y_i, h_\theta(x_i))$$

- Also can use other norms to measure magnitude of θ
  - ℓ_1 norm, $\|\theta\|_1 = \sum_{i=1}^{n} |\theta_i|$ is a popular choice because it often leads to *sparse* solutions, which indicates which input features are "important"

Outline

Classification problems

- Sometimes we want to predict discrete outputs rather than continuous
- Is the email spam or not? (YES/NO)
- What digit is in this image? (0/1/2/3/4/5/6/7/8/9)

Notation

- **Input features**: $x_i \in \mathbb{R}^n, \;\; i = 1, \ldots, m$
  - E.g.: $x_i \in \mathbb{R}^{784} =$ pixel values for 28x28 image

- **Output**: $y_i \in \{-1, +1\}$ (binary classification task)
  - E.g.: $y_i \in \{-1, +1\} =$ Is digit a 0?

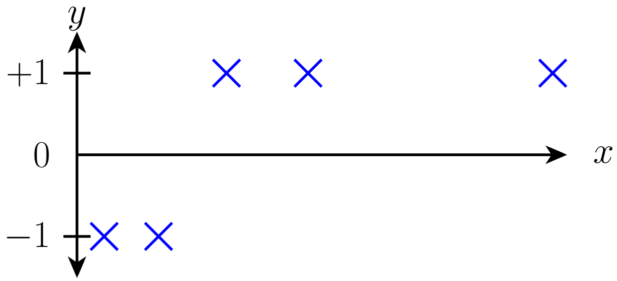
- **Model Parameters**: $\theta \in \mathbb{R}^n$

- **Hypothesis function**: $h_\theta(x) : \mathbb{R}^n \to \mathbb{R}$
  - Returns *continuous* prediction of the output y_i, where the value indicates how "confident" we are that the example is -1 or $+1$; $\mathrm{sign}(h_\theta(x_i))$ is the actual binary prediction

  - Again, we will focus initially on *linear predictors* $h_\theta(x_i) = \theta^T x_i$

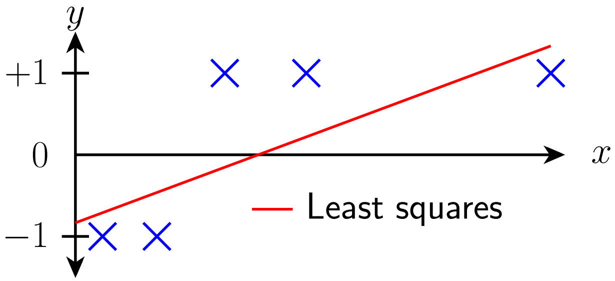
Loss functions

- Loss function $\ell : \{-1, +1\} \times \mathbb{R} \to \mathbb{R}$
- Do we need a different loss function?



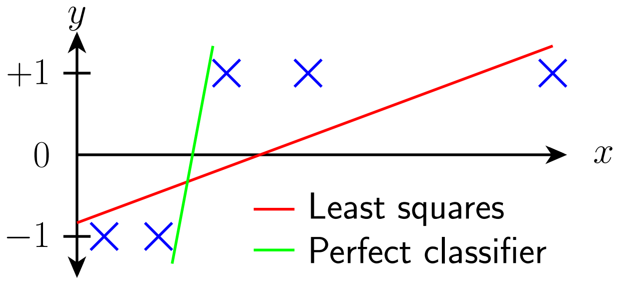
Loss functions

- Loss function $\ell : \{-1, +1\} \times \mathbb{R} \to \mathbb{R}$
- Do we need a different loss function?



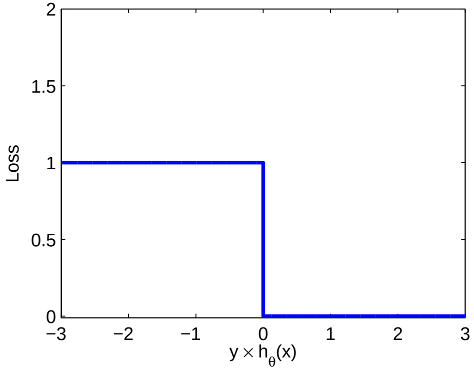
Loss functions

- Loss function $\ell : \{-1, +1\} \times \mathbb{R} \to \mathbb{R}$
- Do we need a different loss function?



- The simplest loss (0/1 loss, accuracy): count the number of mistakes we make

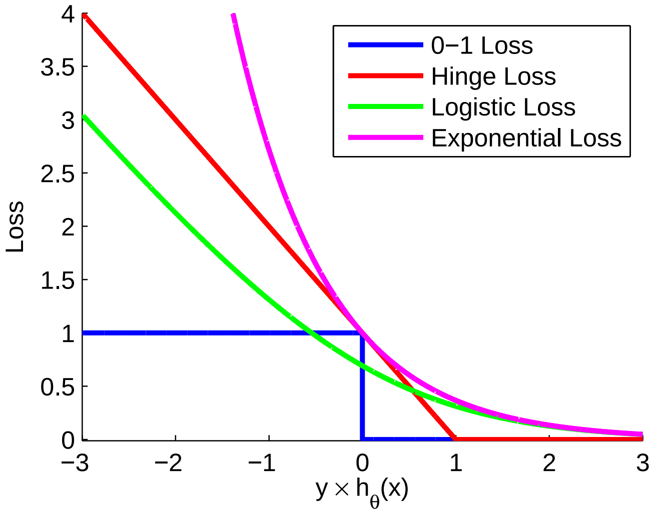
$$\ell(y, h_\theta(x)) = \begin{cases} 1 & \text{if } y \neq \operatorname{sign}(h_\theta(x)) \\ 0 & \text{otherwise} \end{cases}$$
$$= \mathbf{1}\{y \cdot h_\theta(x) \leq 0\}$$



- Unfortunately, minimizing sum of 0/1 losses leads to a non-convex optimization problem

- Because of this, a whole range of alternative "approximations" to 0/1 loss are used instead

$$\begin{aligned}
\text{Hinge loss:} \quad & \ell(y, h_\theta(x)) = \max\{1 - y \cdot h_\theta(x), 0\} \\
\text{Squared hinge loss:} \quad & \ell(y, h_\theta(x)) = \max\{1 - y \cdot h_\theta(x), 0\}^2 \\
\text{Logistic loss:} \quad & \ell(y, h_\theta(x)) = \log(1 + e^{-y \cdot h_\theta(x)}) \\
\text{Exponential loss:} \quad & \ell(y, h_\theta(x)) = e^{-y \cdot h_\theta(x)}
\end{aligned}$$



Common loss functions for classification

Support vector machines

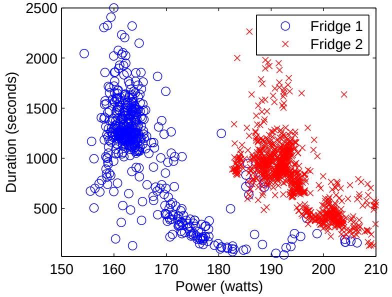
- Support vector machine is just regularized hinge loss and linear prediction (caveat, also common to use "kernel" hypothesis function, more later)

$$\underset{\theta}{\text{minimize}} \;\; \lambda\|\theta\|_2^2 + \sum_{i=1}^{m} \max\{1 - y_i \cdot \theta^T x_i, 0\}$$

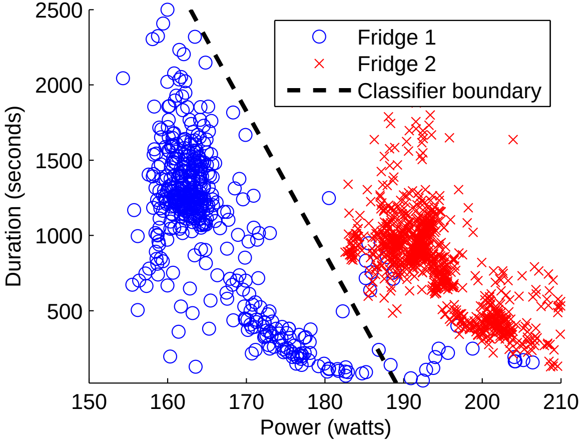
- No analytic solution, but a convex problem (can use off-the-shelf solvers like `cvxpy` for small problems)

- For large problems, methods like gradient descent are (somewhat surprisingly) almost state of the art

- Example: goal is to differentiate between two refrigerators using their power consumption signatures



- Input feature is $x_i = (i\text{th power increase}, i\text{th event duration}, 1)$



Classification boundary of support vector machine

Logistic regression

- Regularized logistic regression

$$\underset{\theta}{\text{minimize}}\ \lambda\|\theta\|_2^2 + \sum_{i=1}^{m} \log(1 + e^{-y\cdot h_\theta(x)})$$

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

- Again a (differentiable) convex problem, but no analytic solution

- Common approach: solve using Newton's method

- Optimization objective

$$f(\theta) = \lambda\|\theta\|_2^2 + \sum_{i=1}^{m} \log(1 + \exp(-y_i \cdot \theta^T \phi(x_i)))$$

- Gradient and Hessian given by (try to prove this):

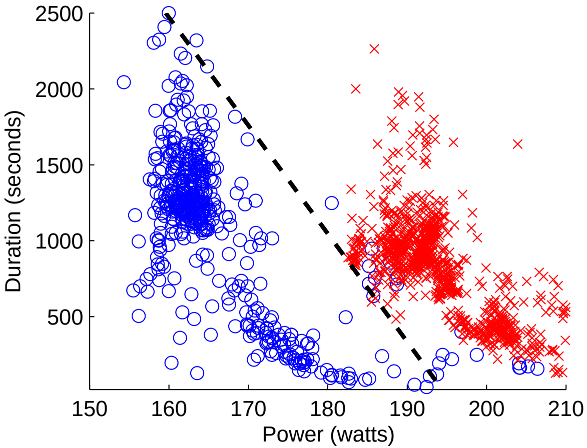
$$\nabla_\theta f(\theta) = -X^T Z y + 2\lambda\theta$$
$$\nabla_\theta^2 f(\theta) = X^T Z(I - Z)X + 2\lambda I$$

where

$$Z \in \mathbb{R}^{m\times m}\text{diagonal},\quad Z_{ii} = \frac{1}{1 + \exp(y_i \cdot \theta^T \phi(x_i))}$$

- Newton's method repeats:

$$\theta \leftarrow \theta - (\nabla_\theta^2 f(\theta))^{-1} \nabla_\theta f(\theta)$$



Classification boundary of logistic regression

Multi-class classification

- When classification is not binary $y \in 0, 1, \ldots, k$ (i.e., classifying digit images), a common approach is "one-vs-all" method

- Create a new set of y's for the binary classification problem "is the label of this example equal to j"

$$\hat{y}_i^j = \begin{cases} 1 & \text{if } y_i = j \\ -1 & \text{otherwise} \end{cases}$$

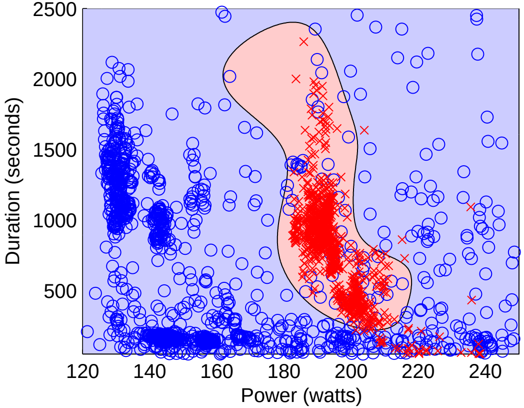
  and train the corresponding θ^j

$$\underset{\theta^j}{\operatorname{minimize}} \quad \lambda \|\theta\|_2^2 + \sum_{i=1}^{m} \ell(\hat{y}_i^j, h_{\theta^j}(x_i))$$

- For input x, classify according to the hypothesis with the highest confidence: $\operatorname{argmax}_j h_{\theta^j}(x)$

Non-linear classification

- Same exact approach as in the regression case: use non-linear features of input to capture non-linear decision boundaries



Classification boundary of support vector machine using non-linear features

Outline

- Linear hypothesis classes (with non-linear features) plus convex loss functions, are a very popular set of machine learning algorithms
- But, they are not the only possibility
- Many other algorithms that use different (possibly non-linear) hypothesis class or a different (possibly non-convex) loss function

Kernel methods

- Kernel methods are a very popular approach to non-linear classification, but they are actually still just a linear hypothesis class

$$h_\theta(x) = \sum_{i=1}^{m} \theta_i K(x, x_i)$$

  where $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a *kernel function* that measures the similarity between x and x_i (larger values for more similar)

- For certain K, can be interpreted as working in a high dimensional feature space without explicitly forming features

- Still linear in θ, can use all the same algorithms as before

- **Important**: $\theta \in \mathbb{R}^m$, as many parameters as examples (*non-parametric* approach)

Nearest neighbor methods

- Predict output based upon closest example in training set

$$h_\theta(x) = y_{\mathrm{argmin}_i \, \mathrm{dist}(x, x_i)}$$

  where $\mathrm{dist} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ is some distance function

- Can also average over k closest examples: k-nearest neighbor

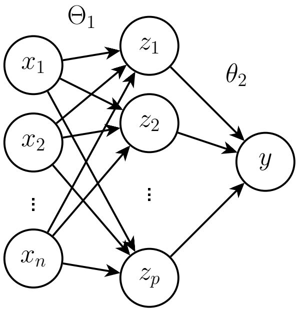
- Requires no separate “training” phase, but (like kernel methods) it is non-parametric, requires that we keep around all the data

Neural networks

- Non-linear hypothesis class

$$h_\theta(x) = \sigma(\theta_2^T \sigma(\Theta_1^T x))$$

for a 2-layer network, where $\theta = \{\Theta_1 \in \mathbb{R}^{n\times p}, \theta_2 \in \mathbb{R}^p$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is an sigmoid function $\sigma(z) = 1/(1 + \exp(-z))$ (applied elementwise to vector)



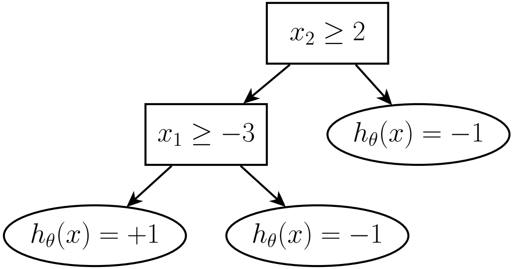
- Non-convex optimization, but smooth (gradient and similar methods can work very well)

- Neural networks are seeing a huge revival in popularity in recent years, thanks to some new algorithmic approaches (algorithms for deep architectures with many layers) and increased computational power

- Some major recent success stories in speech recognition, image classification

Decision trees

- Hypothesis class partitions space into different regions



- Can also have linear predictors (regression or classification) at the leaves

- Greedy training find nodes that best separate data into distinct classes

Ensemble methods

- Combine a number of different hypotheses

$$h_\theta(x) = \sum_{i=1}^{k} \theta_i \mathrm{sign}(h_i(x))$$

- Popular instances
  - Random forests: ensemble of decision trees built from different subsets of training data
  - Boosting: iteratively train multiple classifiers/regressors on reweighted examples based upon performance of the previous hypothesis

Unsupervised learning

- In the unsupervised setting, we don't have have pairs (x_i, y_i), $i = 1, \ldots, m$, but just the inputs x_i

- Goal is to extract some form of "structure" in the data

- Since we don't have an output, need a different form of hypothesis and loss functions

- Many unsupervised learning methods can be cast as using a *reconstruction* loss, e.g.

$$\ell(x_i, h_\theta(x_i) = \|x_i - h_\theta(x_i)\|_2^2$$

  where hypothesis function is now $h_\theta : \mathbb{R}^n \to \mathbb{R}^n$

k-means

- Parameters are a set of "centers" in the data $\theta = \{\mu_1, \ldots, \mu_k\}$ for $\mu_i \in \mathbb{R}^n$

- Hypothesis class picks the closest center

$$h_\theta(x) = \mu_{\operatorname{argmin}_i \|x-\mu_i\|_2^2}$$

- With this framework, training looks the same as supervised learning

$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \|x_i - h_\theta(x_i)\|_2^2 \quad \equiv \operatorname*{minimize}_{\mu_1,\ldots,\mu_k} \left\|x_i - \mu_{\operatorname{argmin}_j \|x-\mu_j\|_2^2}\right\|^2$$

- Not a convex problem, but can solve by iteratively finding the optimal μ_i for each example, then taking μ_i to be the mean of all examples assigned to it

Principal component analysis

- Parameters are two matrices that reduce the effective dimension of the data, $\theta = \{\Theta_1 \in \mathbb{R}^{n \times k}, \Theta_2 \in \mathbb{R}^{k \times n}$ with $k < n$

- Hypothesis class $h_\theta(x) = \Theta_1 \Theta_2 x$

- Interpretation: to reconstruct data $\Theta_2 x \in \mathbb{R}^k$ needs to preserve most of the information in x, so that we can construct it (dimensionality reduction)

- Minimizing loss

$$\operatorname*{minimize}_{\Theta_1, \Theta_2} \sum_{i=1}^{m} \|x_i - \Theta_1 \Theta_2 x_i\|_2^2$$

is not a convex problem, but can be solved (exactly) via an eigenvalue decomposition

ML implementations

- Good news is that you can use a lot of existing libraries for different machine learning approaches: e.g., `scikit.learn`

- But be careful, you usually need some understanding of how the methods work in order to properly evaluate and debug the algorithms

Outline

Introduction to machine learning

Regression

“Non-linear” regression, overfitting, and model selection

Classification

Other ML algorithms and unsupervised learning

Evaluating and debugging ML algorithms

Evaluating ML algorithms

- You have developed a machine learning approach to a certain task, and want to validate that it actually works well (or determine if it doesn't work well)

- Standard: approach, divide data into training and testing sets, train method on training set, and report results on the testing set

- Important: testing set is *not* the same as the validation test

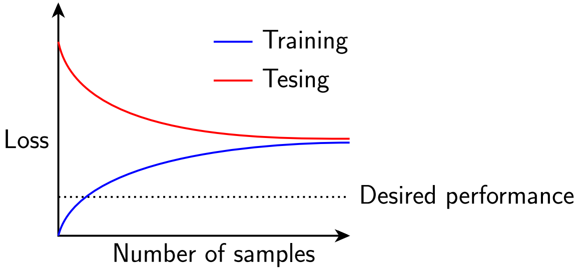
- The proper way to evaluate an ML algorithm
  1. Break all data into training/testing sets (e.g., 70%/30%)
  2. Break training set into training/validation set (e.g., 70%/30% again)
  3. Choose hyperparameters using validation set
  4. (Optional) Once we have selected hyperparameters, retrain using all the training set
  5. Evaluate performance on the testing set

Debugging machine learning algorithms

- You have built a logistic regression classifier on your application of course, and find that the training and testing error are both too high (more than the desired performance)
- Should you:
  - Add more features?
  - Collect more data?
  - Try a neural network?
  - Use a higher/lower regularization parameter?
- Can try all these things, but this will waste a lot of time

- Virtually all my time spent developing machine learning methods is spent writing diagnostics to characterize performance

- For instance, a commonly helpful diagnostic is to plot training and testing error versus number of samples



In this case, collecting more data won't help

Take home points

- Machine learning provides a way of developing complex programs by providing just input/output pairs, let the algorithm decide how best to produce output from similar inputs

- Lots of different machine learning algorithms, but the vast majority attempt to minimize some loss function on a training set, given a hypothesis class; choosing different hypotheses, loss functions, and minimization algorithms give different ML approaches

- When developing ML methods, instead of just "trying everything" always try to think of diagnostics you can write to identify the problem, then fix it