CPSC 540: Machine Learning

340 Overview

Mark Schmidt

University of British Columbia

Winter 2017
Motivating Problem: Depth Estimation from Images

- We want to build a system that predicts “distance to car” for each pixel in an image:

  https://www.gadzooki.com/gadgets/5-ways-technology-is-going-to-make-driving-safer

- For example, pixel (59, 108) has distance 30.4 meters.

- One way to build such a system:
  1. Collect a large number of images and label their pixels with the true depth.
  2. Use supervised learning to build a model that can predict depth of any pixel.
Supervised Learning Notation

- **Supervised learning** input is a set of \( n \) training examples.
- Each training example \( i \) consists of:
  - A set of features \( x^i \).
  - A label \( y^i \).
- For depth estimation:
  - Features could be a bunch of **convolutions** centered around the pixel.
  - Label would be the actual distance to the object in the pixel.
  - Supervised learning is a crucial tool used in self-driving cars.
- **Supervised learning** output is a model:
  - Usually summarized by a \( d \)-dimensional parameter vector \( w \).
  - Given a new input \( \tilde{x}^i \), model makes a prediction \( \hat{y}^i \).
  - Goal is to maximize accuracy on new examples (test error).
Supervised Learning Notation

- We’ll assume that all vectors are column-vectors,

\[
w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix}, \quad x^i = \begin{bmatrix} x^i_1 \\ x^i_2 \\ \vdots \\ x^i_d \end{bmatrix}.
\]

- I’m using \( w_j \) as the scalar parameter \( j \).
- I’m using \( y^i \) as the label of example \( i \) (currently a scalar).
- I’m using \( x^i \) as the list of features for example \( i \).
- I’m using \( x^i_j \) to denote feature \( j \) in training example \( i \).
- I’ll use \( x_j \) to denote feature \( j \) in a generic training example.
Supervised Learning Notation

- We’ll use $X$ to denote the data matrix containing the $x^i$ in the rows:

$$X = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^n)^T \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix},$$

- We’ll use $\tilde{X}$ and $\tilde{y}$ to denote test data:

$$\tilde{X} = \begin{bmatrix} (\tilde{x}^1)^T \\ (\tilde{x}^2)^T \\ \vdots \\ (\tilde{x}^n)^T \end{bmatrix}, \quad \tilde{y} = \begin{bmatrix} \tilde{y}^1 \\ \tilde{y}^2 \\ \vdots \\ \tilde{y}^n \end{bmatrix},$$

and $\hat{y}$ to denote a vector of predictions.

- Our prediction in linear models is $\hat{y}^i = w^T x^i$ (train) or $\hat{y}^i = w^T \tilde{x}^i$ (test).

- Notation alert: I use $\hat{y}^i$ whether it’s a prediction on training or test data.
MAP Estimation

- We typically fit parameters $w$ by MAP estimation,

$$\hat{w} \in \text{argmax}_{w \in \mathbb{R}^d} p(w|X, y).$$

- By Bayes rule this is equivalent to

$$\hat{w} \in \text{argmax}_{w \in \mathbb{R}^d} p(y|X, w)p(w),$$

and also equivalent to

$$\hat{w} \in \text{argmin}_{w \in \mathbb{R}^d} - \log p(y|X, w) - \log p(w).$$

see probability notes as well as notes on max and argmax on the webpage.
MAP Estimation

- If training examples $i$ are IID then first term becomes sum over examples

$$
\hat{w} \in \arg\min_{w \in \mathbb{R}^d} - \sum_{i=1}^{n} \log p(y^i|x^i, w) - \log p(w).
$$

- Gaussian likelihoods and priors are the most common choice,

$$
p(y^i|x^i, w) \propto \exp \left( -\frac{1}{2}(w^T x^i - y^i)^2 \right), \quad p(w_j) \propto \exp \left( \frac{\lambda}{2} w_j^2 \right),
$$

making MAP estimation equivalent to minimizing L2-regularized squared error,

$$
f(w) = \frac{1}{2} \sum_{i=1}^{n} (w^T x^i - y^i)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2.
$$
This is a special case of “loss plus regularizer” framework,

\[ f(w) = \sum_{i=1}^{n} f_i(w) + \lambda g(w). \]

- **Loss function** \( f_i \) measures how well we fit example \( i \) with parameters \( w \).
  - In our example \( f_i(w) = \frac{1}{2}(w^Tx_i - y_i)^2 \).
- **Regularizer** \( g \) measures how complicated the model is with parameters \( w \).
  - In our example \( r(w) = \|w\|^2 \).

- **Regularization parameter** \( \lambda > 0 \) controls strength of regularization:
  - Controls complexity of model, with large \( \lambda \) leading to less overfitting.
  - Usually set by optimizing error on a validation set or with cross-validation.
Other Loss Functions and Regularizers

- "Loss plus regularizer" framework:
  \[
  f(w) = \sum_{i=1}^{n} f_i(w) + \lambda g(w). 
  \]

- Alternative loss functions to squared error:
  - Absolute error \(|w^T x^i - y^i|\) is more robust to outliers.
  - Hinge loss \(\max\{0, 1 - y^i w^T x^i\}\) is better for binary \(y^i\).
  - Logistic loss \(\log(1 + \exp(-y^i w^T x^i))\) is better for binary \(y^i\) and is smooth.
  - Softmax loss \(-w_{y^i}^T x^i + \log(\sum_{c=1}^{k} \exp(w_c^T x^i))\) for discrete \(y^i\).

- Another common regularizer is \textbf{L1-regularization},
  \[
  g(w) = \sum_{j=1}^{d} |w_j|, 
  \]
  which encourages \textbf{sparsity} in \(w\) (many \(w_j\) are set to zero for large \(\lambda\)).
Solution of L2-Regularized Least Squares

Our L2-regularized least squares objective function was

\[
f(w) = \frac{1}{2} \sum_{i=1}^{n} (w^T x^i - y^i)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2,
\]

which we can write in matrix and norm notation as

\[
f(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2.
\]

The gradient of this quadratic objective is given by

\[
\nabla f(w) = X^T(Xw - y) + \lambda w,
\]

and setting the gradient to zero and solving for \(w\) gives

\[
w = (X^T X + \lambda I)^{-1} (X^T y),
\]

where we’ve used that \((X^T X + \lambda I)\) is invertible (we’ll show this later).
Stationary Points and Convexity

Is a stationary point (satisfying $\nabla f(w) = 0$) necessarily a global optimum?

- Yes, if the objective is convex.

In our example,

$$f(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2.$$  

- The second term is convex because squared norms are convex and $\frac{\lambda}{2} \geq 0$.
- The first term is convex because it’s a squared norm composed with the linear function $Xw - y$.
- The sum of convex functions is convex, so $f$ is convex.
Training Cost and Huge Datasets

- It costs $O(nd^2 + d^3)$ to compute the solution,
  \[ w = (X^T X + \lambda I)^{-1}(X^T y). \]

- If $d$ is huge, it might be better to use \texttt{gradient descent}.
  - It costs $O(ndt)$ to do $t$ iterations.
  - As $t$ grows it converges to a stationary point.

- If $n$ is huge, it might be better to use \texttt{stochastic gradient}.
  - It costs $O(dt)$ to do $t$ iterations.
  - As $t$ grows it converges to a stationary with decreasing step sizes.

- Next week we'll talk about how many iterations are needed.
Non-Linear Models

- Our running L2-regularized least squares example:

\[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w^T x^i - y^i)^2 + \lambda \sum_{j=1}^{d} w_j^2, \]

- To model non-linear effects we can use:
  - **Non-linear features transformations** ("change of basis"): Replace each vector \( x_i \) with a set of non-linear transformations \( z_i \).
  - **Kernel trick**: Allows some exponential- or infinite-sized \( z_i \).
  - **Sparse matrix factorization**: Unsupervised learning of the \( z_i \).
  - **Deep learning methods** like **neural networks**: Simultaneous learning of the \( z_i \) and \( w \).
Summary

- Machine learning: automatically detecting patterns in data to help make predictions and/or decisions.
- CPSC 540: advanced/difficult graduate-level 2nd or 3rd course on this topic.
- Overview of CPSC 340 topics: you are expected to know all this already.

Next time: filling in some theory gaps from 340.
“Proportional to” Probability Notation

- When we write
  \[ p(y) \propto f(y), \]
  for a probability distribution \( p \) we mean that
  \[ p(y) = \kappa f(y), \]
  where \( \kappa \) is the number needed to make \( p \) a probability.
- If \( y \) is discrete taking values in \( Y \),
  \[ \kappa = \frac{1}{\sum_{y \in Y} f(y)}. \]
- If \( y \) is continuous taking values in \( Y \),
  \[ \kappa = \frac{1}{\int_{y \in Y} f(y)}. \]