CSCI567 Machine Learning (Spring 2018)

Michael Shindler

Lecture 7: January 31, 2018

Outline

1. Constrained Optimization (MLE)
2. Review of last lecture
3. Multiclass classification
4. Linear regression redux: probabilistic interpretation
5. Summary

Outline

Constrained Optimization

General Case

- Minimize $f(x)$
- such that $g(x) = 0$

Method of Lagrange Multipliers

- $L(x, \lambda) = f(x) + \lambda g(x)$

Lagrange multipliers

1. Set derivative to zero

$$\frac{\partial L(x,\lambda)}{\partial x} = f'(x) + \lambda g'(x) = 0$$

2. Solve x in terms of λ
   $x = h(\lambda)$
3. Substitute into constraint, solve λ, then x
   $g(h(\lambda)) = 0$

Example: Dice rolls

Model

Probability of seeing a number k between 1 and 6 is $P(X = k) = \Theta_k$

Observations

$\mathcal{D} = \{x_1, x_2, \ldots x_n\} \qquad x_n \in \{1, 2, \ldots 6\}$

Likelihood

$$L(\theta) = \prod_{n=1}^{N} P(X = x_n) = \prod_{k=1}^{6} \Theta_k^{n_k}$$

Optimization

Objective function (log-likelihood)

$$\max \sum_k n_k \log \theta_k$$

Constraints

$$\sum_k \theta_k = 1 \qquad\qquad \theta_k \geq 0$$

Lagrangian (ignoring non-negative constraint)

$$L(\theta, \lambda) = \sum_k n_k \log \theta_k + \lambda(\sum_k \theta_k - 1)$$

Finding both multiplier and the parameters

Derivatives

$$\frac{\partial L(\theta, \lambda)}{\partial \theta_k} = \frac{n_k}{\theta_k} + \lambda$$

Setting them to zero

$$\theta_k = -\frac{1}{\lambda} n_k$$

Solving the multiplier by using the constraint

$$\sum_k \theta_k = -\frac{1}{\lambda} \sum_k n_k = 1 \rightarrow \lambda = -\sum_k n_k$$

Finding both multiplier and the parameters

Derivatives

$$\frac{\partial L(\theta, \lambda)}{\partial \theta_k} = \frac{n_k}{\theta_k} + \lambda$$

Setting them to zero

$$\theta_k = -\frac{1}{\lambda} n_k$$

Solving the multiplier by using the constraint

$$\sum_k \theta_k = -\frac{1}{\lambda} \sum_k n_k = 1 \rightarrow \lambda = -\sum_k n_k$$

Finally

$$\theta_k = \frac{n_k}{\sum_k n_k}$$

Outline

Logistic classification

Setup for two classes

- Input: $\boldsymbol{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$
- Model of *conditional probability*

$$p(y = 1|\boldsymbol{x}; b, \boldsymbol{w}) = \sigma[g(\boldsymbol{x})]$$

where

$$g(\boldsymbol{x}) = b + \sum_d w_d x_d = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

- Linear decision boundary

$$g(\boldsymbol{x}) = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = 0$$

Maximum likelihood estimation

Cross-entropy error (negative log-likelihood)

$$\mathcal{E}(b, \boldsymbol{w}) = -\sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
- Newton method: fast but not scalable

Each has its own strength and weakness

Gradient descent (Batch update)

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \sum_n \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n$$

Newton method

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}^{(t)^{-1}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)})$$

Stochastic Gradient descent

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n$$

Optimization

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \mathcal{E}(\boldsymbol{w})$$

- On the right-hand-size $\boldsymbol{w}$ is a variable/argument to the function $\mathcal{E}$ We are searching over all possible values (ie, the subscript to $\arg\min$) for the one that minimizes the function
- On the left-hand-size $\boldsymbol{w}$ is the search result – we call it the minimizer. We can use various ways to denote it

$$\boldsymbol{w}^*, \boldsymbol{w}^{\text{optimal}}, \boldsymbol{w}^{\text{LMS}}, \boldsymbol{w}^{\text{MLE}}, \cdots$$

Or we can give it a different name

$$\boldsymbol{v}, \boldsymbol{\theta}, \cdots$$

without changing its meaning. We tend to drop superscripts or use a different name to avoid notation cluttering. However, this does require you to determine what is being used from the context.

Matlab demo

This code is logistic regression and perceptron implemented in Matlab – your HW asks you to implement in Python.

We will show you how stochastic gradient descent is used to improve a linear classifier.

Outline

Setup

Suppose we need to predict multiple classes/outcomes: $C_1, C_2, \ldots, C_K$

- Weather prediction: sunny, cloudy, raining, etc
- Optical character recognition: 10 digits + 26 characters (lower and upper cases) + special characters, etc

Studied methods

- Nearest neighbor classifier
- Logistic regression

Logistic regression for predicting multiple classes? Easy

The approach of "one versus the rest"

- For each class C_k, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel all the rest data into NEGATIVE (or '0')

  This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

  Example: for class C_2, data go through the following change

$$(\boldsymbol{x}_1, C_1) \rightarrow (\boldsymbol{x}_1, 0), (\boldsymbol{x}_2, C_3) \rightarrow (\boldsymbol{x}_2, 0), \ldots, (\boldsymbol{x}_n, C_2) \rightarrow (\boldsymbol{x}_n, 1), \ldots,$$

Logistic regression for predicting multiple classes? Easy

The approach of "one versus the rest"

- For each class C_k, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel all the rest data into NEGATIVE (or '0')

  This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

  Example: for class C_2, data go through the following change

$$(\boldsymbol{x}_1, C_1) \rightarrow (\boldsymbol{x}_1, 0), (\boldsymbol{x}_2, C_3) \rightarrow (\boldsymbol{x}_2, 0), \ldots, (\boldsymbol{x}_n, C_2) \rightarrow (\boldsymbol{x}_n, 1), \ldots,$$

- Train K binary classifiers using logistic regression to differentiate the two classes C_k versus NotC_k.

Logistic regression for predicting multiple classes? Easy

The approach of "one versus the rest"

- For each class C_k, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel all the rest data into NEGATIVE (or '0')

  This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

  Example: for class C_2, data go through the following change

$$(\boldsymbol{x}_1, C_1) \rightarrow (\boldsymbol{x}_1, 0), (\boldsymbol{x}_2, C_3) \rightarrow (\boldsymbol{x}_2, 0), \ldots, (\boldsymbol{x}_n, C_2) \rightarrow (\boldsymbol{x}_n, 1), \ldots,$$

- Train K binary classifiers using logistic regression to differentiate the two classes C_k versus NotC_k.
- When predicting on $\boldsymbol{x}$, combine the outputs of all binary classifiers
  1. What if all the classifiers say NEGATIVE?

Logistic regression for predicting multiple classes? Easy

The approach of "one versus the rest"

- For each class C_k, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel all the rest data into NEGATIVE (or '0')

  This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

  Example: for class C_2, data go through the following change

$$(\boldsymbol{x}_1, C_1) \rightarrow (\boldsymbol{x}_1, 0), (\boldsymbol{x}_2, C_3) \rightarrow (\boldsymbol{x}_2, 0), \ldots, (\boldsymbol{x}_n, C_2) \rightarrow (\boldsymbol{x}_n, 1), \ldots,$$

- Train K binary classifiers using logistic regression to differentiate the two classes C_k versus NotC_k.
- When predicting on $\boldsymbol{x}$, combine the outputs of all binary classifiers
  1. What if all the classifiers say NEGATIVE?
  2. What if multiple classifiers say POSITIVE?

  Take-home exercise: there are different combination strategies. Can you think of any?

Yet, another easy approach

The approach of "one versus one"

- For each *pair* of classes C_k and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel training data with label $C_{k'}$ into NEGATIVE (or '0')
  3. *Disregard* all other data

  Ex: for class C_1 and C_2,

$$(\boldsymbol{x}_1, C_1), (\boldsymbol{x}_2, C_3), (\boldsymbol{x}_3, C_2), \ldots \rightarrow (\boldsymbol{x}_1, 1), (\boldsymbol{x}_3, 0), \ldots$$

Yet, another easy approach

The approach of "one versus one"

- For each *pair* of classes C_k and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel training data with label $C_{k'}$ into NEGATIVE (or '0')
  3. *Disregard* all other data

  Ex: for class C_1 and C_2,

$$(\boldsymbol{x}_1, C_1), (\boldsymbol{x}_2, C_3), (\boldsymbol{x}_3, C_2), \ldots \rightarrow (\boldsymbol{x}_1, 1), (\boldsymbol{x}_3, 0), \ldots$$

- Train $K(K-1)/2$ binary classifiers using logistic regression to differentiate the two classes

Yet, another easy approach

The approach of "one versus one"

- For each *pair* of classes C_k and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label C_k, into POSITIVE (or '1')
  2. Relabel training data with label $C_{k'}$ into NEGATIVE (or '0')
  3. *Disregard* all other data

  Ex: for class C_1 and C_2,

$$(\boldsymbol{x}_1, C_1), (\boldsymbol{x}_2, C_3), (\boldsymbol{x}_3, C_2), \ldots \rightarrow (\boldsymbol{x}_1, 1), (\boldsymbol{x}_3, 0), \ldots$$

- Train $K(K-1)/2$ binary classifiers using logistic regression to differentiate the two classes
- When predicting on $\boldsymbol{x}$, combine the outputs of all binary classifiers
  There are $K(K-1)/2$ votes! *Take-home exercise: can you think of any good combination strategies?*

Contrast these two approaches

Pros and cons of each approach

- *one versus the rest*: only needs to train K classifiers. Make a *huge* difference if you have a lot of *classes* to go through.
  Can you think of a good application example where there are a lot of classes?

Contrast these two approaches

Pros and cons of each approach

- *one versus the rest*: only needs to train K classifiers. Make a *huge* difference if you have a lot of *classes* to go through.
  Can you think of a good application example where there are a lot of classes?
- *one versus one*: only needs to train a smaller subset of data (only those labeled with those two classes would be involved). Make a *huge* difference if you have a lot of *data* to go through.

Contrast these two approaches

Pros and cons of each approach

- *one versus the rest*: only needs to train K classifiers. Make a *huge* difference if you have a lot of *classes* to go through.
  Can you think of a good application example where there are a lot of classes?
- *one versus one*: only needs to train a smaller subset of data (only those labeled with those two classes would be involved). Make a *huge* difference if you have a lot of *data* to go through.

Bad about both of them

Combining classifiers' outputs seem to be a bit tricky.

Any other good methods?

Multinomial logistic regression

From binary logistic regression

$$p(y = 1|\boldsymbol{x}) = \sigma[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b_1]$$

To multi-class, defining the following model for conditional probability

$$p(y = c|\boldsymbol{x}) = \sigma[\boldsymbol{w}_c^{\mathrm{T}}\boldsymbol{x} + b_c]$$

Would this work?

Multinomial logistic regression

From binary logistic regression

$$p(y = 1|\boldsymbol{x}) = \sigma[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b_1]$$

To multi-class, defining the following model for conditional probability

$$p(y = c|\boldsymbol{x}) = \sigma[\boldsymbol{w}_c^{\mathrm{T}}\boldsymbol{x} + b_c]$$

Would this work?

This would *not* work at least for the reason

$$\sum_c p(y = c|\boldsymbol{x}) = \sum_c \sigma[\boldsymbol{w}_c^{\mathrm{T}}\boldsymbol{x} + b_c] \neq 1$$

as each summand can be any number (independently) between 0 and 1.
But we are close

Definition of multinomial logistic regression

Model

For each class C_k, we have a parameter vector $\boldsymbol{w}_k$ and model the conditional probability as

$$p(y = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}}\boldsymbol{x}}} \qquad \leftarrow \qquad \text{This is called } \textit{softmax} \text{ function}$$

Note that we have shortened the notation by "absorbing" b_c into w_c by augmenting $\boldsymbol{x}$ with a constant feature 1.

Definition of multinomial logistic regression

Model

For each class C_k, we have a parameter vector $\boldsymbol{w}_k$ and model the conditional probability as

$$p(y = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}}\boldsymbol{x}}} \qquad \leftarrow \qquad \text{This is called } \textit{softmax} \text{ function}$$

Note that we have shortened the notation by "absorbing" b_c into w_c by augmenting $\boldsymbol{x}$ with a constant feature 1.

Decision boundary: assign $\boldsymbol{x}$ with the label that is the maximum of the conditional probabilities

$$\arg\max_k p(y = k|\boldsymbol{x}) = \arg\max_k \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

Note: the notation is changed to denote the class C_k as k instead of just c

Why the name softmax?

Suppose we have

$$\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} = 100, \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} = 50, \boldsymbol{w}_3^{\mathrm{T}}\boldsymbol{x} = -20$$

we could have picked the *winning* class label 1 according to our classification rule.

Softness comes in when we compute the probability of selecting that

$$p(y=1|\boldsymbol{x}) = \frac{e^{100}}{e^{100} + e^{50} + e^{-20}} < 1$$

despite its begin the largest among the 3: $p(y=1|\boldsymbol{x}) > p(y=2|\boldsymbol{x})$ and $p(y=1|\boldsymbol{x}) > p(y=2|\boldsymbol{x})$.
We assign a probability that is *not absolute* 1, thus the name *softmax*

Sanity check

Multinomial model when $K = 2$

$$p(y=1|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}}}{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}} + e^{\boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x}}} = \frac{1}{1 + e^{-(\boldsymbol{w}_1 - \boldsymbol{w}_2)^{\mathrm{T}}\boldsymbol{x}}}$$

$$= \frac{1}{1 + e^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}} = \sigma[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}]$$

Namely, *Multinomial logistic regression thus simplifies to the (binary) logistic regression by reparameterizing the model with*

$$\boldsymbol{w} \leftarrow \boldsymbol{w}_1 - \boldsymbol{w}_2$$

Parameter estimation

Maximize conditional $\log$-likelihood

$$\log P(\mathcal{D}) = \sum_n \log P(y_n|\boldsymbol{x}_n)$$

Parameter estimation

Maximize conditional log-likelihood

$$\log P(\mathcal{D}) = \sum_n \log P(y_n|\boldsymbol{x}_n)$$

We will change y_n to $\boldsymbol{y}_n = [y_{n1}\ y_{n2}\ \cdots\ y_{nK}]^{\mathrm{T}}$, a K-dimensional vector using 1-of-K encoding.

$$y_{nk} = \begin{cases} 1 & \text{if } y_n = k \\ 0 & \text{otherwise} \end{cases}$$

Ex: if $y_n = 2$, then, $\boldsymbol{y}_n = [0\ 1\ 0\ 0\ \cdots\ 0]^{\mathrm{T}}$.

Parameter estimation

Maximize conditional log-likelihood

$$\log P(\mathcal{D}) = \sum_n \log P(y_n|\boldsymbol{x}_n)$$

We will change y_n to $\boldsymbol{y}_n = [y_{n1}\ y_{n2}\ \cdots\ y_{nK}]^{\mathrm{T}}$, a K-dimensional vector using 1-of-K encoding.

$$y_{nk} = \begin{cases} 1 & \text{if } y_n = k \\ 0 & \text{otherwise} \end{cases}$$

Ex: if $y_n = 2$, then, $\boldsymbol{y}_n = [0\ {\color{red}1}\ 0\ 0\ \cdots\ 0]^{\mathrm{T}}$.

$$\sum_n \log p(y_n|\boldsymbol{x}_n) = \sum_n \log \prod_{k=1}^{K} p(y=k|\boldsymbol{x}_n)^{y_{nk}} = \sum_n \sum_k y_{nk} \log p(y=k|\boldsymbol{x}_n)$$

Cross-entropy error function

Definition: negated likelihood

$$
\begin{aligned}
\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) &= -\sum_n \sum_k y_{nk} \log p(y = |\boldsymbol{x}_n) \\
&= -\sum_n \sum_k y_{nk} \left\{ \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} - \log \sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}} \boldsymbol{x}} \right\} \quad (1)
\end{aligned}
$$

Cross-entropy error function

Definition: negated likelihood

$$\begin{aligned}\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) &= -\sum_n \sum_k y_{nk} \log p(y = |\boldsymbol{x}_n) \\ &= -\sum_n \sum_k y_{nk} \left\{ \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} - \log \sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}} \boldsymbol{x}} \right\} \quad (1)\end{aligned}$$

Properties

- Optimization requires numerical procedures, analogous to those used for binary logistic regression
  Large-scale implementation, in both the number of classes and the training examples, is non-trivial.

Stochastic gradient descent for multinomial logistic regression

Can you fill the blank?

- Initialize $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K$ to $\boldsymbol{w}_1^{(0)}$, $\boldsymbol{w}_2^{(0)}$, . . . , $\boldsymbol{w}_K^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute the gradients of the error function with respect to the parameters
  3. Update the parameters
  4. $t \leftarrow t + 1$

Stochastic gradient descent for multinomial logistic regression

Inspiration from binary logistic regression

- Initialize ...
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute the gradients of the error function with respect to the parameters
  $$\boldsymbol{g}_n = (\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n$$
  3. Update the parameters $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\boldsymbol{g}_n$
  4. $t \leftarrow t + 1$

Stochastic gradient descent for multinomial logistic regression

- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$, *convert y_n to 1-of-K encoding y_{nk}*
  2. Compute the gradients of the error function with respect to the parameters

$$\begin{aligned}\boldsymbol{g}_{n1} &= (p(y=1|\boldsymbol{x}_n) - y_{n1})\boldsymbol{x}_n \\ \boldsymbol{g}_{n2} &= (p(y=2|\boldsymbol{x}_n) - y_{n2})\boldsymbol{x}_n \\ \cdots & \\ \boldsymbol{g}_{nK} &= (p(y=K|\boldsymbol{x}_n) - y_{nk})\boldsymbol{x}_n\end{aligned}$$

  3. Update the parameters

$$\cdots$$

$$\boldsymbol{w}_k^{(t+1)} = \boldsymbol{w}_k^{(t)} - \eta \boldsymbol{g}_{nk} \tag{2}$$

$$\cdots \tag{3}$$

Challenge

Last lecture, we have showed that logistic regression's stochastic gradient descent is similar to percpetron update — both them are "making prediction $\rightarrow$ correcting mistake by updating parameters $\rightarrow$ repeat"

We have never talked about perceptron for multi-class classification. Do you think you can start from multinomial logistic regression, and suggest what the algorithm of perceptron for multi-class classification would look like?

Outline

Linear regression

Setup

- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
- Model: $f : \boldsymbol{x} \rightarrow y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

Goal: Minimize prediction error as much as possible

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - f(\boldsymbol{x}_n)]^2 = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2$$

Why minimizing RSS is a sensible thing?

Why minimizing RSS is a sensible thing?

Probabilistic interpretation

- Noisy observation model (for simplicity, we have assumed 1-dimensional data)

$$Y = w_0 + w_1 X + \eta$$

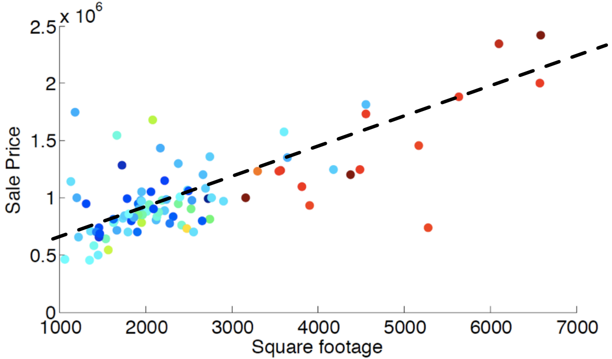
  where $\eta \sim N(0, \sigma^2)$ is a Gaussian random variable

- Likelihood of one training sample (x_n, y_n)

$$p(y_n|x_n) = N(w_0 + w_1 x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2}}$$

Possibly linear relationship

Sale price = price_per_sqft × square_footage + fixed_expense + unexplainable_stuff



Namely, we are saying the unexplainable_stuff is a Gaussian random variable

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n)$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n)$$

$$= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\begin{aligned}
\log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n) \\
&= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\
&= -\frac{1}{2\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 - \frac{\mathsf{N}}{2} \log \sigma^2 - \mathsf{N} \log \sqrt{2\pi}
\end{aligned}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$
\begin{aligned}
\log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n) \\
&= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\
&= -\frac{1}{2\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 - \frac{\mathsf{N}}{2} \log \sigma^2 - \mathsf{N} \log \sqrt{2\pi} \\
&= -\frac{1}{2} \left\{ \frac{1}{\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N} \log \sigma^2 \right\} + \text{const}
\end{aligned}
$$

i.i.d stands for independently and identically distributed.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

[1] This is not generally true but in this particular case, we can do so.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2}\left\{-\frac{1}{s^2}\sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N}\frac{1}{s}\right\} = 0$$

[1] This is not generally true but in this particular case, we can do so.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2}\left\{-\frac{1}{s^2}\sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N}\frac{1}{s}\right\} = 0$$

$$\rightarrow \sigma^{*2} = s^* = \frac{1}{\mathsf{N}}\sum_n [y_n - (w_0 + w_1 x_n)]^2$$

[1] This is not generally true but in this particular case, we can do so.

Why we want to have the probabilistic interpretation?

- It gives a solid footing to our intuition: minimizing $\mathsf{RSS}(\tilde{\boldsymbol{w}})$ is a sensible thing to do as it grows naturally out of the probabilistic model.
- The ability of having estimated σ^* — how much noise could be present in our prediction – is valuable. For example, it allows us to make confidence intervals about our predictions.

Outline

Summary

- Supervised learning
  regression and classification: continuous versus discrete outputs
- Methods
  parametric and nonparametric: linear classifier/regression versus nearest neighbor
  Linear and nonlinear: linear regression versus regression with nonlinear basis
- Learning objectives
  Probabilistic model: conditional probabilistic models for either regression or classification
  Non-probabilistic model: perceptron that minimizes a loss function

  These two objectives are called *discriminative* – we will see *generative* models for unsupervised learning later in the semester.