S&DS 265 / 565

Introductory Machine Learning

Stochastic Gradient Descent

January 28

Yale

Version of 2026/01/28 at 11:43

Reminders

- Quiz 1 will be live after class (2:30)
  - Taken online, on Canvas
  - 48 hour window
  - 20 minutes once started
  - Open notes, etc.
- Assn 1 due week from tomorrow

Outline for today

- Continue discussion of generative/discriminative
- Stochastic gradient descent
- Application to logistic regression
- Regularization
- Learning rate and scaling
- Jupyter notebook example

Two flavors of classifiers

Generative: model both the input X and the output Y.

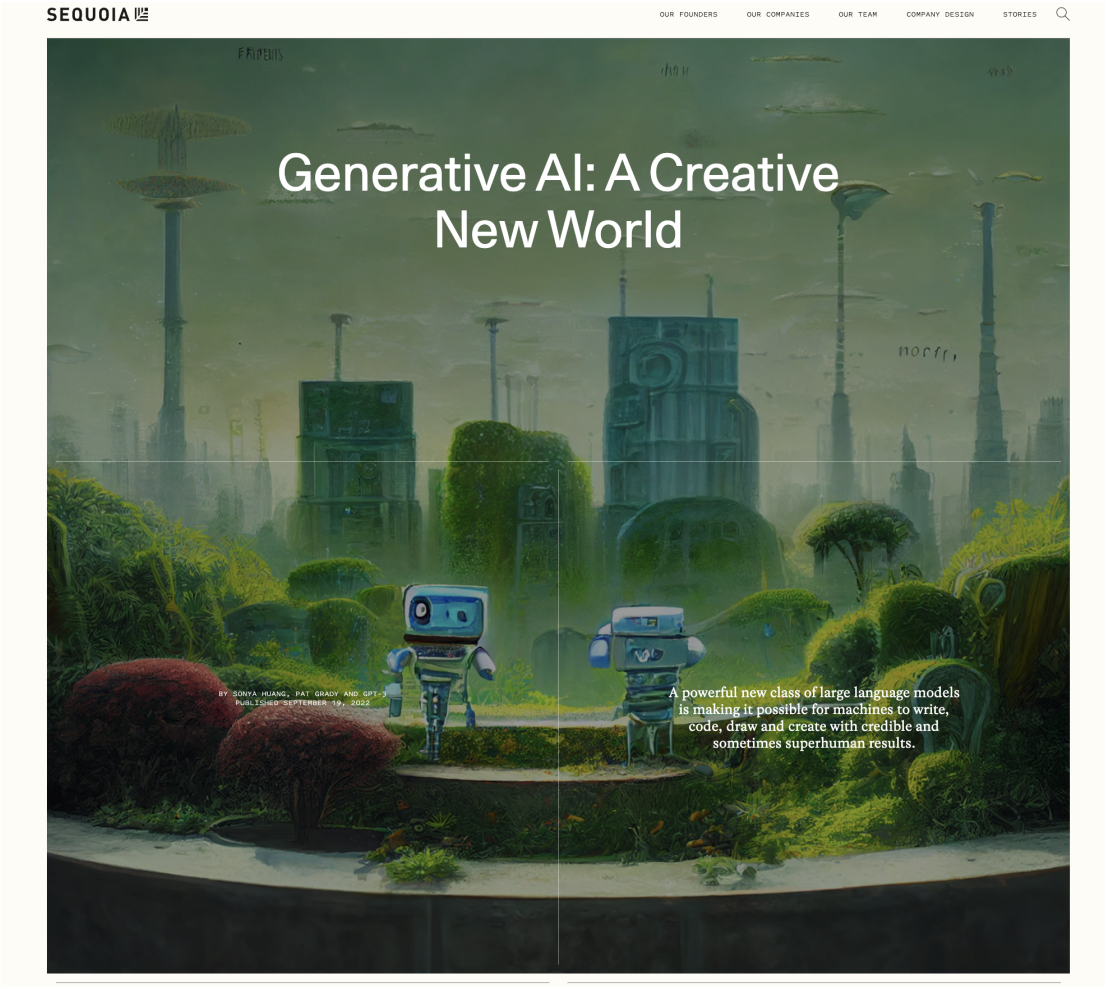
Discriminative: model only the output Y given X.

Generative models

We build a model of the inputs x and the outputs y

In the generative case we typically estimate the model by trying to maximize the *joint likelihood* $p(x, y)$ of the training data

Generative models



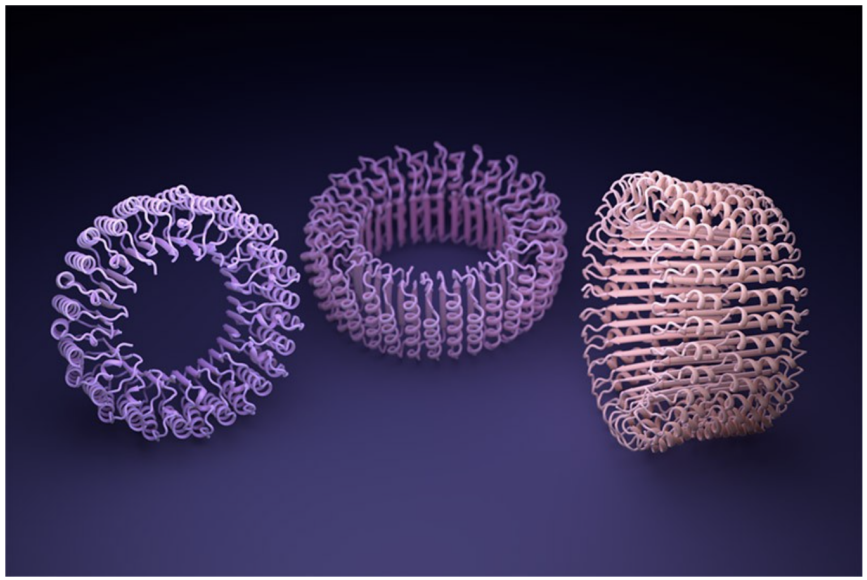
Generative models

NEWS | 15 September 2022

Scientists are using AI to dream up revolutionary new proteins

Huge advances in artificial intelligence mean researchers can design completely original molecules in seconds instead of months.

Ewen Callaway



Artificial-intelligence tools are helping to scientists to come up with proteins that are shaped unlike anything in nature. Credit: Ian C Haydon/UW Institute for Protein Design

Discriminative models

In the discriminative case we focus on the *conditional* distribution of the output given the input

We will typically estimate by maximizing the *conditional likelihood* of the training data

Bayes rule

The form of the Bayes classification rule suggests we should use a generative model

$$m_\theta(x) \equiv \mathbb{P}(Y = 1 \mid X = x) = \frac{\pi_1 p_1(x)}{(1-\pi_1)p_0(x) + \pi_1 p_1(x)}.$$

But we can target the discriminant function directly

Simplest generative model

- We model the inputs x using Gaussians
- Two flavors: Linear and Quadratic

Quadratic discriminant analysis

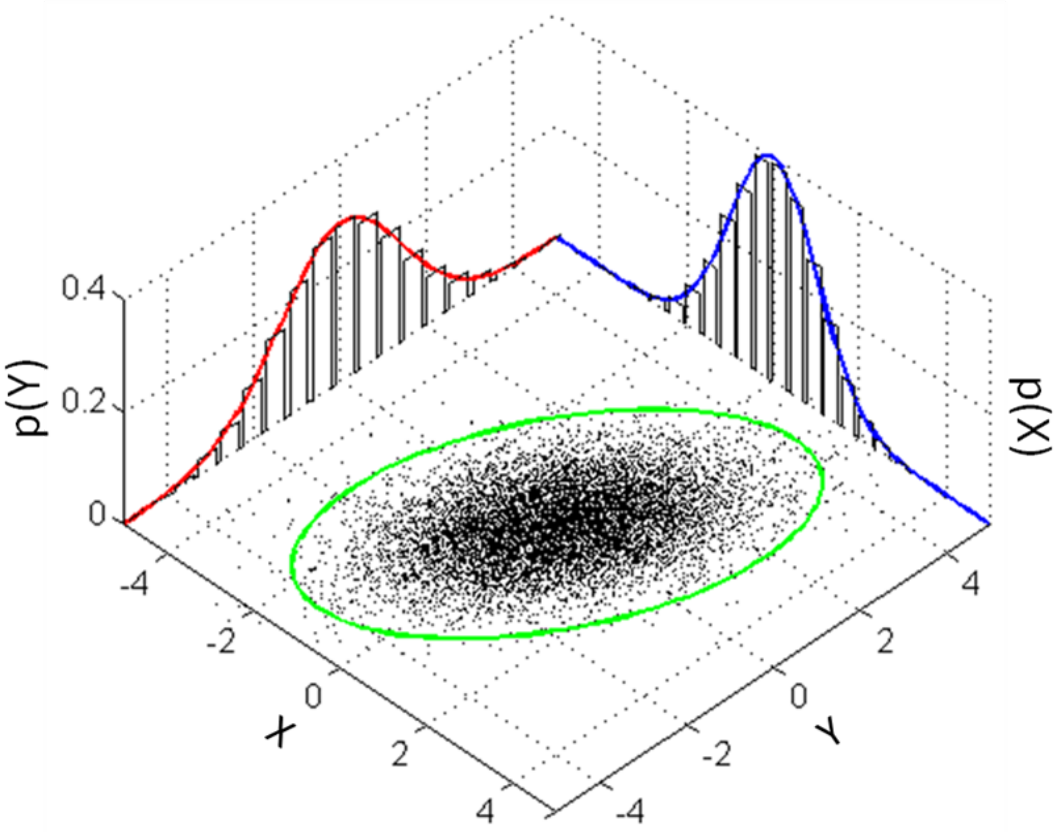
In the binary (two-class) case, we have two Gaussians:

$$\begin{aligned} X \mid y = 1 &\sim N(\mu_1, \Sigma_1) \\ X \mid y = 0 &\sim N(\mu_0, \Sigma_0) \end{aligned}$$

The decision boundary is a quadratic surface (algebra!)

Quadratic discriminant analysis

To estimate this we just separate the training data according to the two labels and estimate two separate Gaussians. Easy-peasy!



Think of *Y* here as another predictor variable, not the class label!

https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Linear discriminant analysis

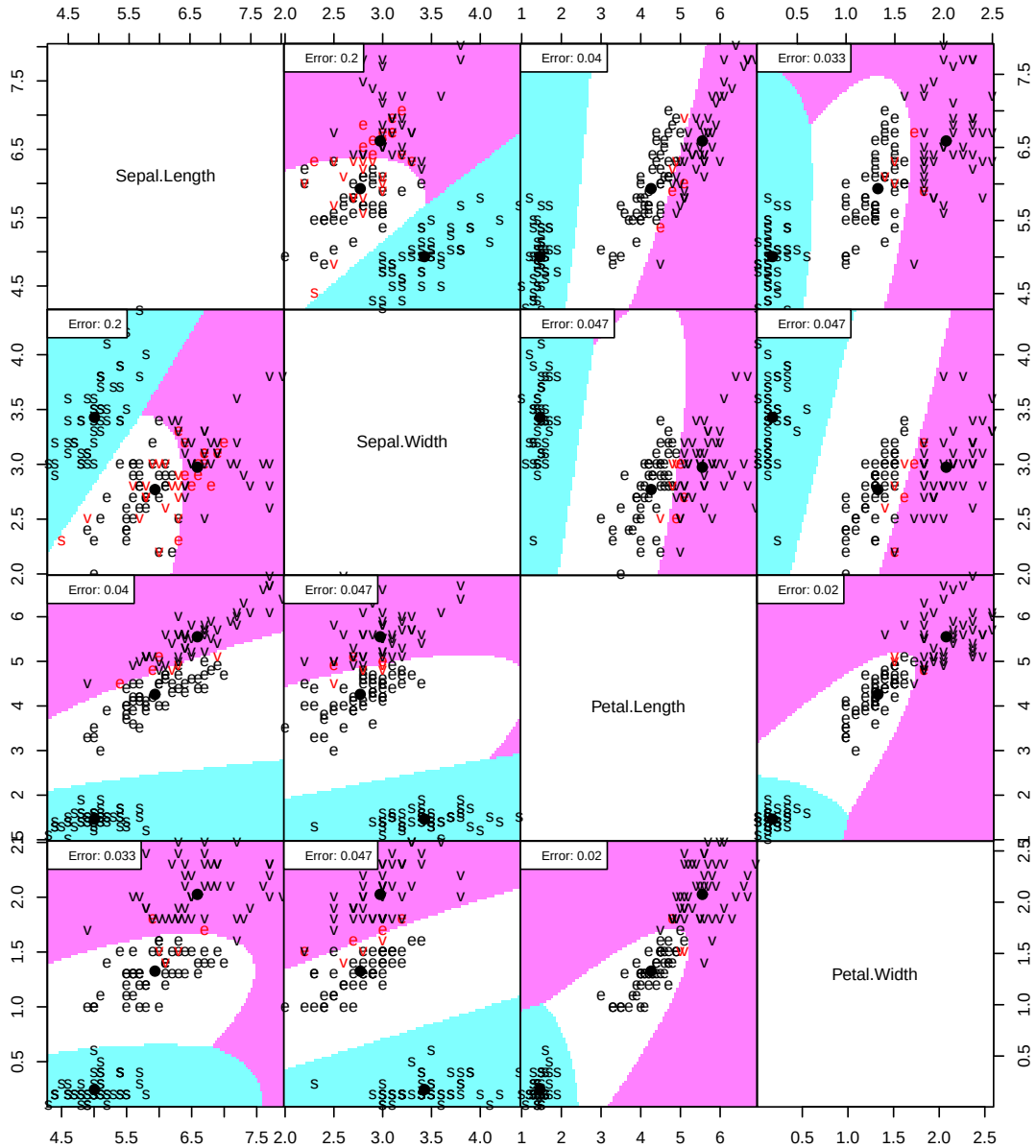
In the binary (two-class) case, we again have two Gaussians:

$$X \mid y = 1 \sim N(\mu_1, \Sigma)$$
$$X \mid y = 0 \sim N(\mu_0, \Sigma)$$

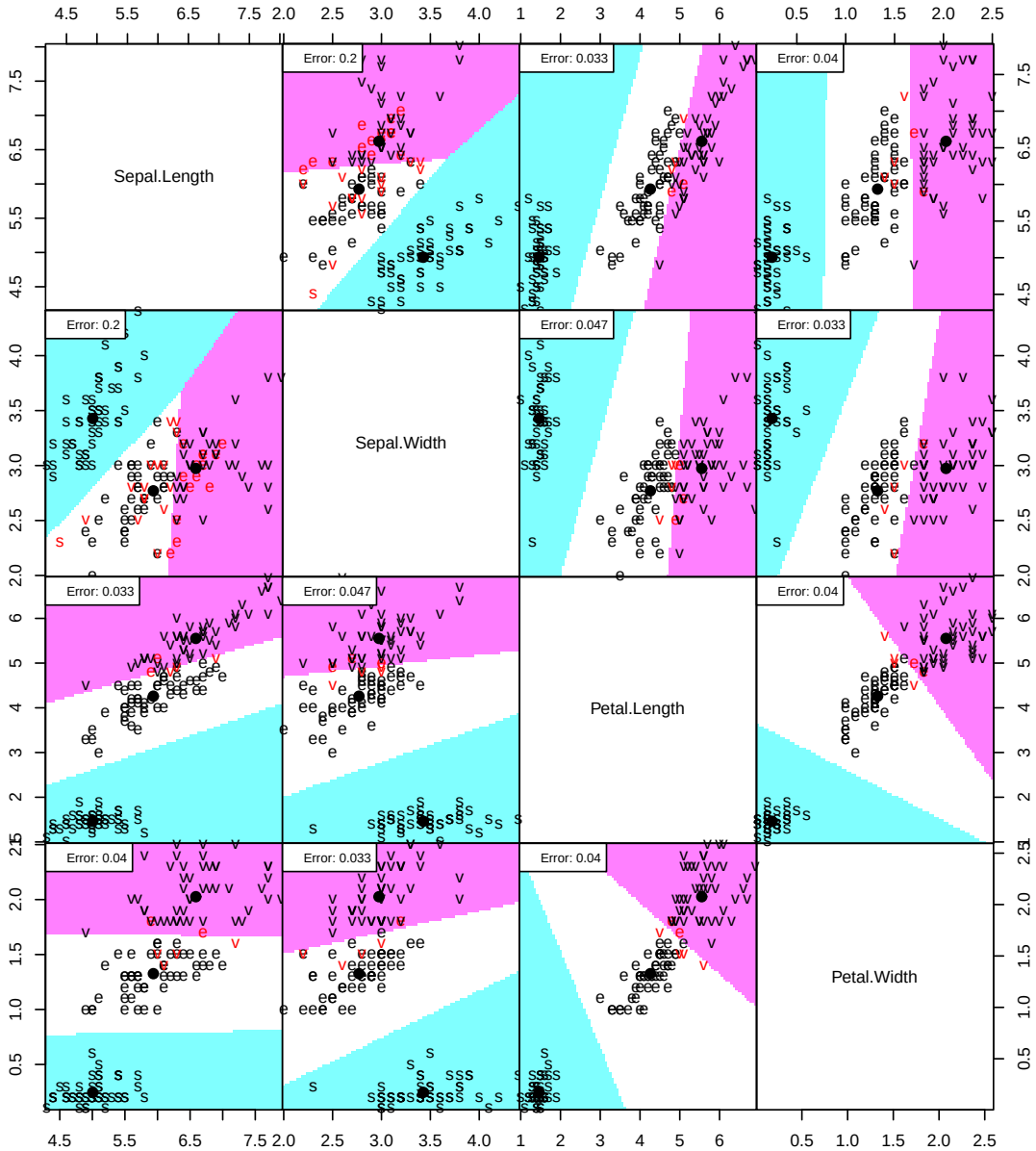
But now we use the *same covariance* matrix for each.

The decision boundary is now *linear*.

Quadratic discriminant analysis: Iris data



Linear discriminant analysis: Iris data

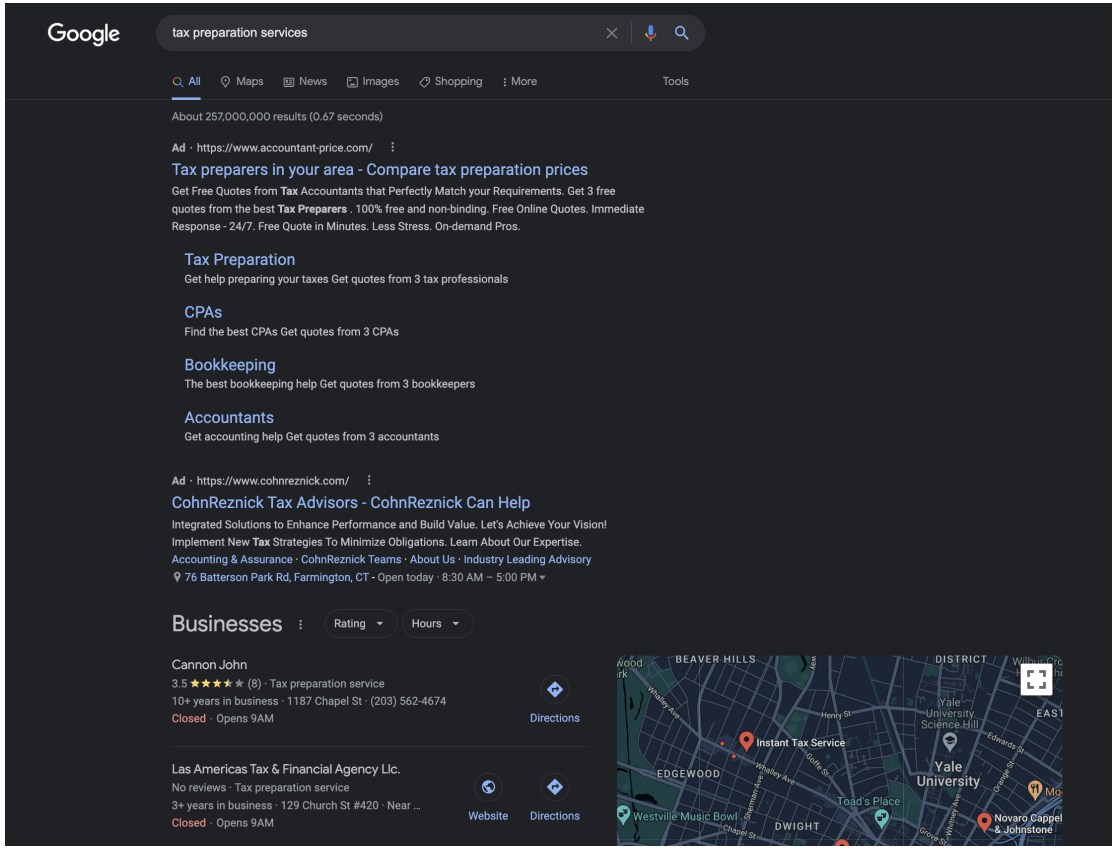


Stochastic gradient descent

- Suppose that we want to fit a really big model, where the number of samples n and number of variables p are very large
- The classical algorithms in standard software packages will fail
- How can we train such models?

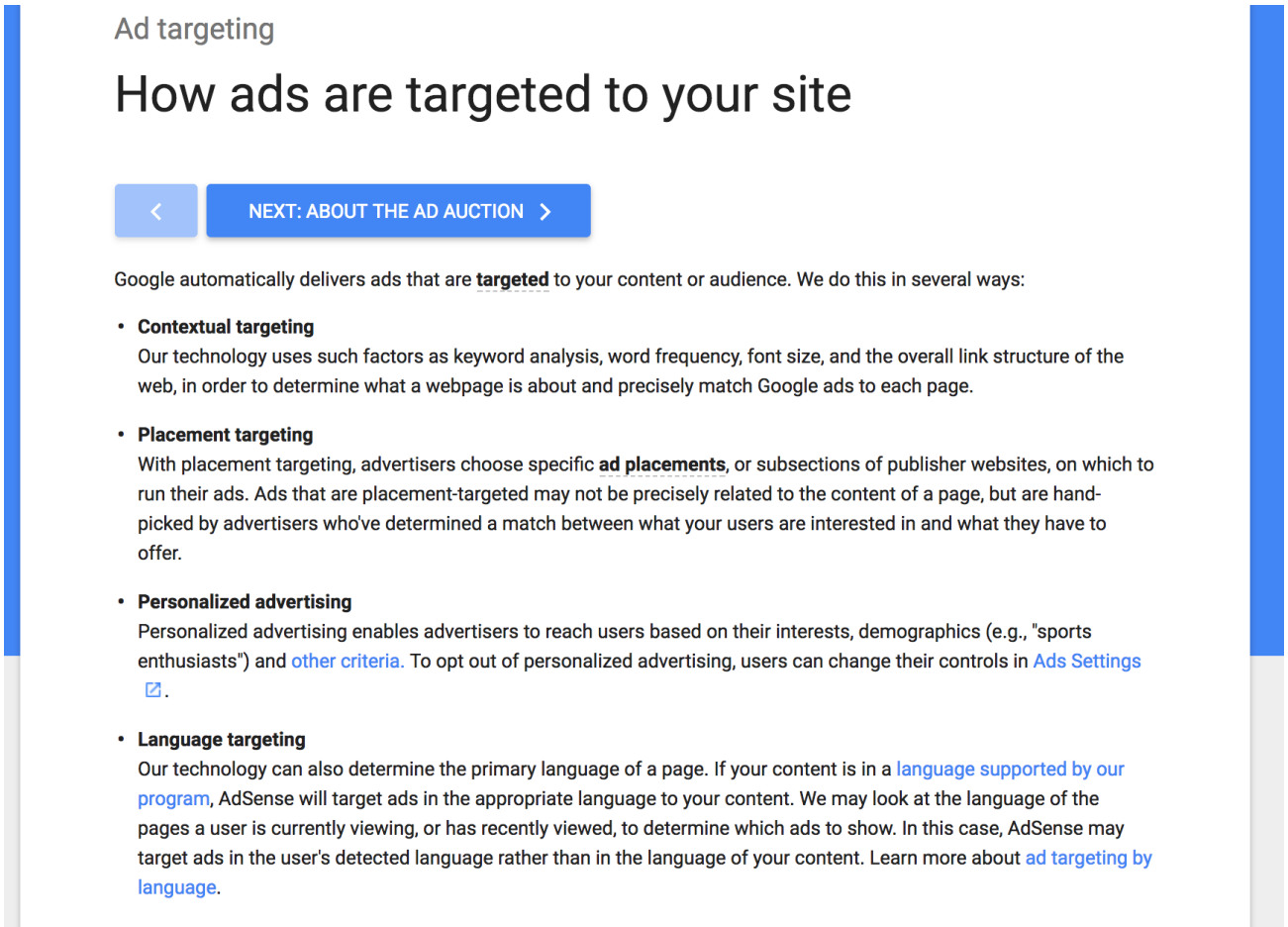
Recall:Classification tasks

- Ad click-through prediction. Predict whether or not a user will click on an ad presented. Used for ranking ads and setting prices.



Recall:Classification tasks

- Ad click-through prediction. Predict whether or not a user will click on an ad presented. Used for ranking ads and setting prices.



Ad targeting

How ads are targeted to your site

NEXT: ABOUT THE AD AUCTION

Google automatically delivers ads that are **targeted** to your content or audience. We do this in several ways:

- **Contextual targeting**
  Our technology uses such factors as keyword analysis, word frequency, font size, and the overall link structure of the web, in order to determine what a webpage is about and precisely match Google ads to each page.
- **Placement targeting**
  With placement targeting, advertisers choose specific **ad placements**, or subsections of publisher websites, on which to run their ads. Ads that are placement-targeted may not be precisely related to the content of a page, but are hand-picked by advertisers who've determined a match between what your users are interested in and what they have to offer.
- **Personalized advertising**
  Personalized advertising enables advertisers to reach users based on their interests, demographics (e.g., "sports enthusiasts") and other criteria. To opt out of personalized advertising, users can change their controls in Ads Settings .
- **Language targeting**
  Our technology can also determine the primary language of a page. If your content is in a language supported by our program, AdSense will target ads in the appropriate language to your content. We may look at the language of the pages a user is currently viewing, or has recently viewed, to determine which ads to show. In this case, AdSense may target ads in the user's detected language rather than in the language of your content. Learn more about ad targeting by language.

Example

- We want to classify ads according to whether or not they will be clicked on by a user
- We have a very large collection of training data
- Ads are represented in terms of a sparse list of features

$$1 \mid 5 : 1.1789641e{-}01 \quad 39 : 6.0373064e{-}02 \quad 45 : 1.3163488e{-}01$$

- The dataset is too large to load into memory, and the number of features is also very large
- New data are continually arriving
- How can we efficiently train a classifier?

Online learning

We will introduce a method that

- Reads in the data points one (or a few) at a time
- Updates the model for each sample
- Exploits sparsity of the features
- Uses little memory, never reads in the entire dataset

Stochastic gradient descent

Initialize all parameters to zero: $\beta_j = 0$, $j = 1, \ldots, p$.

Read through the data one record at a time, and update the model.

1. Read data item x
2. Make a prediction $\widehat{y}(x)$
3. Observe the true response/label y
4. Update the parameters β so $\widehat{y}$ is closer to y

Stochastic gradient descent

To begin, suppose we are doing *linear regression*. We initialize all parameters to zero: $\beta_j = 0$, $j = 1, \ldots, p$.

We read through the data one record at a time, and update the model.

1. Read data item x
2. Make a prediction $\widehat{y}(x) = \sum_{j=1}^{p} \beta_j x_j$
3. Observe the true response/label y
4. Update the parameters β so $\widehat{y}$ is closer to y

SGD idea

Here's the idea:

- For each parameter β_j, see what happens to the loss if that parameter is increased a little bit.
- If the loss goes down (up), then increase (decrease) β_j proportionately
- Do this simultaneously for all of the parameters
- Rinse and repeat

SGD idea

Change β_j by a little bit:

$$\beta_j \to \beta_j + \varepsilon$$

What happens to the squared error?

$$\begin{aligned}
\tfrac{1}{2}(y - \widehat{y})^2 &\to \tfrac{1}{2}(y - \widehat{y} - \varepsilon x_j)^2 \\
&\approx \tfrac{1}{2}(y - \widehat{y})^2 - (y - \widehat{y}) x_j \, \varepsilon \\
&= \tfrac{1}{2}(y - \widehat{y})^2 + \underbrace{(\widehat{y} - y) x_j}_{g} \, \varepsilon
\end{aligned}$$

SGD idea

We then change the parameter as follows:

$$\begin{aligned}\beta_j &\to \beta_j - \eta g \\ &= \beta_j - \eta \underbrace{(\widehat{y} - y)x_j}_{g} \\ &= \beta_j + \eta(y - \widehat{y})x_j\end{aligned}$$

with $\eta > 0$ a small positive "step size".

SGD idea

Why is this a good idea? With this choice of $\varepsilon = -\eta g$ the squared error decreases:

$$\begin{aligned}
\tfrac{1}{2}(y - \widehat{y})^2 &\to \tfrac{1}{2}(y - \widehat{y} - \varepsilon x_j)^2 \\
&\approx \tfrac{1}{2}(y - \widehat{y})^2 + \varepsilon g \\
&= \tfrac{1}{2}(y - \widehat{y})^2 - \eta g^2 \\
&< \tfrac{1}{2}(y - \widehat{y})^2
\end{aligned}$$

so we're moving "downhill"!

SGD for general loss

Suppose $L(y, \beta^T x)$ is the loss for an input (x, y), e.g., $(y - \beta^T x)^2$

SGD update:

$$\beta_j \longleftarrow \beta_j - \eta \frac{\partial L(y, \beta^T x)}{\partial \beta_j}$$

$$\boldsymbol{\beta} \longleftarrow \boldsymbol{\beta} - \eta \nabla_\beta L(y, \boldsymbol{\beta}^T x) \quad \text{(vector notation)}$$

- η is the *learning rate* or "step size"
- Needs to be chosen carefully, getting smaller over time

Gradient descent for general loss

If $L(\boldsymbol{\beta})$ is the loss function over subset of training set:

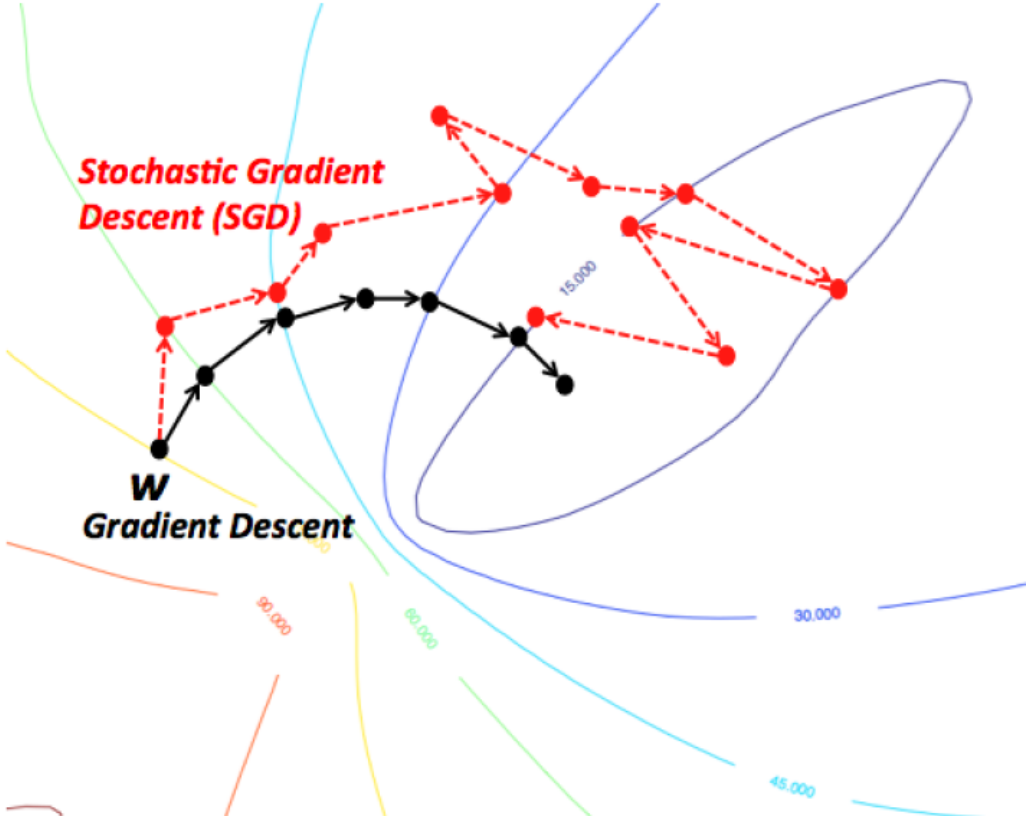
$$L(\boldsymbol{\beta} + \eta \boldsymbol{v}) \approx L(\boldsymbol{\beta}) + \eta \boldsymbol{v}^T \nabla L(\boldsymbol{\beta})$$
$$L(\boldsymbol{\beta} - \eta \nabla L(\boldsymbol{\beta})) \approx L(\boldsymbol{\beta}) - \eta \|\nabla L(\boldsymbol{\beta})\|^2$$

This is why gradient descent is going downhill — if η is small enough.

"Batch" gradient descent uses the entire training set in each step of gradient descent.

Stochastic gradient descent computes a quick approximation to this gradient, using only a single or a small "mini-batch" of data points

Batch vs. stochastic gradient descent



https://wikidocs.net/3413

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small? *small change*
- Suppose $y = 0$ and probability $p(x)$ is big?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small? *small change*
- Suppose $y = 0$ and probability $p(x)$ is big? *big change* $\downarrow$

SGD: choice of step size

In theory, we need to let the step size η decrease as the algorithm progresses.

This prevents the estimates from oscillating back and forth without converging.

Demo

Open the demo notebook `sgd.ipynb` and follow along...

SGD: Scaling

We generally want to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

SGD: Scaling

We generally want to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

Solution: The mean and variance can be updated in an online manner, in constant time, by storing auxiliary variables for each component j.

SGD: Regularization

A "ridge" penalty $\frac{1}{2}\lambda \sum_{j=1}^{p} \beta_j^2$ is easily handled.

Gradient changes by an additive term $\lambda\beta$. Update becomes

$$\begin{aligned} \beta_j &\longleftarrow \beta_j + \eta\{(y - p(x))x_j - \lambda\beta_j\} \\ &= (1 - \eta\lambda)\beta_j + \eta(y - p(x))x_j \end{aligned}$$

Observe that this "does the right thing" whether β_j wants to be large positive or negative.

- *The penalty shrinks β_j toward zero*

What did we learn today?

- Stochastic gradient descent is a simple algorithm that can be applied to large classification and regression problems
- A parameter is updated according to how much the loss changes when that parameter is changed by a little bit
- This is the “go to” algorithm for fitting large or complex machine learning models
- Choosing the learning rate is a little tricky