Probabilistic machine learning

What we've learnt so far ...

Lecture 2

Supervised Learning

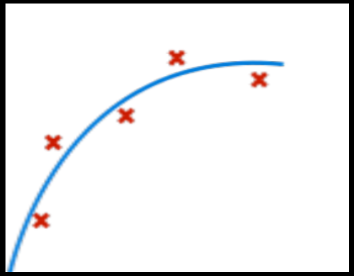
Dataset: $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \ldots, \langle x_M, y_M\rangle\}$

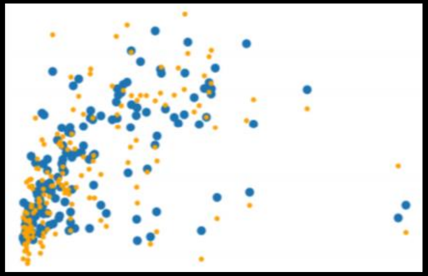
Input instances: $x_1, x_2, x_3, \ldots, x_M$

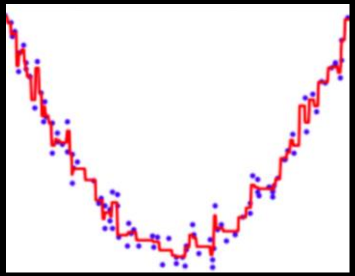
Known (desired) outputs: $y_1, y_2, y_3, \ldots, y_M$

Our goal: Learn the mapping $f: X \rightarrow Y$
such that $y_i = f(x_i)$ for all $i = 1,2,3, \ldots, M$

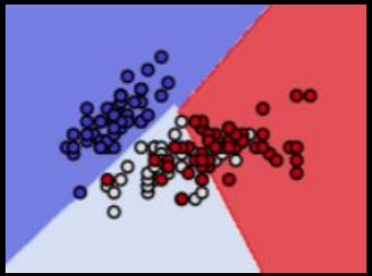
Regression (lecture 2, 4)

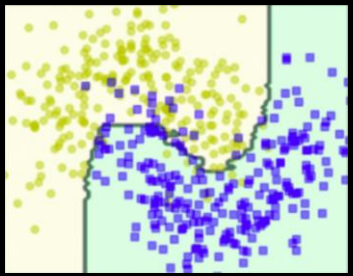


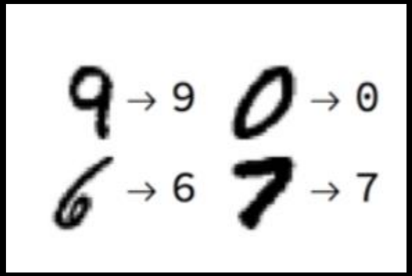




Classification (lecture 3, 4)







Supervised Learning

Dataset: $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \dots, \langle x_M, y_M\rangle\}$

Input instances: $x_1, x_2, x_3, \dots, x_M$

Known (desired) outputs: $y_1, y_2, y_3, \dots, y_M$

Our goal: Learn the mapping $f: X \rightarrow Y$
such that $y_i = f(x_i)$ for all $i = 1,2,3,\dots,M$

Loss function

Our goal: Learn weights θ for a predictor $\hat{y} = f(x;\theta)$ that minimize a loss function. For regression,

$$\text{loss} = \frac{1}{2}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2$$

Unsupervised Learning

Dataset: $\{x_1, x_2, x_3, \dots, x_M\}$

Input instances: $x_1, x_2, x_3, \dots, x_M$

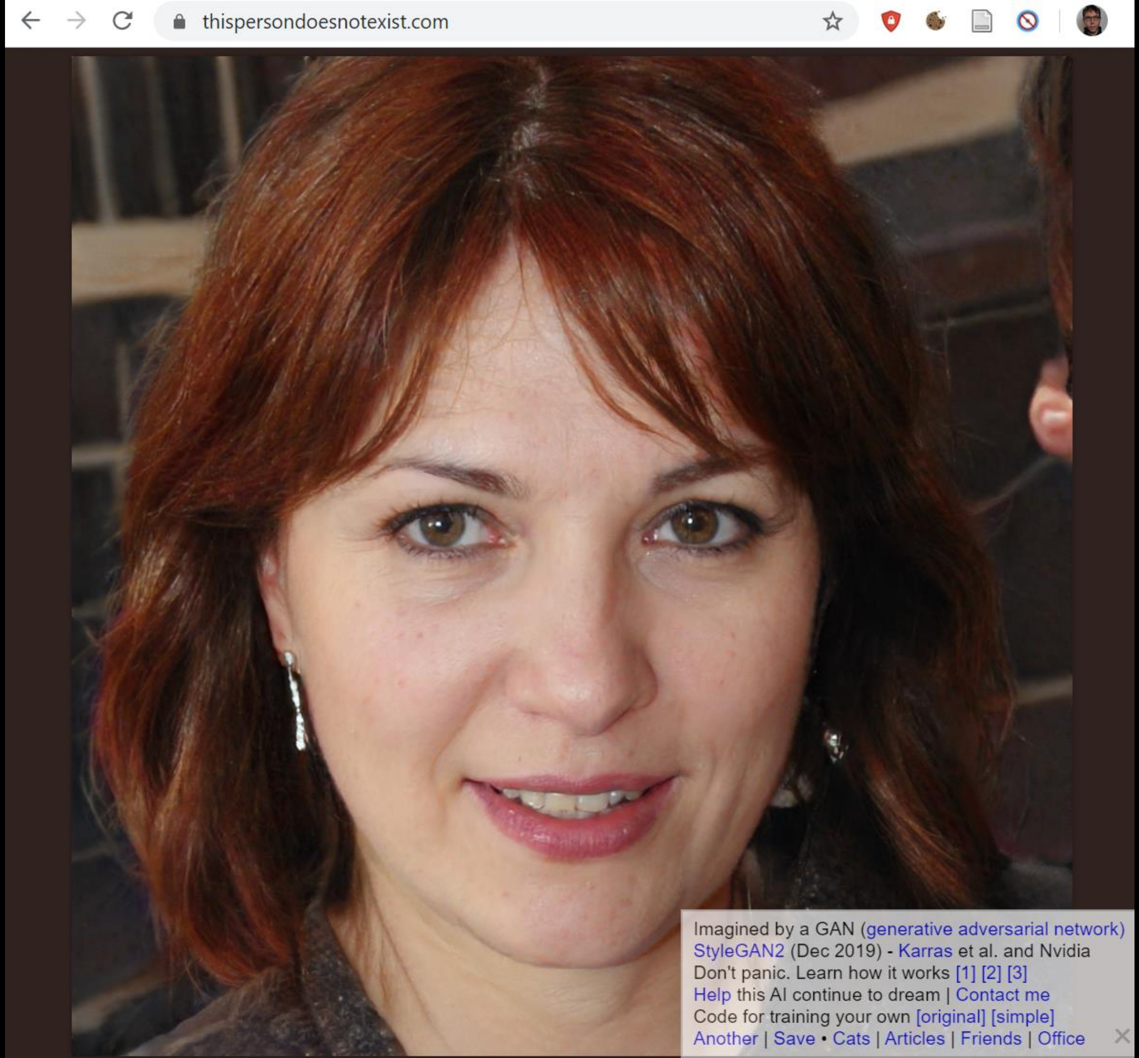
Known (desired) outputs: n/a

Our goal: synthesize new instances
similar to those in the dataset

How can we turn this into a gradient descent problem? What loss function?

Dataset:
a list of craft beer names from untappd.com

Dataset:
Flickr-Faces-HQ dataset,
https://github.com/NVlabs/ffhq-dataset



Andrej Karpathy
@karpathy

Follow

Gradient descent can write code better than you. I'm sorry.

1:56 pm - 4 Aug 2017

348 Retweets **1,174** Likes

70 348 1.2K

Probabilistic machine learning (a better way to think of loss functions)

Supervised Learning

Dataset: $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \ldots, \langle x_M, y_M\rangle\}$

Predictors: $x_1, x_2, x_3, \ldots, x_M$

Probability model: $\Pr_Y(y_i | x_i, \theta)$

Observations: $y_1, y_2, y_3, \ldots, y_M$

Our goal: Learn θ to maximize $\prod_{i=1}^{M} \Pr_Y(y_i | x_i, \theta)$

It's up to us to pick a probability model.

Just as it was up to us to pick a loss function.

Probabilistic machine learning (a better way to think of loss functions)

Supervised Learning

Dataset:	$\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \dots, \langle x_M, y_M\rangle\}$
Predictors:	$x_1, x_2, x_3, \dots, x_M$
Probability model:	$\Pr_Y(y_i \mid x_i, \theta)$
Observations:	$y_1, y_2, y_3, \dots, y_M$
Our goal:	Learn θ to maximize $\prod_{i=1}^{M} \Pr_Y(y_i \mid x_i, \theta)$

$$\Pr_Y(y_i \mid x_i, \theta, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - f_\theta(x_i))^2 / 2\sigma^2}$$

Example: regression

Observations:	$y_i \in \mathbb{R}$
Probability model:	$Y_i \sim N(f_\theta(x_i), \sigma^2)$
Our goal:	Learn θ and/or σ to maximize ...

maximize $\prod_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - f_\theta(x_i))^2 / 2\sigma^2}$

equivalently, maximize $\left\{ -\frac{M}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{M} (y_i - f_\theta(x_i))^2 \right\}$

the standard loss function for regression

Probabilistic machine learning (a better way to think of loss functions)

Supervised Learning

Dataset: $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \ldots, \langle x_M, y_M\rangle\}$

Predictors: $x_1, x_2, x_3, \ldots, x_M$

Probability model: $\Pr_Y(y_i | x_i, \theta)$

Observations: $y_1, y_2, y_3, \ldots, y_M$

Our goal: Learn θ to maximize $\prod_{i=1}^{M} \Pr_Y(y_i | x_i, \theta)$

$$\mathbb{P}(Y_i = 1) = f_\theta(x_i)$$
$$\mathbb{P}(Y_i = 0) = 1 - f_\theta(x_i)$$

needs f to be in the range $[0,1]$

Example: binary classification

Observations: $y_i \in \{0,1\}$

Probability model: $Y_i \sim \mathrm{Bin}(1, f_\theta(x_i))$

Goal: Learn θ to maximize …

$$\text{maximize} \sum_{i=1}^{m} \log \begin{cases} f_\theta(x_i) & \text{if } y_i = 1 \\ 1 - f_\theta(x_i) & \text{if } y_i = 0 \end{cases}$$

$$= \sum_{i=1}^{m} \sum_{k \in \{0,1\}} 1_{y_i = k} \log g_\theta(x_i, k) \quad \text{where} \quad g_\theta(x_i, k) = \begin{cases} f_\theta(x_i) & \text{if } k = 1 \\ 1 - f_\theta(x_i) & \text{if } k = 0 \end{cases}$$

cross-entropy loss function

Training a neural network $\equiv$ maximum likelihood estimation

How to do unsupervised learning with gradient descent

Supervised Learning

Dataset: $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \langle x_3, y_3\rangle, \dots, \langle x_M, y_M\rangle\}$

Predictors: $x_1, x_2, x_3, \dots, x_M$

Probability model: $\Pr_Y(y_i|x_i, \theta)$

Observations: $y_1, y_2, y_3, \dots, y_M$

Our goal: Learn θ to maximize $\prod_{i=1}^{M} \Pr_Y(y_i|x_i, \theta)$

Unsupervised Learning

Dataset: $\{x_1, x_2, x_3, \dots, x_M\}$

Predictors: n/a

Probability model: $\Pr_X(x_i|\theta)$

Observations: $x_1, x_2, x_3, \dots, x_M$

Our goal: Learn θ to maximize $\prod_{i=1}^{M} \Pr_X(x_i|\theta)$

Application: name generation

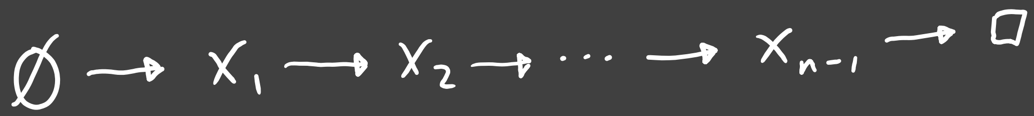
Let the dataset be a collection of names {Øabigail□, Øandrew□, ...}

Let the letters of a name x be $\emptyset x_1 x_2 \cdots x_n$

MARKOV MODEL

Generate each X_j randomly, based on X_{j-1}, and when we hit □ then stop

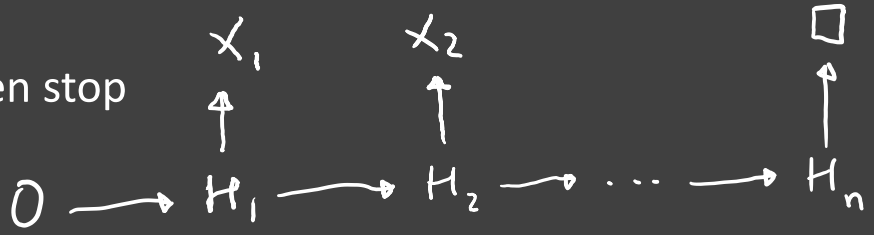
$\Pr(x_1 \cdots x_n) = P_{\emptyset x_1} P_{x_1 x_2} \cdots P_{x_{n-1} x_n}$



HIDDEN MARKOV MODEL

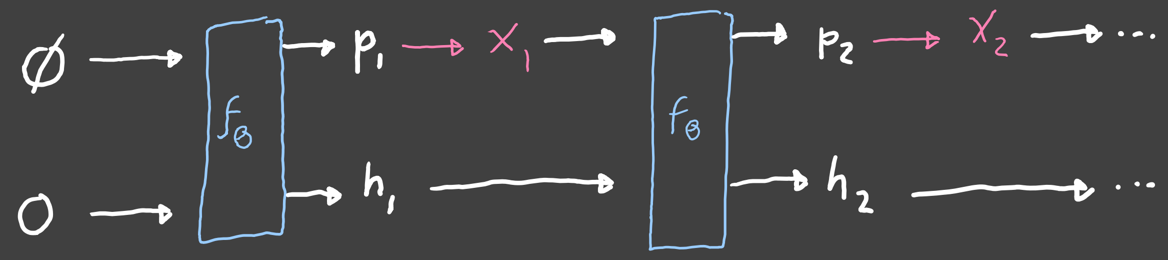
Generate a hidden Markov sequence $0 H_1 H_2 \cdots$

Generate each X_j randomly, based on H_j, and when we hit □ then stop



RECURRENT NEURAL NETWORK

$(X, h) = ([\emptyset], 0)$
while $X.\text{last} \neq \square$:
    $(p, h) = f_\theta(X.\text{last}, h)$
    newchar = random.choice(alphabet, prob=p)
    X.append(newchar)



RNN is richer than HMM, because each X_j depends on the entire history $X_1 X_2 \cdots X_{j-1}$

RNN is simpler than HMM, because there's less randomness.
We can explicitly write out the probability model $\Pr_X(x)$, which we need for training.

$$\Pr(x_1 \cdots x_n) = p_1[x_1] \times p_2[x_2] \times \cdots \times p_n[x_n]$$

Evaluating an unsupervised model

Lecture 2

Dataset splits

Training Set
for training your models, fitting the parameters

Dev Set
for hyper-parameter selection

Test Set
for realistic evaluation

Training goal, summing over the training dataset

$$\max_{\theta} \frac{1}{M} \sum_{i=1}^{M} \log \Pr_X(x_i | \theta)$$

Evaluation metric, summing over the test set

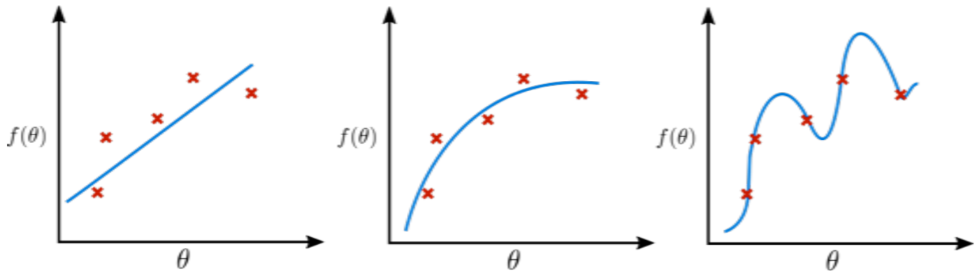
$$\frac{1}{N} \sum_{i=1}^{N} \log \Pr_X(x_i | \hat{\theta})$$

called the *average log likelihood* (linked to *perplexity*)

Evaluating a probabilistic model

Lecture 2

Overfitting

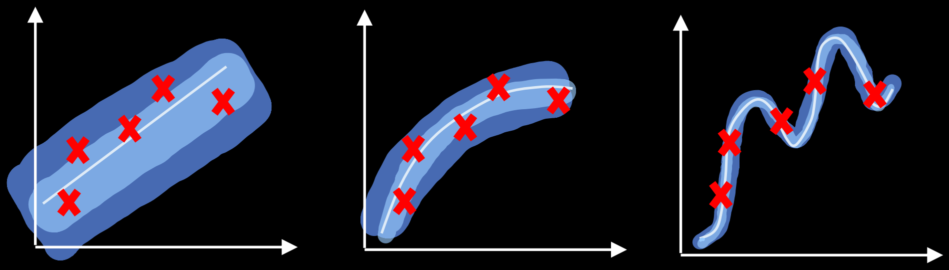


Underfitting

the model is too simple to fit the data well

Overfitting

the model is too complex / has too many parameters



An underfit model thinks the data is mostly noise

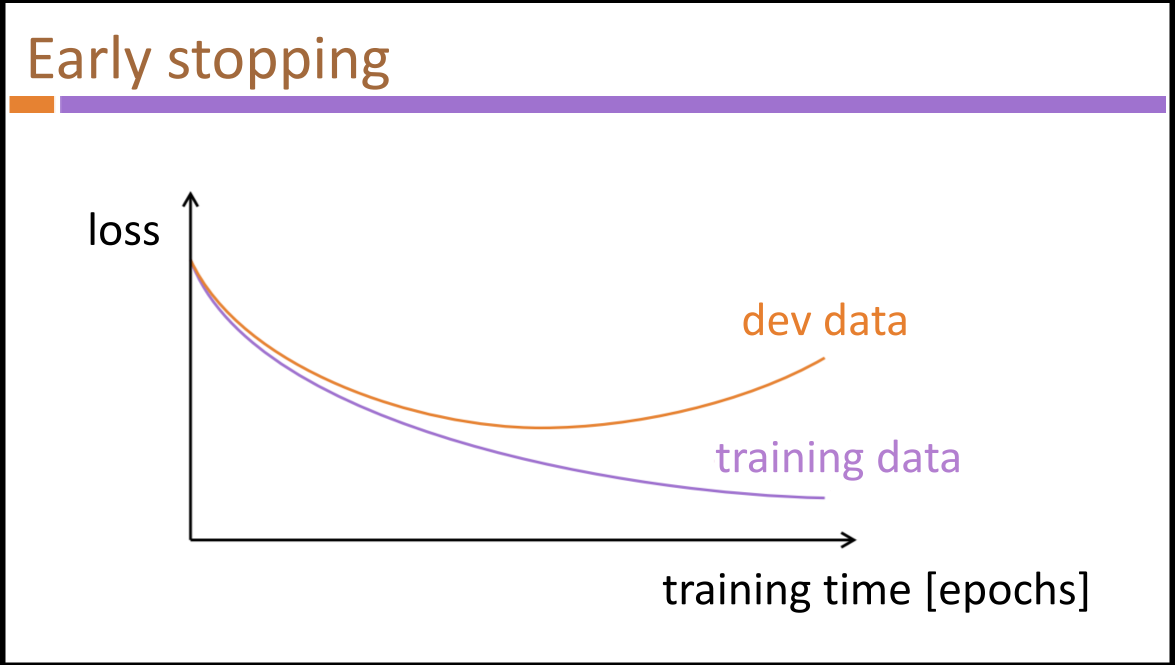
An overfit model thinks every last variation is explicable

Evaluating a probabilistic model

For a probabilistic model, use

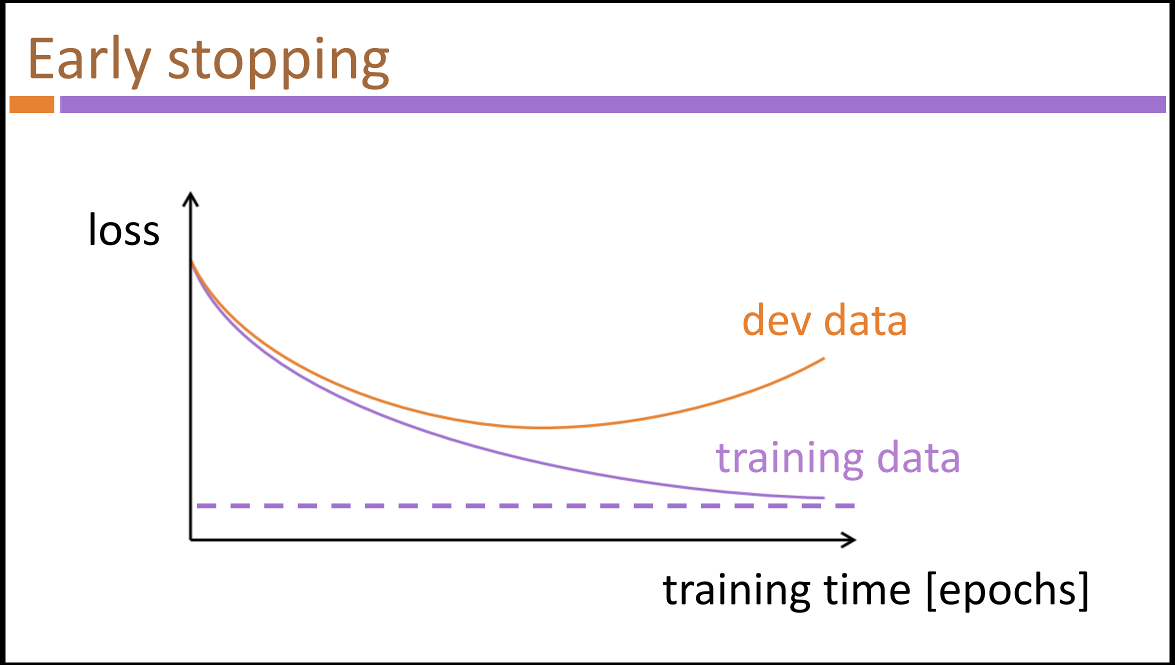
loss = - average log lik(data)

Lecture 7



Evaluating a probabilistic model

Lecture 7



For an unsupervised model, we can calculate the theoretical lower bound on training loss.

If our model doesn't reach this bound, it's underfitted.

negative loss

= Av. log likelihood on training dataset $\{x_1, \dots, x_M\}$

$$= \frac{1}{M} \sum_{i=1}^{M} \log \Pr_X(x_i \mid \theta) \leq \frac{1}{M} \sum_{i=1}^{M} \log \frac{1}{M} = \log M$$

The best-fitting distribution is the empirical distribution, which assigns probability $1/M$ to each datapoint.

- Code for regression
- Code for binary classification
- Code + derivation for multiclass?