Probabilistic machine learning
Lecture 2

Supervised Learning

Dataset: \( \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_M, y_M)\} \)

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 \( i = 1, 2, 3, \ldots, M \)
Supervised Learning

Dataset: \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_M, y_M)\}

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 \( f : X \to Y \) such that \( y_i = f(x_i) \) for all \( i = 1, 2, 3, \ldots, M \)

Loss function

Our goal: Learn weights \( \theta \) 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, \ldots, x_M\}

Input instances: \{x_1, x_2, x_3, \ldots, 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
Gradient descent can write code better than you. I'm sorry.
Probabilistic machine learning (a better way to think of loss functions)

<table>
<thead>
<tr>
<th>Supervised Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataset:</strong></td>
</tr>
<tr>
<td><strong>Predictors:</strong></td>
</tr>
<tr>
<td><strong>Probability model:</strong></td>
</tr>
<tr>
<td><strong>Observations:</strong></td>
</tr>
<tr>
<td><strong>Our goal:</strong></td>
</tr>
</tbody>
</table>

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:**\{\(\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \langle x_3, y_3 \rangle, \ldots, \langle x_M, y_M \rangle\}\)

**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 \(\theta\) to maximize \(\prod_{i=1}^{M} \Pr(Y | x_i, \theta)\)

**Example: regression**

Observations: \(y_i \in \mathbb{R}\)

Probability model: \(Y_i \sim N(f_\theta(x_i), \sigma^2)\)

Our goal: Learn \(\theta\) and/or \(\sigma\) to maximize ... 

\[
\Pr_Y(y_i | x_i, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}}
\]

equivalently, maximize

\[
\max_{\theta} \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}}
\]

the standard loss function for regression

\[
-\frac{M}{2} \log (2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{M} (y_i - f_\theta(x_i))^2
\]
Probabilistic machine learning (a better way to think of loss functions)

Supervised Learning

Dataset: \( \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_M, y_M)\} \)

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 \( \theta \) to maximize \( \prod_{i=1}^{M} \Pr_Y(y_i | x_i, \theta) \)

Example: binary classification

Observations: \( y_i \in \{0,1\} \)

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

Goal: Learn \( \theta \) to maximize ...

\[
\begin{align*}
\maximize \quad & \sum_{i=1}^{M} \log \begin{cases} f_\theta(x_i) & \text{if } y_i = 1 \\ 1 - f_\theta(x_i) & \text{if } y_i = 0 \end{cases} \\
= & \sum_{i=1}^{M} \sum_{k \in \{0,1\}} \mathbb{1}_{y_i = k} \log g_\theta(x_i, k)
\end{align*}
\]

where \( g_\theta(x_i, k) = \begin{cases} f_\theta(x_i) & \text{if } k = 1 \\ 1 - f_\theta(x_i) & \text{if } k = 0 \end{cases} \)

cross-entropy loss function
Training a neural network $\equiv$ maximum likelihood estimation
### Supervised Learning

<table>
<thead>
<tr>
<th>Dataset</th>
<th>{ (x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_M, y_M) }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictors</td>
<td>x_1, x_2, x_3, \ldots, x_M</td>
</tr>
<tr>
<td>Probability model</td>
<td>Pr_Y(y_i</td>
</tr>
<tr>
<td>Observations</td>
<td>y_1, y_2, y_3, \ldots, y_M</td>
</tr>
<tr>
<td>Our goal</td>
<td>Learn \theta to maximize \prod_{i=1}^{M} Pr_Y(y_i</td>
</tr>
</tbody>
</table>

### Unsupervised Learning

<table>
<thead>
<tr>
<th>Dataset</th>
<th>{ x_1, x_2, x_3, \ldots, x_M }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictors</td>
<td>n/a</td>
</tr>
<tr>
<td>Probability model</td>
<td>Pr_X(x_i</td>
</tr>
<tr>
<td>Observations</td>
<td>x_1, x_2, x_3, \ldots, x_M</td>
</tr>
<tr>
<td>Our goal</td>
<td>Learn \theta to maximize \prod_{i=1}^{M} Pr_X(x_i</td>
</tr>
</tbody>
</table>
Application: name generation

Let the dataset be a collection of names \{\emptyset abigail\,\Box,\, \emptyset andrew\,\Box,\, ...\}

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 \(\Box\) 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 \(0H_1 H_2 \cdots\)

Generate each \(X_j\) randomly, based on \(H_j\), and when we hit \(\Box\) 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)
    \(X, \text{append}(\text{newchar})\)

RNN is richer than HMM, because each \(X_j\) depends on the entire history \(X_1X_2 \cdots X_{j-1}\).

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

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Dev Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>for training your models, fitting the parameters</td>
<td>for hyper-parameter selection</td>
<td>for realistic evaluation</td>
</tr>
</tbody>
</table>

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

\[ \text{loss} = - \text{average log lik(data)} \]
Evaluating a probabilistic model

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.

\[
\text{negative loss} = \text{av. log likelihood on training dataset } \{x_1, \ldots, x_M\}
\]

\[
= \frac{1}{M} \sum_{i=1}^{M} \log \Pr_x(x_i | \theta) \leq \frac{1}{M} \sum_{i=1}^{M} \log \left( \frac{1}{M} \right) = \log M
\]

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?