Perceptron
<table>
<thead>
<tr>
<th>Q:</th>
<th>We pick the best hyperparameters by learning on the training data and evaluating error on the validation error. For our final model, should we then learn from training + validation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>Yes.</td>
</tr>
<tr>
<td></td>
<td>Let's assume that {train-original} is the original training data, and {test} is the provided test dataset.</td>
</tr>
<tr>
<td></td>
<td>1. Split {train-original} into {train-subset} and {validation}.</td>
</tr>
<tr>
<td></td>
<td>2. Pick the hyperparameters that when training on {train-subset} give the lowest error on {validation}. Call these hyperparameters {best-hyper}.</td>
</tr>
<tr>
<td></td>
<td>3. Retrain a new model using {best-hyper} on {train-original} = {train-subset} \cup {validation}.</td>
</tr>
<tr>
<td></td>
<td>4. Report test error by evaluating on {test}.</td>
</tr>
<tr>
<td></td>
<td>Alternatively, you could replace Step 1/2 with the following:</td>
</tr>
<tr>
<td></td>
<td>Pick the hyperparameters that give the lowest cross-validation error on {train-original}. Call these hyperparameters {best-hyper}.</td>
</tr>
</tbody>
</table>
Reminders

• Homework 2: Decision Trees
  – Out: Wed, Jan 24
  – Due: Mon, Feb 5 at 11:59pm
THE PERCEPTRON ALGORITHM
Perceptron: History

Imagine you are trying to build a new machine learning technique... your name is Frank Rosenblatt... and the year is 1957
Perceptron: History

Imagine you are trying to build a new machine learning technique... your name is Frank Rosenblatt... and the year is 1957

_The New Yorker, December 6, 1958 P. 44_

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization of its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference betw. a cat and a dog, although it wouldn't be able to tell whether the dog was to the left or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.
Linear Models for Classification

Key idea: Try to learn this hyperplane directly

Looking ahead:
• We’ll see a number of commonly used Linear Classifiers
  • These include:
    – Perceptron
    – Logistic Regression
    – Naïve Bayes (under certain conditions)
    – Support Vector Machines

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]
In-Class Exercise

Draw a picture of the region corresponding to:

\[ w_1 x_1 + w_2 x_2 + b > 0 \]

where \( w_1 = 2, w_2 = 3, b = 6 \)

Draw the vector \( w = [w_1, w_2] \)
Visualizing Dot-Products

Chalkboard:

- vector in 2D
- line in 2D
- adding a bias term
- definition of orthogonality
- vector projection
- hyperplane definition
- half-space definitions
Linear Models for Classification

Key idea: Try to learn this hyperplane directly

Looking ahead:
- We’ll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]
Online vs. Batch Learning

**Batch Learning**
Learn from all the examples at once

**Online Learning**
Gradually learn as each example is received
Online Learning

Examples

1. **Stock market** prediction (what will the value of Alphabet Inc. be tomorrow?)

2. **Email** classification (distribution of both spam and regular mail changes over time, but the target function stays fixed - last year's spam still looks like spam)

3. **Recommendation** systems. Examples: recommending movies; predicting whether a user will be interested in a new news article

4. **Ad placement** in a new market
Online Learning

For \( i = 1, 2, 3, \ldots \) :

- **Receive** an unlabeled instance \( \mathbf{x}^{(i)} \)
- **Predict** \( y' = h_\theta(\mathbf{x}^{(i)}) \)
- **Receive** true label \( y^{(i)} \)
- **Suffer loss** if a mistake was made, \( y' \neq y^{(i)} \)
- **Update** parameters \( \Theta \)

Goal:

- **Minimize** the number of **mistakes**
Perceptron

Chalkboard:

– (Online) Perceptron Algorithm
Perceptron Algorithm: Example

Example: $(-1,2) - \times$
- $(1,0) + \checkmark$
- $(1,1) + \times$
- $(-1,0) - \checkmark$
- $(-1,-2) - \times$
- $(1,-1) + \checkmark$

Perceptron Algorithm: (without the bias term)
- Set $t=1$, start with all-zeros weight vector $w_1$.
- Given example $x$, predict positive iff $w_t \cdot x \geq 0$.
- On a mistake, update as follows:
  - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

Slide adapted from Nina Balcan
Perceptron

**Chalkboard:**
- Why do we need a bias term?
- (Batch) Perceptron Algorithm
- Inductive Bias of Perceptron
- Limitations of Linear Models
Background: Hyperplanes

Hyperplane (Definition 1):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = b \} \]

Hyperplane (Definition 2):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} = 0 \]
and \( x_0 = 1 \} \]
\[ \mathbf{\theta} = [b, w_1, \ldots, w_M]^T \]

Half-spaces:
\[ \mathcal{H}^+ = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} > 0 \text{ and } x_0 = 1 \} \]
\[ \mathcal{H}^- = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} < 0 \text{ and } x_0 = 1 \} \]

Notation Trick: fold the bias \( b \) and the weights \( \mathbf{w} \) into a single vector \( \mathbf{\theta} \) by prepending a constant to \( \mathbf{x} \) and increasing dimensionality by one!
(Online) Perceptron Algorithm

Data: Inputs are continuous vectors of length \( M \). Outputs are discrete.

\[^{(1)}, \ y^{(1)} \), \ (x^{(2)}, y^{(2)}) \ldots\]

\[\text{where } x \in \mathbb{R}^M \text{ and } y \in \{+1, -1\}\]

Prediction: Output determined by hyperplane.

\[
\hat{y} = h_\theta(x) = \text{sign}(\theta^T x)
\]

\[
\text{sign}(a) = \begin{cases} 
1, & \text{if } a \geq 0 \\
-1, & \text{otherwise}
\end{cases}
\]

Assume \( \theta = [b, w_1, \ldots, w_M]^T \) and \( x_0 = 1 \)

Learning: Iterative procedure:

• initialize parameters to vector of all zeroes

• while not converged

• receive next example \((x^{(i)}, y^{(i)})\)

• predict \(y' = h(x^{(i)})\)

• if positive mistake: add \( x^{(i)} \) to parameters

• if negative mistake: subtract \( x^{(i)} \) from parameters
(Online) Perceptron Algorithm

**Data:** Inputs are continuous vectors of length $M$. Outputs are discrete.

$$\begin{align*}
(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots \\
\text{where } x \in \mathbb{R}^M \text{ and } y \in \{+1, -1\}
\end{align*}$$

**Prediction:** Output determined by hyperplane.

$$\hat{y} = h_\theta(x) = \text{sign}(\theta^T x)$$

Assume $\theta = [b, w_1, \ldots, w_M]$

**Implementation Trick:** same behavior as our “add on positive mistake and subtract on negative mistake” version, because $y^{(i)}$ takes care of the sign

---

**Algorithm 1** Perceptron Learning Algorithm

```
1: procedure PERCEPTRON(D = {(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots})
2: \quad \theta \leftarrow 0
3: \quad for \ i \in \{1, 2, \ldots\} \ do
4: \quad \quad \hat{y} \leftarrow \text{sign}(\theta^T x^{(i)})
5: \quad \quad if \ \hat{y} \neq y^{(i)} \ then
6: \quad \quad \quad \theta \leftarrow \theta + y^{(i)} x^{(i)}
7: \quad return \theta
```

- Initialize parameters
- For each example
  - Predict
  - If mistake
    - Update parameters
(Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the “batch” setting in contrast to the “online” setting that we’ve discussed so far.

**Algorithm 1** Perceptron Learning Algorithm (Batch)

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$)
2: $\theta \leftarrow 0$ ▷ Update parameters
3: while not converged do ▷ For each example
4: for $i \in \{1, 2, \ldots, N\}$ do ▷ Predict
5: $\hat{y} \leftarrow \text{sign}(\theta^T \mathbf{x}^{(i)})$
6: if $\hat{y} \neq y^{(i)}$ then ▷ If mistake
7: $\theta \leftarrow \theta + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
8: return $\theta$
Learning for Perceptron also works if we have a fixed training dataset, D. We call this the “batch” setting in contrast to the “online” setting that we’ve discussed so far.

**Discussion:**
The Batch Perceptron Algorithm can be derived in two ways.
1. By extending the online Perceptron algorithm to the batch setting (as mentioned above)
2. By applying **Stochastic Gradient Descent (SGD)** to minimize a so-called **Hinge Loss** on a linear separator
Extensions of Perceptron

• **Voted Perceptron**
  – generalizes better than (standard) perceptron
  – memory intensive (keeps around every weight vector seen during training, so each one can vote)

• **Averaged Perceptron**
  – empirically similar performance to voted perceptron
  – can be implemented in a memory efficient way (running averages are efficient)

• **Kernel Perceptron**
  – Choose a kernel $K(x', x)$
  – Apply the **kernel trick** to Perceptron
  – Resulting algorithm is **still very simple**

• **Structured Perceptron**
  – Basic idea can also be applied when $y$ ranges over an exponentially large set
  – Mistake bound **does not** depend on the size of that set
ANALYSIS OF PERCEPTRON
Analysis: Perceptron

Perceptron Mistake Bound

**Guarantee:** If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn’t change the number of mistakes; algo is invariant to scaling.)