Classification with Perceptrons

Reading:


We will cover material in Chapters 1-2 only, but you’re encouraged to read Chapter 3 (and the rest of the book!) as well.
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots, x_n)\)

Weights are \((w_1, w_2, \ldots, w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots x_n)\)

Weights are \((w_1, w_2, \ldots w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

\[
\text{If } w_1x_1 + w_2x_2 + \ldots + w_nx_n > \text{threshold} \\
\text{then } y = 1, \text{ else } y = 0
\]
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots x_n)\)

Weights are \((w_1, w_2, \ldots w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

\[
\text{If } w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \text{threshold} \\
\text{then } y = 1, \text{ else } y = 0
\]
Perceptrons as simplified "neurons"

$w_0$ is called the "bias"

$-w_0$ is called the "threshold"

Input is $(x_1, x_2, \ldots x_n)$

Weights are $(w_1, w_2, \ldots w_n)$

Output $y$ is 1 ("the neuron fires") if the sum of the inputs times the weights is greater or equal to the threshold:

$$\text{If } w_1x_1 + w_2x_2 + \ldots + w_nx_n > \text{threshold}$$

$$\text{then } y = 1, \text{ else } y = 0$$
Perceptrons as simplified “neurons”

$w_0$ is called the “bias”

$-w_0$ is called the “threshold”

Input is $(x_1, x_2, ... x_n)$

Weights are $(w_1, w_2, ... w_n)$

Output $y$ is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

If $w_1x_1 + w_2x_2 + ... + w_nx_n > \text{threshold}$
then $y = 1$  else $y = 0$

If $w_1x_1 + w_2x_2 + ... + w_nx_n > -w_0$
then $y = 1$  else $y = 0$
Perceptrons as simplified “neurons”

$w_0$ is called the “bias”

$-w_0$ is called the “threshold”

Input is $(x_1, x_2, \ldots x_n)$

Weights are $(w_1, w_2, \ldots w_n)$

Output $y$ is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

If $w_1x_1 + w_2x_2 + \ldots + w_nx_n > \text{threshold}$
then $y = 1$, else $y = 0$

If $w_1x_1 + w_2x_2 + \ldots + w_nx_n > -w_0$
then $y = 1$, else $y = 0$

If $w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0$
then $y = 1$, else $y = 0$
Perceptrons as simplified “neurons”

$w_0$ is called the “bias”

$-w_0$ is called the “threshold”

Input is $(x_1, x_2, \ldots x_n)$

Weights are $(w_1, w_2, \ldots w_n)$

Output $y$ is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

If $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \text{threshold}$
then $y = 1$, else $y = 0$

If $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > -w_0$
then $y = 1$, else $y = 0$

If $w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$
then $y = 1$, else $y = 0$
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots, x_n)\)

Weights are \((w_1, w_2, \ldots, w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

- If \(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \text{threshold}\) then \(y = 1\), else \(y = 0\)
- If \(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > -w_0\) then \(y = 1\), else \(y = 0\)
- If \(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0\) then \(y = 1\), else \(y = 0\)
- If \(w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0\) then \(y = 1\), else \(y = 0\)
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots x_n)\)

Weights are \((w_1, w_2, \ldots w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

\[
\text{If } w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \text{threshold} \quad \text{then } y = 1, \text{ else } y = 0
\]

\[
\text{If } w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > -w_0 \quad \text{then } y = 1, \text{ else } y = 0
\]

\[
\text{If } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0 \quad \text{then } y = 1, \text{ else } y = 0
\]

\[Let \ a(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}\]

\[Then \ y = a(w \cdot x)\]
Perceptrons as simplified “neurons”

Input is \((x_1, x_2, \ldots x_n)\)

Weights are \((w_1, w_2, \ldots w_n)\)

Output \(y\) is 1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

\[
If \ w_1x_1 + w_2x_2 + \ldots + w_nx_n > \text{threshold} \\
\text{then } y = 1, \text{ else } y = 0
\]

\[
If \ w_1x_1 + w_2x_2 + \ldots + w_nx_n > -w_0 \\
\text{then } y = 1, \text{ else } y = 0
\]

\[
If \ w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0 \\
\text{then } y = 1, \text{ else } y = 0
\]

\[
Let \ a(z) = \begin{cases} 
1 & \text{if } z > 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[
Then \ y = a(w \cdot x)
\]

\(a\) is called an “activation function”
Decision Surfaces

• Assume data can be separated into two classes, positive and negative, by a linear decision surface.

• Assuming data is $n$-dimensional, a perception represents a $(n-1)$-dimensional hyperplane that separates the data into two classes, positive (1) and negative (0).
Example where line won’t work?
Example

- What is the output $y$?

\[
\begin{align*}
1 + & 1 \\
-1 + & -1 \\
.4 + & -.1 \\
\end{align*}
\]
Geometry of the perceptron

In 2D:

\[ w_1 x_1 + w_2 x_2 + w_0 = 0 \]

\[ x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \]
In-Class Exercise 1
Perceptron Learning

\[ w_0 + x_0 \]

\[ w_1 \]

\[ w_2 \]

\[ \ldots \]

\[ w_n \]

\[ x_1 \]

\[ x_2 \]

\[ x_n \]

\[ y \]

input

output
Perceptron Learning

Input instance: $x^k = (x_1, x_2, ..., x_n)$,
with target class $t^k \in \{0,1\}$
Perceptron Learning

Goal is to use the training data to learn a set of weights that will:

(1) correctly classify the training data

(2) generalize to unseen data

Input instance: $x^k = (x_1, x_2, \ldots, x_n)$, with target class $t^k \in \{0,1\}$
Perceptron Learning

Learning is often framed as an optimization problem:

• Find $w$ that minimizes average “loss”:

$$J(w) = \frac{1}{M} \sum_{k=1}^{M} L(w, x^k, t^k)$$

where $M$ is number of training examples and $L$ is a “loss” function.

One part of the “art” of ML is to define a good loss function.

• Here, define the loss function as follows:

Let $y = a(w \cdot x)$

$$L(w, x^k, t^k) = \frac{1}{2}(t^k - y)^2 \quad \text{"squared loss"}$$
Example

Training set:
((0, 0), 1)
(1, 1), 0)
(1, 0), 0)

What is the average loss for this training set, using the squared loss function?
Perceptron Learning

\[ J(w) = \frac{1}{M} \sum_{k=1}^{M} L(w, x^k, t^k) \]

Let \( y = a(w \cdot x) \)

\[ L(w, x^k, t^k) = \frac{1}{2} (t^k - y)^2 \] "squared loss"

Goal of learning is to minimize \( J(w) \).

How to solve this minimization problem?

**Gradient descent.**
“Batch” gradient descent

- One *epoch* = one iteration through the training data.

- After each epoch, compute average loss over the training set:

\[
J(w) = \frac{1}{M} \sum_{k=1}^{M} L(w, x^k, t^k) = \frac{1}{M} \sum_{k=1}^{M} \frac{1}{2} (t^k - y)^2
\]

- Change the weights to move in direction of steepest descent in the average-loss surface:

From T. M. Mitchell, *Machine Learning*
Gradient descent

• To find direction of steepest descent, take the derivative of $J(w)$ with respect to $w$.

• A vector derivative is called a “gradient”: $\nabla J(w)$

$$
\nabla J(w) = \left[ \frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, ..., \frac{\partial J}{\partial w_n} \right]
$$
• Here is how we change each weight:

For $i = 0$ to $n$:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial J}{\partial w_i}$$

and $\eta$ is the *learning rate* (i.e., size of step to take downhill).
• Problem with *true gradient descent*:

  Training process is slow.

  Training process can land in local optimum.

• Common approach to this: use *stochastic gradient descent*:

  – Instead of doing weight update after all training examples have been processed, do weight update after each training example has been processed (i.e., perceptron output has been calculated).

  – Stochastic gradient descent approximates true gradient descent increasingly well as $\eta \to 1/\infty$. 
Derivation of perceptron learning rule (stochastic gradient descent)

We defined \( J = \frac{1}{2} (t - y)^2 \)

Here, use

\[
J = \frac{1}{2} \left( t - (w \cdot x) \right)^2
\]

\[
= \frac{1}{2} \left( t - (w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \right)^2
\]

Then,

\[
\frac{\partial J}{\partial w_i} = -\left( t - (w \cdot x) \right) x_i^k
\]

But we'll use

\[
\frac{\partial J}{\partial w_i} = -(t - y) x_i^k
\]

\[
\Delta w_i = -\eta \frac{\partial J}{\partial w_i} = \eta (t^k - y^k) x_i^k
\]

This is called the “perceptron learning rule”
Perceptron Learning Algorithm

Start with small random weights, \( \mathbf{w} = (w_0, w_1, w_2, \ldots, w_n) \), e.g., \( w_i \in [-0.05, 0.05] \).

Repeat until accuracy on the training data stops increasing or for a maximum number of *epochs* (iterations through training set):

For \( k = 1 \) to \( M \) (total number in training set):

1. Select next training example \((x^k, t^k)\).
2. Run the perceptron with input \( x^k \) and weights \( \mathbf{w} \) to obtain \( y \).
3. If \( y \neq t^k \), update weights:
   for \( i = 0, \ldots, n \): ; Note that bias weight \( w_0 \) is changed just like all other weights!
   \[
   w_i \leftarrow w_i + \Delta w_i
   \]
   where
   \[
   \Delta w_i = \eta (t^k - y^k) x_i^k
   \]
4. Go to 1.
Example

Training set:
((0, 0), 1)
(1, 1), 0)
(1, 0), 0)

Initial weights:
\{w_0, w_1, w_2\} = \{0.4, -0.1, 0.2\}

What is initial accuracy?

Apply perceptron learning rule for one epoch with \(\eta = 0.1\)

What is new accuracy?
In-Class Exercise 2
Recognizing Handwritten Digits

- **MNIST dataset**
  - 60,000 training examples
  - 10,000 test examples

Each example is a 28×28-pixel image, where each pixel is a grayscale value in [0,255].

See csv files.

First value in each row is the target class.

Label: “2”
Architecture for handwritten digits classification:
10 individual perceptrons

785 inputs
($= 28 \times 28 + 1$)

bias input

Fully connected

Recognize: ‘0’
Architecture for handwritten digits classification:
10 individual perceptrons

785 inputs
(= 28×28 + 1)

Fully connected

Recognize: ‘1’

bias input
Architecture for handwritten digits classification:
10 individual perceptrons

785 inputs
(= 28×28 + 1)

Fully connected

Recognize: ‘2’
Architecture for handwritten digits classification:
10 individual perceptrons

785 inputs
(= 28×28 + 1)

Fully connected

bias input

Recognize: ‘2’ Etc.
Preprocessing
(To keep weights from growing very large)

Scale each feature to a fraction between 0 and 1:

$$x'_i = \frac{x_i}{255}$$
Processing an input

For each perceptron, compute

\[ \sum_{i=0}^{n} w_i x_i \]
For each perceptron, compute
$$\sum_{i=0}^{n} W_i x_i$$

- Perceptron ‘0’
  - \( \sum_{i=0}^{n} w_i x_i = 6.5 \)
- Perceptron ‘1’
  - \( \sum_{i=0}^{n} w_i x_i = 2.4 \)
- Perceptron ‘2’
  - \( \sum_{i=0}^{n} w_i x_i = -7.2 \)

Etc.
For each perceptron, if \( \sum_{i=0}^{n} w_i x_i > 0 \), then output \( y = 1 \); otherwise \( y = 0 \).
Computing outputs

For each perceptron, if \( \sum_{i=0}^{n} w_i x_i > 0 \), then output \( y = 1 \); otherwise \( y = 0 \).
Computing targets

If input’s class corresponds to perceptron’s class, then target $t = 1$, otherwise target $t = 0$.

For example, suppose $x$ is a ‘2’.

Perceptron ‘0’
- $\sum_{i=0}^{n} w_i x_i = 6.5 \quad y = 1$

Perceptron ‘1’
- $\sum_{i=1}^{n} w_i x_i = 2.4 \quad y = 1$

Perceptron ‘2’
- $\sum_{i=2}^{n} w_i x_i = -7.2 \quad y = 0$

Etc.
Computing targets

If input’s class corresponds to perceptron’s class, then target $t = 1$, otherwise target $t = 0$.

For example, suppose $x$ is a ‘2’.

Input $x$

Perceptron ‘0’

$\sum_{i=0}^{n} w_i x_i = 6.5 \quad y = 1, t = 0$

Perceptron ‘1’

$\sum_{i=0}^{n} w_i x_i = 2.4 \quad y = 1, t = 0$

Perceptron ‘2’

$\sum_{i=0}^{n} w_i x_i = -7.2 \quad y = 0, t = 1$

Etc.
Learning

Repeat for $M$ epochs:

Perceptron ‘0’

\[ \sum_{i=0}^{n} w_i x_i = 6.5 \quad y = 1, \quad t = 0 \]

Perceptron ‘1’

\[ \sum_{i=0}^{n} w_i x_i = 2.4 \quad y = 1, \quad t = 0 \]

Perceptron ‘2’

\[ \sum_{i=0}^{n} w_i x_i = -7.2 \quad y = 0, \quad t = 1 \]

Etc.

Input $x$
Learning

Repeat for $M$ epochs:

For each training example $\mathbf{x}$:

Perceptron ‘0’

\[ w_{0} \cdot \sum_{i=0}^{n} w_{i} x_{i} = 6.5 \quad y = 1, \ t = 0 \]

Perceptron ‘1’

\[ w_{1} \cdot \sum_{i=0}^{n} w_{i} x_{i} = 2.4 \quad y = 1, \ t = 0 \]

Perceptron ‘2’

\[ w_{2} \cdot \sum_{i=0}^{n} w_{i} x_{i} = -7.2 \quad y = 0, \ t = 1 \]

Etc.
Learning

Repeat for $M$ epochs:

For each training example $x$:

In each perceptron, adjust all weights, including bias weight, according to perceptron learning rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t - y)x_i$$

If $t = y$, weights do not change.
Learning

Repeat for $M$ epochs:

For each training example $x$:

In each perceptron, adjust all weights, including bias weight, according to perceptron learning rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t - y)x_i$$

If $t = y$, weights do not change.

Goal of learning: We want $y = 1$ if perceptron corresponds to input; $y = 0$ for all other perceptrons.
Determining predictions (for computing accuracy)

The prediction of the group of perceptrons is the digit corresponding to the highest value of

\[ \sum_{i=0}^{n} w_i x_i \]

Perceptron ‘0’
\[ w_0' \]
\[ \sum_{i=0}^{n} w_i x_i = 6.5 \quad y = 1, \ t = 0 \]

Perceptron ‘1’
\[ w_1' \]
\[ \sum_{i=0}^{n} w_i x_i = 2.4 \quad y = 1, \ t = 0 \]

Perceptron ‘2’
\[ w_2' \]
\[ \sum_{i=0}^{n} w_i x_i = -7.2 \quad y = 0, \ t = 1 \]

Etc.
The prediction of the group of perceptrons is the digit corresponding to the highest value of

$$\sum_{i=0}^{n} w_i x_i$$

**Prediction is ‘0’**
The prediction of the group of perceptrons is the digit corresponding to the highest value of

\[ \sum_{i=0}^{n} w_i x_i \]

**Prediction is '0'**

You need the prediction values to compute the accuracy of the system at different epochs.

Accuracy = fraction of correct predictions
Homework 1

• Implement perceptron (785 inputs, 10 outputs) and perceptron learning algorithm in any programming language you would like.

• Download MNIST data from class website

• Preprocess MNIST data: Scale each feature to a fraction between 0 and 1.
Homework 1 Experiments

In each experiment, train the 10 perceptrons with the given learning rate:
Experiment 1: $\eta = 0.001$
Experiment 2: $\eta = 0.01$
Experiment 3: $\eta = 0.1$

For each learning rate:

1. Initialize perceptrons with small random weights $w_i \in [-0.05, 0.05]$ (chosen independently for each connection in each perceptron)

2. Run perceptron learning for 50 epochs. At each epoch (including epoch 0), compute accuracy on training and test data. (Don’t change the weights while computing accuracy.)
Homework 1: Presenting results

For each experiment, plot training and test accuracy over epochs:
Homework 1: Presenting results

For each experiment, plot training and test accuracy over epochs:

(Note: This is just an illustration. Your accuracies likely won’t get this high.)
For each experiment, give confusion matrix for results on test data after 50 epochs of training.

<table>
<thead>
<tr>
<th>Actual class</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In each cell \((i,j)\), put number of test examples that were predicted (classified) as class \(i\), and whose actual class is class \(j\).
Homework 1 FAQ

Q: Can I use code for perceptrons from another source?
A: No, you need to write your own code. You can use vector / matrix multiplication libraries.

Q: How long will it take to train the perceptrons?
A: Depends on your computer and your code, but probably not more than an hour. It helps to use vectorized functions.

Q: What accuracy should I expect on the test set?
A: It depends on initial weights, but probably over 80%.

Q: Can I ask questions while doing the assignment?
A: Yes, feel free to ask questions! You can ask me, the TA, and fellow students for help (but not code). You can also send questions to the class mailing list (and you can answer questions as well).

Q: Should I wait until the last minute?
A: No!!! You only have a week and a half. Start as soon as possible.
Homework: Report

- Report should be in pdf format and include graphs and formatted confusion matrices

- Code should be in plain text file
In-Class Exercise 3