Intro to Machine Learning

Recitation 2: Perceptron, Visualization
Agenda

- Perceptron algorithm
  - Supervised algorithm for binary classification
- Gradient decent
- Weights visualization
<table>
<thead>
<tr>
<th>Age</th>
<th>Education Number</th>
<th>Marital Status</th>
<th>Sex</th>
<th>Hours-per-week</th>
<th>Native-Country</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>12</td>
<td>Married</td>
<td>Female</td>
<td>45</td>
<td>US</td>
<td>&lt;=50K</td>
</tr>
<tr>
<td>41</td>
<td>16</td>
<td>Divorced</td>
<td>Female</td>
<td>30</td>
<td>US</td>
<td>&lt;50K</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>Single</td>
<td>Male</td>
<td>25</td>
<td>Canada</td>
<td>&lt;=50K</td>
</tr>
<tr>
<td>56</td>
<td>12</td>
<td>Divorced</td>
<td>Male</td>
<td>45</td>
<td>IL</td>
<td>&gt;50K</td>
</tr>
<tr>
<td>33</td>
<td>15</td>
<td>Single</td>
<td>Male</td>
<td>50</td>
<td>France</td>
<td>&lt;=50K</td>
</tr>
</tbody>
</table>
**Recall**

- In kNN every dimension in the input has equal contribution
- In reality some features are more important than others
- We would like to learn the importance of each feature / to weight them

<table>
<thead>
<tr>
<th>Age</th>
<th>Education Number</th>
<th>Marital Status</th>
<th>Sex</th>
<th>Hours-per-week</th>
<th>Native-Country</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>12</td>
<td>Married</td>
<td>Female</td>
<td>45</td>
<td>US</td>
<td>&lt;=50K</td>
</tr>
<tr>
<td>41</td>
<td>16</td>
<td>Divorced</td>
<td>Female</td>
<td>30</td>
<td>US</td>
<td>&lt;50K</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>Single</td>
<td>Male</td>
<td>25</td>
<td>Canada</td>
<td>&lt;=50K</td>
</tr>
<tr>
<td>56</td>
<td>12</td>
<td>Divorced</td>
<td>Male</td>
<td>45</td>
<td>IL</td>
<td>&gt;50K</td>
</tr>
<tr>
<td>33</td>
<td>15</td>
<td>Single</td>
<td>Male</td>
<td>50</td>
<td>France</td>
<td>&lt;=50K</td>
</tr>
</tbody>
</table>
Perceptron

\[
o(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if } w_0 + w_1x_1 + \cdots + w_nx_n > 0 \\
-1 & \text{otherwise}
\end{cases}
\]
Perceptron
Perceptron Training

- **Objective:**
  - To find a separating line (hyperplane) between positive and negative examples
- **Start with some random line (random weights)**
- **Look at one example at a time:**
  - If the example is on the wrong side of the hyperplane, rotate the line very slightly towards the example
- **Since the hyperplane is represented by \( \mathbf{w} \), this means altering \( \mathbf{w} \) slightly**

**How can we alter \( \mathbf{w} \) so that the hyperplane shifts in the right direction?**
Perceptron Training

Let \( w \leftarrow (0, 0, 0, \ldots, 0) \)

Repeat

Accept training example \( i : (x_i, y_i) \)

\[ u_i \leftarrow w \cdot x_i \]

if \( y_i \cdot u_i \leq 0 \)

\[ w \leftarrow w + y_i x_i \]
Perceptron Training

- $w^1 \cdot x_3 < 0$
- $w^2 = w^1 + y_3 x_3$
- $w^2 x_3 = (w^1 + y_3 x_3) \cdot x_3$
- $w^2 x_3 = w^1 x_3 + y_3 \|x_3\|^2 > w_1 \cdot x_3$
Code Example
Example

- Dataset: MNIST database of handwritten digits (10 classes)
- In following example we use only 2 (number 8 vs. number 1)

```python
def read_data(test_fraction=0.1):
    test_fraction = 0.1
    X, Y = mnist.data / 255., mnist.target
    x = [ex for ex, ey in zip(X, Y) if ey == 1 or ey == 8]
    y = [1 if ey == 1 else -1 for ex, ey in zip(X, Y) if ey == 1 or ey == 3]
    x, y = shuffle(x, y, random_state=1)
```
Normalization

Why is important to normalize?
- convergence rate (some weights will update faster than others)
- contribution to loss (euclidean distance)
- many components in neural nets assume normalized values (e.g. sigmoid)
Normalization Methods

- Min-Max Normalization

\[ v' = \frac{v - \text{min}_A}{\text{max}_A - \text{min}_A} (\text{new}_\text{max}_A - \text{new}_\text{min}_A) + \text{new}_\text{min}_A \]

- Z-Score Normalization

\[ v' = \frac{v - \text{mean}_A}{\text{stand}_\text{dev}_A} \]
import matplotlib.pyplot as plt

from matplotlib import image

plt.Figure(1)
for i in range(1,26):
    ax = plt.subplot(5,5,i)
    ax.imshow('off'
    if y[i] > 0:
        ax.imshow(x[i].reshape(28,28), cmap='gray'
    else:
        ax.imshow((255-x[i].reshape(28,28), cmap='gray')
plt.show()
Example

```python
import numpy as np
import random

# weight vector
m = len(x)
d = len(x[0])
mxn = m * d
w = np.zeros(mxn)

# Perceptron
T = 5000
for t in range(0, T):
    # choose example
    i = random.randint(0, m-1)
    # predict
    y_hat = np.sign(np.dot(w, z[i]))
    # update
    if y_hat * y[i] <= 0:
        w = w + y[i] * z[i]

# testing
M_correct = 0
for i in range(0, m):
    if z[i][int(y[i])].dot(w) > 0:
        M_correct = M_correct + 1
print "perceptron err =", float(M_correct) / m

perceptron err = 0.0022345767010
```

Let \( w \leftarrow (0,0,0,...,0) \)
Repeat

Accept training example \( i : (x_i, y_i) \)

\[ u_i \leftarrow w \cdot x_i \]

if \( y_i \cdot u_i \leq 0 \)

\[ w \leftarrow w + y_i x_i \]
Practical guidelines

- In practice we would like to make smaller steps
- Use *learning rate*
Example

```python
# show the mask learnt by Perceptron
plt.figure(2);
ax1 = plt.subplot(1,2,1);
ax1.axis('off'); # no need for axis marks
ax2 = plt.subplot(1,2,2);
ax2.axis('off'); # no need for axis marks
ax1.imshow(w_perceptron.reshape(28,28), cmap='gray');
tmp = 1/(1 + np.exp(-10 * w_perceptron / w_perceptron.max()));
ax2.imshow(tmp.reshape(28,28), cmap='gray');
plt.show();
```
Example

```python
# check performance on test data
X, Y = mnist.data[60000:] / 255., mnist.target[60000:]
x = [ex for ex, ey in zip(X, Y) if ey == 1 or ey == 8]
# convert 1 to +1 and 8 to -1
y = [1 if ey == 1 else -1 for ex, ey in zip(X, Y) if ey == 1 or ey == 8]
m = len(x)

M_perceptron = 0
for t in range(0, m):
    y_hat = np.sign(np.dot(w_perceptron, x[t]))
    if y[t] != y_hat:
        M_perceptron += 1
print "perceptron err =", float(M_perceptron) / m

perceptron err = 0.0132764343291
```
Extending to k-C classes
From **Binary** to **Multiclass** - One vs. All

- Y vs. B, C, R
- C vs. B, Y, R
- R vs. B, Y, C
- B vs. R, Y, C
From **Binary** to **Multiclass** - One vs. One

- Train classifier for every pair of classes (sometimes called pairwise)
- Number of classifiers $k$ choose 2
From **Binary** to **Multiclass**

- Define K weight vectors (matrix)

**Binary Case**

\[ \hat{y} = \text{sign}(w \cdot x_i) \]

if \( \hat{y} \neq y_i \):

\[ w \leftarrow w + y_i x_i \]

**Multiclass Case**

\[ \hat{y} = \arg\max_{y \in \{1, \ldots, k\}} w^y \cdot x \]

if \( \hat{y} \neq y_i \):

\[ w^r \leftarrow w^r + \tau_r x_i \]

\[ \tau_r = \begin{cases} 
+1 & \text{if } r = y_i \\
-1 & \text{if } r = \hat{y} \\
0 & \text{otherwise}
\end{cases} \]
From **Binary** to **Multiclass**

- Define K weight vectors (matrix)

**Binary Case**

$$\hat{y} = \text{sign}(w \cdot x_i)$$

If $\hat{y} \neq y_i$:

$$w \leftarrow w + y_i x_i$$

**Multiclass Case**

$$\hat{y} = \arg\max_{y \in \{1, \ldots, k\}} w^y \cdot x$$

If $\hat{y} \neq y_i$:

$$w^r \leftarrow w^r + \tau_r x_i$$

$$\tau_r = \begin{cases} +1 & \text{if } r = y_i \\ -1 & \text{if } r = \hat{y} \\ 0 & \text{otherwise} \end{cases}$$
From **Binary** to **Multiclass**

```python
epochs = 10
for e in range(epochs):
    X_train, Y_train = shuffle(X_train, Y_train, random_state=1)
    for x, y in zip(X_train, Y_train):
        # predict
        y_hat = np.argmax(np.dot(w, x))
        # update
        if y != y_hat:
            w[y, :] = w[y, :] + eta * x
            w[y_hat, :] = w[y_hat, :] - eta * x
```
Perceptron Properties

- Linear classifier
- Will converge if the data is linearly separable
- Doesn’t guarantee correct classification after update
Gradient Descent

- We would like to find the parameters that minimize loss function
- Calculate the gradient of the loss w.r.t to the parameters

\[ w^t \leftarrow w^{t-1} - \eta \cdot \frac{\partial l}{\partial w^{t-1}} \]

- In each update we average the gradients for all training examples
- Convergence is slow (per epoch)
Stochastic Gradient Descent

- In every update sample an example at i.i.d
- In practice we shuffle the data after every epoch

\[ w^t \leftarrow w^{t-1} - \eta \cdot \frac{\partial l}{\partial w^{t-1}} \]

- In each update we calculate the gradients of a single example
- Approximate the full gradient
- Convergence is noisy but faster (per epoch)
Perceptron as an ERM

\[
J(w) = \frac{1}{N} \sum_{i=1}^{N} \max(0, -y_i w \cdot x_i)
\]

\[
J_i(w) = \max(0, -y_i w \cdot x_i)
\]

\[
\nabla J_i = \begin{cases} 
0 & \text{if } y_i w \cdot x_i > 0 \\
-y_i x_i & \text{otherwise}
\end{cases}
\]
Questions?