RECITATION 3: LOGISTIC REGRESSION & MLE/MAP

10-301/10-601 Introduction to Machine Learning (Summer 2024)
http://www.cs.cmu.edu/~hchai2/courses/10601

1 Gradient Descent for Linear Regression

Consider the following dataset:

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$
x_1	1.0	2.0	3.0	4.0	5.0
x_2	-2.0	-5.0	-6.0	-8.0	-11.0
y	2.0	4.0	7.0	8.0	11.0

1. Suppose we want to implement gradient descent using a stepsize of $\eta = 0.1$. Assuming $\mathbf{w}$ has been initialized to $[0, 0, 0]^T$, let's perform one iteration of gradient descent: What is the gradient of the objective function $\ell(\mathbf{w})$ with respect to $\mathbf{w}$: $\nabla_{\mathbf{w}}\ell(\mathbf{w})$?

Your answer:

Solution

$$\frac{d\ell(\mathbf{w})}{dw_k} = \frac{1}{5}\sum_{i=1}^{5} -2x_k^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)})$$

$$\nabla_{\mathbf{w}}\ell(\mathbf{w}) = \begin{pmatrix} \frac{d\ell(\mathbf{w})}{dw_0} \\ \frac{d\ell(\mathbf{w})}{dw_1} \\ \frac{d\ell(\mathbf{w})}{dw_2} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) \\ \frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) \end{pmatrix}$$

2. How do we carry out the update rule?

Your answer:

Solution Given the initial value

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

we follow the update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \underbrace{\eta}_{\text{“Cross-validated”}} \nabla_{\mathbf{w}|\mathbf{w}=\mathbf{w}^{(k)}} \ell(\mathbf{w})$$

where $k = 0$ here

$$\frac{1}{5}\sum_{i=1}^{5} -2x_0^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) = \frac{-2}{5} \cdot (2+4+7+8+11) = -12.8$$

$$\frac{1}{5}\sum_{i=1}^{5} -2x_1^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) = \frac{-2}{5} \cdot (2+8+21+32+55) = -47.2$$

$$\frac{1}{5}\sum_{i=1}^{5} -2x_2^{(i)}(y^{(i)} - \sum_{j=0}^{2} w_j x_j^{(i)}) = \frac{-2}{5} \cdot (-4-20-42-64-121) = 100.4$$

$$\begin{aligned} \rightarrow \mathbf{w}^{(1)} &= \mathbf{w}^{(0)} - \alpha \nabla_{\mathbf{w}|\mathbf{w}=\mathbf{w}^{(0)}} \ell(\mathbf{w}) \\ &= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - 0.1 \begin{pmatrix} -12.8 \\ -47.2 \\ 100.4 \end{pmatrix} \\ &= \begin{pmatrix} 1.28 \\ 4.72 \\ -10.4 \end{pmatrix} \end{aligned}$$

3. How could we pick which value of η to use if we weren't given the step size?

Your answer:

Solution Cross-validation or use a held-out validation dataset

2 Logistic Regression

2.1 MLE/MAP

As a reminder, in MLE, we have

$$\begin{aligned}\hat{w}_{MLE} &= \underset{w}{\operatorname{argmax}}\, p(\mathcal{D}|w) \\ &= \underset{w}{\operatorname{argmin}} - \log\left(p(\mathcal{D}|w)\right)\end{aligned}$$

For MAP, we have

$$\begin{aligned}\hat{w}_{MAP} &= \underset{w}{\operatorname{argmax}}\, p(w|\mathcal{D}) \\ &= \underset{w}{\operatorname{argmax}}\, \frac{p(\mathcal{D}|w)p(w)}{\text{Normalizing Constant}} \\ &= \underset{w}{\operatorname{argmax}}\, p(\mathcal{D}|w)p(w) \\ &= \underset{w}{\operatorname{argmin}} - \log\left(p(\mathcal{D}|w)p(w)\right)\end{aligned}$$

---

1. Suppose you are an avid BTS stan who monitors the social media accounts of each of the members. Suppose you wish to find the probability that a BTS member will post at any time of day. Over three days you look on Instagram and find the following number of new posts:

   $$x = [3, 4, 1]$$

   A fellow stan tells you that this comes from a Poisson distribution:

   $$p(x|w) = \frac{e^{-w}w^x}{x!}$$

   Also, you are told that $w \sim \text{Gamma}(2, 2)$ — that is, its pdf is:

   $$p(w) = \frac{1}{4}we^{-\frac{w}{2}},\ w > 0$$

(a) Calculate $\hat{w}_{MLE}$. (Example from https://en.wikipedia.org/wiki/Conjugate_prior)

Solution

Note:

$$p(\mathcal{D}|w) = \frac{e^{-w}w^3}{3!}\frac{e^{-w}w^4}{4!}\frac{e^{-w}w^1}{1!}$$

$$\begin{aligned}\hat{w}_{MLE} &= \underset{w}{\operatorname{argmin}} - \log\left(p(\mathcal{D}|w)\right) \\ &= \underset{w}{\operatorname{argmin}} - \log\left(\frac{e^{-w}w^3}{3!}\frac{e^{-w}w^4}{4!}\frac{e^{-w}w^1}{1!}\right) \\ &= \underset{w}{\operatorname{argmin}} - \log e^{-3w}w^8 + \log 144\end{aligned}$$

Setting the derivative equal to zero yields

$$\begin{aligned}0 &= 3 - \frac{8}{w} \\ \implies w_{MLE} &= \frac{8}{3} = 2.667\end{aligned}$$

(b) Calculate $\hat{w}_{MAP}$. (Example from https://en.wikipedia.org/wiki/Conjugate_prior)

Solution

Note:

$$p(\mathcal{D}|w) = \frac{e^{-w}w^3}{3!}\frac{e^{-w}w^4}{4!}\frac{e^{-w}w^1}{1!}$$

$$\begin{aligned}
\hat{w}_{MAP} &= \underset{w}{\operatorname{argmin}} - \log\left(p(\mathcal{D}|w)p(w)\right) \\
&= \underset{w}{\operatorname{argmin}} - \log\left(\frac{e^{-w}w^3}{3!}\frac{e^{-w}w^4}{4!}\frac{e^{-w}w^1}{1!}\frac{1}{\Gamma(2)2^2}w^{(2-1)}e^{\frac{-w}{2}}\right) \\
&= \underset{w}{\operatorname{argmin}} - \log e^{-3w}w^8 w^{(2-1)}e^{\frac{-w}{2}} \\
&= \underset{w}{\operatorname{argmin}} - \log e^{-3w-\frac{w}{\beta}}w^{8+\alpha-1} \\
&= \underset{w}{\operatorname{argmin}} -((-3w - \frac{w}{2}) + (8+2-1)\log(w))
\end{aligned}$$

Setting the derivative equal to zero yields

$$\begin{aligned}
0 &= -3 - \frac{1}{2} + \frac{(7+2)}{w} \\
\implies w_{MAP} &= \frac{7+2}{3+\frac{1}{2}} = 2.57142857143
\end{aligned}$$

2.2 Logistic Regression: Toy Example

Let's go through a toy problem.

Y	X_1	X_2	X_3
1	1	2	1
1	1	1	-1
0	1	-2	1

1. What is $\ell(\mathbf{w})$ of above data given initial $\mathbf{w} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$?

Solution $\ell(\mathbf{w}) = \frac{-1}{N} \sum y^i \log(\sigma(w^T x)) + (1 - y^i) \log(1 - \sigma(w^T x))$

$\ell(\mathbf{w}) = -\frac{1}{3}[\log(\sigma(3)) + \log(\sigma(-1)) + \log(1 - \sigma(-5))] \approx 0.46$

2. Observe that $\frac{\partial \ell^{(i)}(\mathbf{w})}{\partial w_j} = x_j^{(i)}(\sigma(\mathbf{w}^T \mathbf{x}) - y^{(i)})$. Using this information, calculate $\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_1}$, $\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_2}$ and $\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_3}$ for first training example. Note that $\sigma(3) \approx 0.95$. **Solution**

$$\frac{\partial \ell^{(i)}(\mathbf{w})}{\partial w_j} = x_j^{(i)}(\sigma(\mathbf{w}x) - y^{(i)})$$

$$\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_1} = (\sigma(3) - 1)1 = -0.05$$
$$\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_2} = (\sigma(3) - 1)2 = -0.10$$
$$\frac{\partial \ell^{(1)}(\mathbf{w})}{\partial w_3} = (\sigma(3) - 1)1 = -0.05$$

3. Calculate $\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_1}$, $\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_2}$ and $\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_3}$ for second training example. Note that $\sigma(-1) \approx 0.25$. **Solution**

$$\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_1} = (\sigma(-1) - 1)1 = -0.75$$
$$\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_2} = (\sigma(-1) - 1)1 = -0.75$$
$$\frac{\partial \ell^{(2)}(\mathbf{w})}{\partial w_3} = (\sigma(-1) - 1) - 1 = 0.75$$

4. Assuming we are doing stochastic gradient descent with a learning rate of 1.0, what are the updated parameters $\mathbf{w}$ if we update $\mathbf{w}$ using the second training example?
Solution

$$\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} - 1 \begin{bmatrix} -0.75 \\ -0.75 \\ 0.75 \end{bmatrix} = \begin{bmatrix} -1.25 \\ 2.75 \\ 0.25 \end{bmatrix}$$

5. What is the new $\ell(\mathbf{w})$ after doing the above update? Would you expect it to decrease or increase?
   Solution $\ell(\mathbf{w}) = 0.09$

   It should decrease for logistic classifier to learn.

6. Given a test example where $(X_1 = 1, X_2 = 3, X_3 = 4)$, what will the classifier output following this update?
   Solution $\mathbf{w}^T X = -1.25 * 1 + 2.75 * 3 + 0.25 * 4 = 8$

   $\sigma(\mathbf{w}^T X) = \sigma(8) \approx 0.999 > 0.5 \implies Y = 1$

3 Programming

3.1 Feature Representation for Sentiment Classification

In many machine learning problems, we will want to find appropriate representations for the inputs of the algorithm we are developing. In Programming Assignment 3, we will work on using logistic regression for a sentiment classification task, where our algorithm takes a paragraph of movie review as the input and outputs a binary value denoting whether the review is positive or not. To build an appropriate representation for the input (aka. the review text), we consider a representation built using GloVe[1] word embeddings.

In this section, consider a scenario where we are interested in representing the following text:

```
a hot dog is not a sandwich because it is not square
```
(1)

We consider the following dictionary (denoted below as **Vocab**) as the set of vocabulary that we will consider. Note that the vocabulary dictionary might not contain all words in the text shown above.

```
dictionary = {
    "the": 0,
    "square": 1,
    "hot": 2,
    "is": 3,
    "not": 4,
    "a": 5,
    "happy": 6,
    "sandwich": 7
}
```

[1] You can read more about GloVe in the original research paper. You can also check out this explainer on Word2Vec, which is a similar technique for obtaining word embeddings.

1. **Word Embedding Based Representation**

1. Word embeddings are reduced dimension vector representations (features) of words. Given a single word in the dictionary, word embeddings can convert it to a vector of fixed dimension. In Programming Assignment 3, we will provide a dictionary file specifying pre-computed mappings between every word in **Vocab** and their corresponding word embeddings. To facilitate better understanding towards word embeddings, we produce a plot showing the spatial relationship between several sample words from the vocabulary used in Programming Assignment 3, with their corresponding word embeddings (reduced to 2D vectors from 300D vectors using a technique called PCA we will learn about later in this course!):

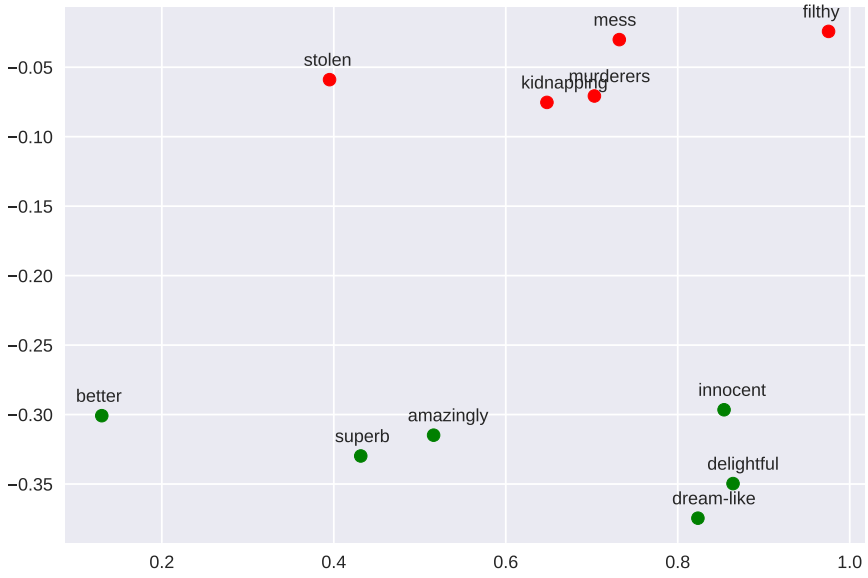

Figure 1: Visualization of word embeddings. We select a few positive words (shown in green) and a few negative words (shown in red). To make the plot, we map the high-dimensional word representations of these words to 2D space using PCA and then visualize them in the scatter plot above.

Please comment on your observations and findings based on this plot.

Solution Closer-related words are located closer in the representation space, while farther-related words are located farther from each other.

2. Now, we much translate these word embeddings to sentence embeddings (a vector representing the sentence as a whole). One approach to building a sentence embedding is to average out the vector representation of every word in the sentence that is in the dictionary. For example, given text "`a hot dog flies like a sandwich`", we can find the sentence embedding for this text by taking the average of the vector representation of the words "`a`", "`hot`", "`a`", and "`sandwich`".

   Now suppose we have the following word embedding dictionary for building sentence embeddings (this is a toy example used for illustrative purposes; actual word embeddings will have higher dimensions than this example):

```
dictionary = {
    "the": [0.2, 0.3],
    "square": [0.8, 0.9],
    "hot": [0.1, -0.2],
    "is": [0.1, 0.1],
    "not": [-0.2, -0.3],
    "a": [0.0, 0.0],
    "happy": [0.4, 0.4],
    "sandwich": [0.2, -0.3]
}
```

   Write the word embedding based representation of the sample text define above, repeated here for convenience:

$$\texttt{a hot dog is not a sandwich because it is not square} \tag{2}$$

Solution

$$\begin{aligned}\phi_2(\mathbf{x}) &= \frac{1}{9}\big(f(\text{square}) + f(\text{hot}) + 2\cdot f(\text{is}) + 2\cdot f(\text{not}) + 2\cdot f(\text{a}) + f(\text{sandwich})\big)\\ &= \begin{bmatrix}0.1 & 0.0\end{bmatrix}^T.\end{aligned}$$

3.2 Gradient Descent and Stochastic Gradient Descent

Now we will compare two different optimization methods using pseudocode. Consider a model with parameter $\mathbf{w} \in \mathbb{R}^M$ being trained with a design matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and labels $\mathbf{y} \in \mathbb{R}^N$. Say we update $\mathbf{w}$ using the objective function $\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} \ell^{(i)}(\mathbf{w}|\mathbf{x}^{(i)}, y^{(i)}) \in \mathbb{R}$. Recall that an epoch refers to one complete cycle through the dataset.

1. Complete the pseudocode for gradient descent.

```
def dℓ(w, X, y, i):
    (omitted)   # Returns ∂ℓ^(i)(w|x^(i), y^(i))/∂w
    # You may call this function in your pseudocode.

def GD(w, X, y, learning_rate):
    for epoch in range(num_epoch):
        Complete this section with the update rule
    return w  # return the updated w
```

Solution grad = zeros(M)
for i in range(N):
    grad += dℓ(w, X, y, i)
w -= learning_rate * grad / N

2. Complete the pseudocode for stochastic gradient descent that samples *without* replacement.

```
def dℓ(w, X, y, i):
    (omitted)   # Returns ∂ℓ^(i)(w|x^(i), y^(i))/∂w
    # You may call this function in your pseudocode.

def SGD(w, X, y, learning_rate):
    for epoch in range(num_epoch):
        indices = shuffle(range(len(X)))
        for i in indices:
            Complete this section with the update rule
    return w  # return the updated w
```

Solution w -= learning_rate * $d\ell(w, X, y, i)$

3.3 The Need For Speed: Vectorization and Numpy

Performing mathematical operations on vectors and matrices is ubiquitous in most machine learning algorithms. Whether it's a simple similarity measure that works by calculating the dot product between two vectors, or deep neural networks, they all involve repeated matrix operations. This makes it imperative that our underlying code design to perform matrix operations is efficient.

3.3.1 The Perils of Python

While Python is widely the language of choice for machine learning researchers across the globe (thanks to the speed of development and code readability it offers and the support it enjoys from the open-source community), Python as a high-level language on average is much slower than a lower level language like C++. To combat this, libraries like numpy and scipy implement most of the back-end operations they perform in C/C++, while providing wrappers in Python to be able to call underlying C code seamlessly from a Python script.

3.3.2 Speed Comparison: Numpy and Python

We highly recommend you to use *numpy* extensively in this course, it will be difficult to pass the programming portion of Programming Assignment 3 without writing most of your matrix operations in numpy. In this section, we'll see why.

Consider you have two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. To see how similar they are, as measured by the cosine angle between them, you want to compute their dot product. This translates to the following operation:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + ... + a_n b_n$$

When translated to code, notice how the dot product in NumPy is a whopping 100x faster than the native Python!

```
from timeit import timeit
import numpy as np
import array

VECTOR_SIZE = int(1e8)

# NumPy arrays
a = np.random.rand(VECTOR_SIZE)
b = np.random.rand(VECTOR_SIZE)

# Python arrays
aArr = array.array('d', a)
bArr = array.array('d', b)


def test_np():
    return np.dot(a, b)

# faster than multiprocessing, python lists, or numpy arrays with
    python loops
# faster than using a range and indexing
```

```
def test_py_arr():
    return sum(x * y for x, y in zip(aArr, bArr))

def time_dot_product(f):
    return timeit(f, setup=f, number=5) / 5



if __name__ == "__main__":
    print(f"NumPy = {time_dot_product(test_np):.2f}")   # 0.05s
    print(f"Python on an array =
        {time_dot_product(test_py_arr):.2f}")   # 5.45s
```

3.3.3 Useful Numpy Operations

Some operations in numpy that you will find really useful in your assignments are:

- np.matmul: Matrix multiplication of two matrices
- np.unique: Returns unique elements along an axis.
- np.hstack: Stack two arrays horizontally (column-wise)
- np.expand_dims: Convert a row vector of size n into a matrix of size $n * 1$ or $1 * n$
- np.log, np.sum, np.exp, and so on...

You can read C vs. Python for more details, and you can also read these two tutorials (beginner, intermediate) from the official numpy website. For instance, understanding broadcasting is recommended. It will help you debug the shape errors you might face in all future homeworks.