Recitation 5
Logistic Regression

10-301/601: Introduction to Machine Learning

06/16/2022

This recitation consists of 3 parts: In part 1, we will look at code efficiency through the lens of the speed benefits that *numpy* provides over python for loops. In part 2, we will go over how to **represent text data using two different feature extraction methods**. Finally, Part 3 will go over the **negative log likelihood** and **gradient derivations** for **binary logistic regression**, as well as a small toy example. These materials are designed to help you with Homework 4.

1 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.

1.1 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.

1.2 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 Homework 4 without writing most of your matrix operations in numpy. In this section, we'll see why. Please refer to this Google Colab notebook for demos and example operations.

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 350x faster than the sequential native for loop in Python!

```
import time
import numpy as np
VECTOR_SIZE = int(1e8)
a = np.random.rand(VECTOR_SIZE)
b = np.random.rand(VECTOR_SIZE)

start_time = time.time()
ans = np.dot(a, b)
end_time = time.time()
print(f"Time taken to compute answer = {end_time-start_time:.2f}s")
## Output: Time taken to compute answer = 0.18s

ans = 0
start_time = time.time()
for i in range(len(a)):
    ans += (a[i] * b[i])
end_time = time.time()
print(f"Time taken to compute answer = {end_time-start_time:.2f}s")
## Output: Time taken to compute answer = 68.50s
```

1.3 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...

Other than the Colab notebook, 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.

2 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 Homework 4, 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 two different representations – (1) a bag-of-word representation and (2) a representation built using Word2vec[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. **Bag-of-words Representation**

   A bag-of-words representation $\boldsymbol{\phi}_1(\mathbf{x})$ of text $\mathbf{x}$ is defined by $\boldsymbol{\phi}_1(\mathbf{x}) = \mathbf{1}_{\mathbf{occur}}(\mathbf{x^{(i)}}, \mathbf{Vocab})$, indicating which words in vocabulary **Vocab** of the dictionary occur at least once in the movie review example $\mathbf{x}^{(i)}$. Let $\mathbf{x}$ be the sample text defined above. Write the bag-of-words representation of $\mathbf{x}$.

[1] You can read more about Word2vec on this blog (or the original research paper for the enthusiasts!)

2. **Word Embedding Based Representation**

   (a) 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 Homework 4, 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 Homework 4, 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!):

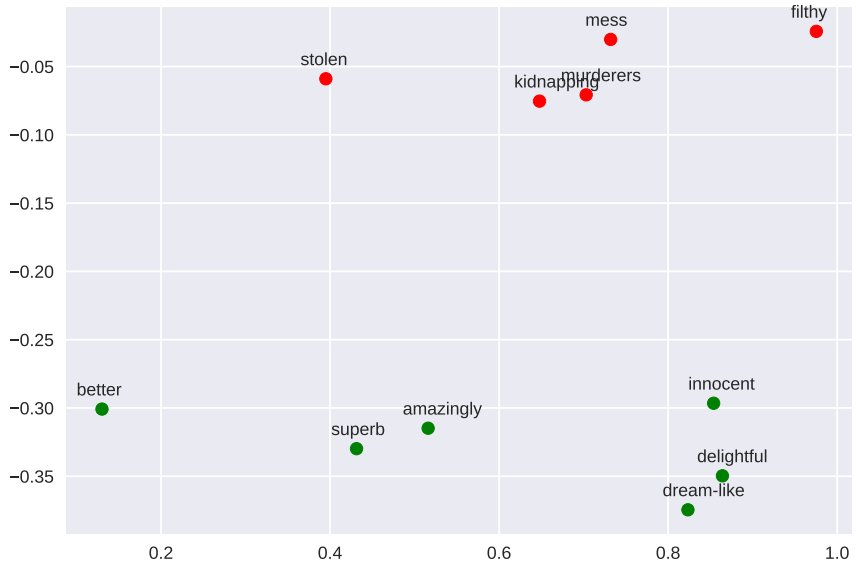

Figure 1: Visualization of word embeddings. We select a few positive words (shown in green) and a few negative words (shown in red). To make the plot, we map the high-dimensional word representations of these words to 2D space using PCA and then visualize them in the scatter plot above.

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

(b) 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:

```
a hot dog is not a sandwich because it is not square
```
(2)

3 Binary Logistic Regression

Consider the following dataset,

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)}\right) \right\} \text{ where } \mathbf{x}^{(i)} \in \mathbb{R}^M, y^{(i)} \in \{0, 1\}.$$

Notice that instead of regressing on continuous variables, as we do in linear regression, we wish to predict on discrete variables, and, more specifically, binary outcomes (this process is called classification). Since we want to take noise into account when making predictions, we will predict the probability of each outcome given some inputs, each denoted $\mathbf{x}^{(i)}$. This means that the result of the linear combination of each input, $\mathbf{x}^{(i)}$ with the parameter vector, $\boldsymbol{\theta}$, or, $\boldsymbol{\theta}^T\mathbf{x}^{(i)}$, must be manipulated to fit between zero and one. We use the sigmoid function, $\sigma(\cdot)$, for this purpose.

Recall that

$$\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T\mathbf{x}^{(i)})} = \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1 + \exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}.$$

The conditional probability of $y^{(i)}$ given $\mathbf{x}^{(i)}$ is,

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}\right) = \begin{cases} \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}) & y^{(i)} = 1 \\ 1 - \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}) & y^{(i)} = 0 \end{cases}.$$

We can rewrite this as a single statement,

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}\right) = \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)})^{y^{(i)}}(1 - \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}))^{(1-y^{(i)})}.$$

Can you show why this is true?

We assume each observation of $y^{(i)}$ in the data is independent and identically distributed. This means we can write down the likelihood of the data as a product of negative conditional probabilities.

Let's plug in the expression for $p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}\right)$ and simplify.

In stochastic gradient descent, we use only a single $\mathbf{x}^{(i)}$. Given $\boldsymbol{\phi}^{(i)} = \sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ and

$$J^{(i)}(\boldsymbol{\theta}) = -y^{(i)} \log(\boldsymbol{\phi}^{(i)}) - (1 - y^{(i)}) \log(1 - \boldsymbol{\phi}^{(i)})$$

Show that the partial derivative of $J^{(i)}(\boldsymbol{\theta})$ with respect to the jth parameter θ_j is as follows:

$$\frac{\partial J^{(i)}(\boldsymbol{\theta})}{\partial \theta_j} = (\sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) - y^i) x_j^{(i)}$$

Remember,

$$\frac{\partial \boldsymbol{\phi}^{(i)}}{\partial \boldsymbol{\theta}_j} = \boldsymbol{\phi}^{(i)} * (1 - \boldsymbol{\phi}^{(i)}) * \frac{\partial \boldsymbol{\theta}^T \mathbf{x}^{(i)}}{\partial \theta_j}$$

.

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

(a) What is $J(\boldsymbol{\theta})$ of above data given initial $\boldsymbol{\theta} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$?

(b) Calculate $\frac{\partial J^{(1)}(\boldsymbol{\theta})}{\partial \theta_1}$, $\frac{\partial J^{(1)}(\boldsymbol{\theta})}{\partial \theta_2}$ and $\frac{\partial J^{(1)}(\boldsymbol{\theta})}{\partial \theta_3}$ for first training example. Note that $\sigma(3) \approx 0.95$.

(c) Calculate $\frac{\partial J^{(2)}(\boldsymbol{\theta})}{\partial \theta_1}$, $\frac{\partial J^{(2)}(\boldsymbol{\theta})}{\partial \theta_2}$ and $\frac{\partial J^{(2)}(\boldsymbol{\theta})}{\partial \theta_3}$ for second training example. Note that $\sigma(-1) \approx 0.25$.

(d) Assuming we are doing stochastic gradient descent with a learning rate of 1.0, what are the updated parameters $\boldsymbol{\theta}$ if we update $\boldsymbol{\theta}$ using the second training example?

(e) What is the new $J(\boldsymbol{\theta})$ after doing the above update? Should it decrease or increase?

(f) Given a test example where $(X_1 = 1, X_2 = 3, X_3 = 4)$, what will the classifier output following this update?