Advanced Machine Learning Assignment

Anjila Budathoki

January 2024

1 Refreshers on Optimization and probability fundamentals.

1. (Continuous optimization) Deriving the optimal solution θ^* that minimizes $f(\theta)$.

   Given, $f(\theta) = \sum_{i=1}^{n} w_i(\ x_i - \theta)^2$

   Assumption, $w_i > 0$

   To derive optimal solution θ^*,

   $f(\theta) = \sum_{i=1}^{n} w_i(x_i - \theta)^2$

   $argmin_\theta\ f(\theta) =$

   Derivating $f(\theta)$ w.r.t θ,

   $\frac{df(\theta)}{d\theta}$ = -2 $\sum_{i=1}^{n} w_i(x_i - \theta)$

   Setting derivative to 0, to find optimal solution:

   $0 =$ -2 $\sum_{i=1}^{n} w_i(x_i - \theta)$

   $0 = \sum_{i=1}^{n} w_i x_i - \sum_{i=1}^{n} w_i\theta$

   $\sum_{i=1}^{n} w_i\theta = \sum_{i=1}^{n} w_i x_i$

   $n\theta^* = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$

   $\theta^* = \frac{\sum_{i=1}^{n} x_i}{n}$

   What happens if some $w_i$are negative?

   Answer: When the w_i are negative, these negative values might cancel out positive terms or reduce the actual function output. Hence, it wouldn't give the accurate measure.

2. (Counting and combinatorics)

   The total ways of grouping 2n boys = 2nCn and total ways of grouping 2n boys in 2 groups is given by

$= \frac{2nCn}{2!}$

$= \frac{(2n)!}{2! * n! * n!}$

$= \frac{(2n)!}{2 * n! * n!}$

Now,

Removing tall boys from the group = (2n-2) and total boys in same group will be n-2, So, Total ways in which 2 tallest kids in the same group

= (2n-2)C(n-2)

$= \frac{(2n-2)!}{(n-2)! * (2n-2-n+2)!}$

$= \frac{(2n-2)!}{(n-2)! * (n)!}$

i. The probability that the two tallest kids will be in the same subgroup

$= \frac{(2n-2)!}{(n-2)! * n!} * \frac{2 * n! * n!}{(2n)!}$

$= \frac{(2n-2)!}{(n-2)!} * \frac{2n * (n-1) * (n-2)!}{2n * (2n-1) * (2n-2)!}$

$= \frac{n-1}{2n-1}$

ii. The probability that the two tallest kids will be in the different subgroups

= 1 - $\frac{n-1}{2n-1}$

$= \frac{2n-1-n+1}{2n-1}$

$= \frac{n}{2n-1}$

3. (Bayes rule)

Given:

Probability of a candidate knows the answer P(KA) = p

Probability of a candidate knows the answer P($\overline{KA}$) = 1 - p

Probability of correct answer given that candidate knows the answer P(CA | KA) = 0.99

Probability of correct answer given that candidate does not know the answer P(CA | $\overline{KA}$) = $\frac{1}{k}$

To find the conditional probability that the candidate knew the answer to a question, given that she has made the correct answer: P(KA | CA)

Using Bayes rule: P(A|B) = $\frac{P(B \mid A)P(A)}{P(A)}$

For marginal probability P(KA):

$= P(CA \mid KA)P(KA) + P(CA \mid \overline{K}A)P(\overline{KA})$

= 0.99p + $\frac{1}{k}$(1-p)

= $\frac{0.99pk + 1 - p}{k}$

P(KA | CA) = $\frac{P(CA \mid KA)P(KA)}{P(KA)}$

$= \frac{0.99pk}{0.99pk + 1 - p}$

4. (Likelihood and maximum likelihood)

Given a biased coin with the probability of head = p.

Sequence of outcomes: T,H,H,T,T,H,H,H,T,H.

Probability (likelihood) of observing this sequence

L(p) = (1-p) · p · p · (1-p) · (1-p) · p · p · p · (1-p) · p = p^6(1-p)^4

Calculate the value of p that maximizes the likelihood L(p).

Using log on both sides:

logL(p) = log(p^6(1-p)^4)

= log(p^6(1-p)^4)

= log(p^6)+ log(1-p)^4

= 6logp+ 4log(1-p)

Finding derivative of logL(p)

$\frac{dlogL(p)}{dp}$ = 6logp+ 4log(1-p)

= $\frac{6}{p}$ - $\frac{4}{1-p}$

Setting derivative to 0:

0 = $\frac{6}{p}$ - $\frac{4}{1-p}$

0 = $\frac{6-6p-4p}{p(1-p)}$

$0 = \dfrac{6 - 10p}{p(1-p)}$

0 = 6-10p

10p = 6

$p^* = \dfrac{6}{10}$

The value of p that maximizes the likelihood L(p) = $\dfrac{6}{10}$.

Maximizing logL(p) will maximize L(p) because log function is monotonic in nature, Hence, maximizing log likelihood is same as maximizing likelihood itself.

5. (Calculus, gradients)

$\nabla_w f = [\dfrac{\partial f}{\partial w_1}, \dfrac{\partial f}{\partial w_2}, \dfrac{\partial f}{\partial w_3}, ...\dfrac{\partial f}{\partial w_n}]^T$

Using chain-rule:

f(w) = $\sum_{i=1}^{n}(x_i^T w - y_i)^2 + \lambda \sum_{i=1}^{d} w_i^2$

$\dfrac{\partial f}{\partial w_1} = \dfrac{\partial \sum_{i=1}^{n}(x_i^T w - y_i)^2}{\partial (x_i^T w - y_i)} * \dfrac{\partial (x_i^T w - y_i)}{\partial w_1} + \lambda \dfrac{\partial \sum_{i=1}^{d}(w_i)^2}{\partial w_i} * \dfrac{\partial (w_i)}{\partial w_1}$

$\dfrac{\partial f}{\partial w_1} = 2 \sum_{i=1}^{n}(x_i^T w - y_i) * x_{i1} + 2\lambda \sum_{i=1}^{d} w_i$

Similarly,

$\dfrac{\partial f}{\partial w_2}$ =2 $\sum_{i=1}^{n}(x_i^T w - y_i) * x_{i2} + 2\lambda \sum_{i=1}^{d} w_i$

So,

$\nabla_w f = [\dfrac{\partial f}{\partial w_1}, \dfrac{\partial f}{\partial w_2}, \dfrac{\partial f}{\partial w_3}, ...\dfrac{\partial f}{\partial w_n}]^T$

$\nabla_w f = 2 \sum_{i=1}^{n}(x_i^T w - y_i) \sum_{i=1}^{d} w_i \; [x_{i1} + \lambda, x_{i2} + \lambda, ..., x_{in} + \lambda]^T$

6. (Chain-rule and softmax function) Calculate the gradient of f w.r.t. vector x= $(x_1, ..., x_n)^T$.Here, $f(x_1, x_2, ...x_n = \log \sum_{i=1}^{n} e^{x_i}$

We know that,

$\nabla_x f = [\dfrac{\partial f}{\partial x_1}, \dfrac{\partial f}{\partial x_2}, \dfrac{\partial f}{\partial x_3}, ...\dfrac{\partial f}{\partial x_n}]^T$

Using chain-rule:

$\dfrac{\partial f}{\partial x_1} = \dfrac{\partial log(e^{x_1} + e^{x_2} + ... + e^{x_n}}{\partial e^{x_1} + e^{x_2} + ... + e^{x_n}} * \dfrac{\partial e^{x_1} + e^{x_2} + ... + e^{x_n}}{\partial x_1}$

$\frac{\partial f}{\partial x_1} = \frac{1}{e^{x_1+x_2+x_3+...+x_n}} * e^{x_1}$
Similarly,
$\frac{\partial f}{\partial x_2} = \frac{\partial log(e^{x_1}+e^{x_2}+...+e^{x_n}}{\partial e^{x_2}+e^{x_2}+...+e^{x_n}} * \frac{\partial e^{x_2}+e^{x_2}+...+e^{x_n}}{\partial x_2}$
$\frac{\partial f}{\partial x_2} = \frac{1}{e^{x_1+x_2+x_3+...+x_n}} * e^{x_2}$
So,
$\nabla_x f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_2}, ...\frac{\partial f}{\partial x_n}]^T$
$\nabla_x f = \frac{1}{\sum_{i=1}^n e^{x_i}}[e^{x_1}, e^{x_2}, e^{x_3}, ..., e^{x_n}]^T$

7. (Simple mathematical proof) For the soft-max function in part (f). Prove that $max_i x_i \leq f(x_1, ..., x_n) \geq max_i(x_i) + logn$.Here, $f(x_1, x_2, ...x_n = \log \sum_{i=1}^n e^{x_i}$

   Taking: Lower bound i.e $max_i x_i \leq f(x_1, ..., x_n)$

   We know that,

   = loga $\leq$ logb if a $\leq$ b

   = $e^{x_i} \geq 0$

   = $e^{max_i x_i} \leq \sum_{i=1}^n e^{x_i}$ [As, max component of x is there along with other components of x, so only max will be smaller than the whole component sum.]

   Taking log on both sides:

   $= log(e^{max_i x_i}) \leq log(\sum_{i=1}^n e^{x_i})$

   $= max_i x_i \leq log(\sum_{i=1}^n e^{x_i})$

   $= max_i x_i \leq f(x_1, ..., x_n)$

   Taking upper bound: $f(x_1, ..., x_n) \geq max_i(x_i) + logn$

   We know that,

   $log(\sum_{i=1}^n e^{x_i}) = log(e^{max_i(x_i)}) + log \sum_{i=1}^n e^{x_i} - log(e^{max_i(x_i)})$

   $log(\sum_{i=1}^n e^{x_i}) = log(e^{max_i(x_i)}) + log(\frac{\sum_{i=1}^n e^{x_i}}{e^{max_i x_i}})$

   $log(\sum_{i=1}^n e^{x_i}) \leq max_i(x_i) + logn$

   $f(x_1, ..., x_n) \leq max_i(x_i) + logn$

   So, combining both we get:

   $max_i x_i \leq f(x_1, ..., x_n) \leq max_i(x_i) + logn$

problem2

February 13, 2024

[1]:
```
import numpy as np
import scipy
import os
from scipy.sparse import diags, dia_matrix
```

0.1 a. Matrix multiplication

[2]:
```
A =np.arange(1,21)
A = A.reshape((5,4))
# A = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20]])
print(A.shape, A)

B = np.arange(1,13)
B = B.reshape((3,4)).T
# B = np.array([[1,5,9],[2,6,10],[3,7,11],[4,8,12]])
print(B.shape, B)
```

```
(5, 4) [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
(4, 3) [[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]
```

[3]:
```
def mat_multiply(A,B):
    return np.dot(A,B)

print('Matrix multiplication',mat_multiply(A,B))
```

```
Matrix multiplication [[ 30  70 110]
 [ 70 174 278]
 [110 278 446]
 [150 382 614]
 [190 486 782]]
```

0.2 b. Sparse matrix-vector multiplication

[4]:
```
def generate_sparse_matrix(n):
    print("N is",n)
    vector_x = np.random.randint(5,size=(n)) # generates number in an array␣
 ↪from 0 to 5
    diagonals = [np.ones((n-1)), np.ones((n-1)) * -1, np.zeros((n-1))]
    sparse_mat_A = diags(diagonals, offsets=[0, 1,2], shape=(n-1,n)).toarray()
    return sparse_mat_A @ vector_x.T


print(generate_sparse_matrix(7))
```

```
N is 7
[-2.  2. -4.  2. -1.  0.]
```

The worst case time complexity (in Big O notation) of multiplying a matrix A of dimension Rn×n with a dense vector v Rn = O(n^2) = Mulitplication & addition operation

What if matrix A is sparse, denote the number of non-zero elements by nnz(A) = O(nnz(A) + n)

0.3 c. Manipulating text using python, data structures

[5]:
```
file_name = 'data_example.txt'

file_content_arr = []




def unique_words_with_count(file_name):
    current_directory = os.getcwd() # cost = 1, time = 1
    file_path = os.path.join(current_directory, file_name)
    dict_unique_words_with_count = {}

    try:
        with open(file_path, 'r') as file:
        # Read the entire contents of the file
            file_contents = file.read()
            file_content_arr = file_contents.lower().split(' ')
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

    # print(file_content_arr)
    for i in file_content_arr:
        if i in dict_unique_words_with_count:
            dict_unique_words_with_count[i] = dict_unique_words_with_count[i] +␣
 ↪1
```

```python
        else:
            dict_unique_words_with_count[i] = 1

    return dict_unique_words_with_count


print('Count of unique words', unique_words_with_count(file_name))
```

```
Count of unique words {'this': 5, 'film': 4, 'took': 1, 'me': 1, 'by': 1,
'surprise.': 1, 'i': 5, 'make': 2, 'it': 5, 'a': 7, 'habit': 1, 'of': 6,
'finding': 1, 'out': 1, 'as': 3, 'little': 1, 'possible': 1, 'about': 2,
'films': 1, 'before': 1, 'attending': 1, 'because': 1, 'trailers': 1, 'and': 10,
'reviews': 1, 'provide': 1, 'spoiler': 1, 'after': 1, 'spoiler.': 1, 'all': 1,
'knew': 2, 'upon': 1, 'entering': 1, 'the': 8, 'theater': 1, 'is': 4, 'that': 4,
'was': 5, 'documentary': 2, 'long': 2, 'married': 1, 'couple.': 1, 'filmmaker':
2, 'doug': 2, 'block': 1, 'decided': 1, 'to': 5, 'record': 1, 'his': 2,
'parents': 1, '"for': 1, 'posterity"': 1, 'at': 1, 'beginning': 2, 'we': 1,
'are': 1, 'treated': 1, 'requisite': 1, 'interviews': 1, 'with': 3, 'parents,':
1, 'outspoken': 1, 'mother': 1, 'mina,': 1, 'less': 1, 'than': 1, 'forthcoming':
1, 'dad,': 1, 'mike.': 1, 'immediately': 1, 'found': 1, 'couple': 1,
'interesting': 1, 'had': 1, 'no': 1, 'idea': 1, 'where': 2, '(mike': 1, '&': 1,
"mina's": 1, 'son': 1, 'doug)': 1, 'going': 2, 'take': 1, 'us.': 1, 'matter': 1,
'fact,': 1, 'doubt': 1, 'himself': 1, 'he': 1, 'this!': 1, 'life': 2, 'takes':
1, 'unexpected': 1, 'twists': 1, 'turns': 1, 'beautifully': 1, 'expressive': 1,
'follows': 1, 'journey.': 1, 'difficult': 1, 'verbalize': 1, 'just': 1, 'how':
1, 'moved': 1, 'story': 1, 'unique': 1, 'way': 1, 'in': 1, 'which': 1, 'told.':
1, 'absolutely': 1, 'riveting': 1, 'from': 1, 'end': 1, 'really': 1, 'must-see':
1, 'even': 2, 'if': 1, 'you': 3, "aren't": 1, 'fan': 1, 'genre.': 1, 'will': 1,
'think': 1, 'your': 1, 'own': 1, 'might': 1, 'evoke': 1, 'memories': 1,
'thought': 1, 'were': 1, 'forgotten.': 1, '"51': 1, 'birch': 1, 'street"': 1,
'one': 1, 'those': 1, 'rare': 1, 'filmgoing': 1, 'experiences': 1, 'makes': 1,
'deep': 1, 'impression': 1, 'never': 1, 'leaves': 1, 'you.': 1}
```

1. I have used array and dictionary to solve this problem. As, time complexity (is not time required to execute the function), the measure of runtime of an algorithm (number of basic operations involved, time it takes to execute the operation for a given input. suppose your function takes input n, and that function is simply print operations, no matter what the value of n is, it's always one operation & is fixed. But in case of loop with n, as the number of n increases, operations inside loop also increase.) will change as the input changes.

2. Time complexity of above function is O(n + m), where n = size of array and m = how big the file size is.
3. Space complexity is O(nm)

0.4 d. Recursion with bookkeeping in python

[6]:
```python
# [0,1,1,2,3,5,8,13,....]
def fibonacci(n):
    if n < 0:
        return 'Please, give positive number only'
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(38))
```

39088169

This will have n^2 complexity for time & space. O(n^2).

[7]:
```python
def fibonacci_reduced_complexity(n, dict ={}):
    if n in dict:
        return dict[n]
    if n < 0:
        return 'Please, give positive number only'
    elif n == 0:
        # dict[n] = 0
        return 0
    elif n == 1 or n == 2:
        dict[n] = 1
        return 1
    else:
        dict[n] = fibonacci_reduced_complexity(n-1, dict) + 
  ↪fibonacci_reduced_complexity(n-2, dict)
    return dict[n]

print(fibonacci_reduced_complexity(10))
```

55

1 This will have 2n complexity for time & space. O(2n) = O(n).

```
[ ]:
```