

# General Notes

CS, ML and Stats

Patrick Daly

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Computer Science</b>   | <b>4</b>  |
| 1.1      | Algorithms . . . . .  | 4         |
| 1.2      | Data Sctructures . . . . .  | 9         |
| 1.3      | Linux . . . . .   | 9         |
| <b>2</b> | <b>Machine Learning</b>   | <b>11</b> |
| 2.1      | Supervised . . . . .  | 11        |
| 2.1.1    | Ordinary Least Squares (OLS) . . . . .  | 11        |
| 2.1.2    | Generalized Linear Model(GLM) . . . . .   | 11        |
| 2.1.3    | Logistic Regression . . . . .   | 11        |
| 2.1.4    | Linear Discriminant Analysis . . . . .  | 11        |
| 2.1.5    | Support Vector Machines . . . . .   | 11        |
| 2.1.6    | K-Nearest Neighbors . . . . .   | 11        |
| 2.1.7    | Gaussian Process . . . . .  | 11        |
| 2.1.8    | K-Nearest Neighbors . . . . .   | 11        |
| 2.1.9    | Decision Trees . . . . .  | 11        |
| 2.1.10   | Random Forest . . . . .   | 11        |
| 2.1.11   | Gaussian Process . . . . .  | 11        |
| 2.1.12   | Naive Bayes . . . . .   | 11        |
| 2.1.13   | Kalman Filter? dunno where this should go yet...<br>maybe estimation section? . . . . . | 11        |
| 2.2      | Unsupervised . . . . .  | 11        |
| 2.2.1    | Gaussian Mixture Models . . . . .   | 11        |
| 2.2.2    | K-Means . . . . .   | 11        |
| 2.2.3    | Density-Based Spatial Clustering of Applications with<br>Noise (DBSCAN) . . . . .       | 11        |
| 2.2.4    | Spectral Clustering . . . . .   | 11        |
| 2.2.5    | Hierarchical Clustering . . . . .   | 11        |
| 2.2.6    | Factor Analysis . . . . .   | 11        |
| 2.2.7    | Independent Component Analysis . . . . .  | 11        |
| 2.2.8    | Principal Component Analysis . . . . .  | 11        |
| 2.2.9    | Non-Negative Matrix Factorization (NMF) . . . . .                                       | 11        |
| 2.2.10   | Latent Dirichlet Allocation (LDA) . . . . .   | 11        |
| 2.2.11   | Outlier Detection? . . . . .  | 11        |
| <b>3</b> | <b>Deep Learning</b>  | <b>11</b> |
| 3.1      | Convolutional Networks . . . . .  | 11        |
| 3.2      | Recurrent Networks . . . . .  | 11        |

|          |                                      |           |
|----------|--------------------------------------|-----------|
| 3.3      | Long Short-Term Memory (LSTM)        | 11        |
| 3.4      | Autoencoders                         | 11        |
| 3.5      | Reinforcement Learning               | 11        |
| <b>4</b> | <b>Linear Algebra</b>                | <b>11</b> |
| 4.1      | Norms                                | 11        |
| 4.1.1    | Euclidean / Frobenius                | 11        |
| 4.1.2    | Manhattan                            | 11        |
| 4.1.3    | Infinity                             | 11        |
| 4.1.4    | Nuclear                              | 11        |
| 4.1.5    | Spectral                             | 11        |
| 4.1.6    | Symmetric                            | 11        |
| 4.1.7    | Positive Definite                    | 11        |
| 4.1.8    | Positive Semi-Definite               | 11        |
| 4.1.9    | Negative Definite                    | 11        |
| 4.1.10   | Negative Semi-Definite               | 11        |
| 4.2      | Eigendecomposition                   | 11        |
| 4.3      | Singular Value Decomposition (SVD)   | 11        |
| 4.4      | Principal Component Analysis (PCA)   | 11        |
| 4.5      | Independent Component Analysis (ICA) | 11        |
| 4.6      | Canonical Component Analysis (CCA)   | 11        |
| 4.7      | Factor Analysis                      | 11        |
| <b>5</b> | <b>Statistics</b>                    | <b>11</b> |
| 5.1      | Probability Theory                   | 11        |
| 5.2      | Distributions                        | 11        |
| 5.3      | Combinatorics                        | 11        |

# 1 Computer Science

## 1.1 Algorithms

**DFS** Time:  $O(n)$ , Space:  $O(n)$   
Solution exists far away.

Recursive

```
1 def dfs(node):
2     if node:
3         # do stuff if pre-order
4         if node.left:
5             dfs(node.left)
6         # do stuff if in-order
7         if node.right:
8             dfs(node.right)
9         # do stuff if post-order
```

Iterative

```
1 def dfs(node): # if bst, may need to swap search left/right
2     visited = set()
3     stack = [node]
4     while stack:
5         current = stack.pop(-1)
6         print(current.val)
7         if current not in visited:
8             visited.add(current)
9             if current.left and current.left not in visited:
10                stack.append(current.left)
11            if current.right and current.right not in visited:
12                stack.append(current.right)
13     return visited
```

**BFS** Time:  $O(n)$ , Space:  $O(n)$   
Solution exists nearby.

Iterative

```
1 def bfs(node):
```

```

2     stack = [node]
3     while stack:
4         current = stack.pop(-1)
5         if current.left:
6             stack.append(current.left)
7         if current.right:
8             stack.append(current.right)

```

Mergesort Time:  $O(n \log n)$ , Space:  $O(n)$

```

1 def mergesort(array, start, end):
2     if start < end:
3         mid = (start+end) // 2
4         mergesort(array, start, mid)
5         mergesort(array, mid+1, end)
6         merge(array, start, mid, end)

1 def merge(array, start, mid, end):
2     left = array[start: mid+1]
3     right = array[mid+1: end+1]
4     i, j, k = 0, 0, start
5     while i < len(left) and j < len(right):
6         if left[i] < right[j]:
7             array[k] = left[i]
8             i += 1
9         else:
10            array[k] = right[j]
11            j += 1
12            k += 1
13    if j == len(right):
14        array[k: end+1] = left[i:]

1 def binarysearch(array, val, low, high):
2     if high < low:
3         # can also return high or low index
4         return 'Not found!'
5     mid = (low + high) // 2
6     if array[mid] > val:
7         return binarysearch(array, val, low, mid-1)

```

```

8     elif array[mid] < val:
9         return binarysearch(array, val, mid+1, high)
10    return mid

```

**Greatest Common Divisor (GCD)** Time: ?, Space: ?

Euclid: Time:  $O(\ln^2 \min(a, b))$ , Space:  $O(1)$

```

1 def gcd(a, b):
2     return gcd(b, a % b) if b else a

```

Stein: Time: 60% faster than Euclid in theory, similar in practice Space:  $O(1)$

```

1 def stein(a, b):
2     # binary gcd
3     if a == b: return a
4     if a == 0: return v
5     if b == 0: return a
6     if ~a & 1: # a is even
7         if b & 1: # b is odd
8             return stein(a >> 1, v)
9         else: # b is even
10            return stein(a >> 1, b >> 1) << 1
11    else:
12        if ~b & 1: # b is even
13            return stein(a, b >> 1)
14        else: # b is odd
15            if a > b:
16                return stein((a-b) >> 1, b)
17            else:
18                return stein(u, (b-a) >> 1)

```

**Fibonacci** Time:  $O(n)$ , Space:  $O(1)$

Iterative

```

1 def fib_iterative(n):
2     if n == 0: return 0
3     if n == 1: return 1
4     first, second, fibn = 0, 1, 0
5     for i in range(2, n+1):

```

```

6         fib_n = fib_n2 + fib_n1
7         fib_n = fib_n2
8         fib_n2 = fib_n
9     return fibn

```

Recursive

```

1 def fib_recursive(n, dp={0: 0, 1: 1}):
2     if n in dp:
3         return dp[n]
4     return fib_recursive(n-2, dp) + fib_recursive(n-1, dp)

```

**Longest Increasing Subsequence** Time:  $O(n \log n)$ , Space:  $O(n)$

```

1 def lis(nums):
2     tails = [0]*len(nums)
3     maxlen = 0
4     for num in nums:
5         start, end = 0, maxlen
6         while start != end:
7             mid = (start+end)//2
8             if tails[mid] < num:
9                 start = mid + 1
10            else:
11                end = mid
12            tails[start] = num
13            maxlen = max(maxlen, start+1)
14    return maxlen

```

**Topological Sort** Time:  $(V + E)$ , Space:  $O(V)$

Identifying a linear ordering of vertices such that if the graph  $G$  contains an edge  $(u, v)$ , then  $u$  appears before  $v$  in the ordering. Often used in identifying dependency graphs or sources in a event chain. Multiple solutions may exist.

```

1 def topo_sort(G):
2     def dfs(u):
3         visited.add(u)
4         for v in G[u]:
5             if v not in visited:
6                 dfs(v)

```

```

7         order.insert(0, u)
8     order = list()
9     visited = set()
10    for u in G:
11        if u not in visited:
12            dfs(u)
13    return order

```

**Strongly Connected Components** Time:  $O(V + E)$ , Space:  $O(V + E)$

Return all subgraphs for which each node is reachable from all other nodes in the subgraph. Tarjan and Pearce algorithms reduce the space requirement by 1-2x but they're more complicated to remember.

```

1 def transpose(G):
2     Gt = defaultdict(set)
3     for u in G:
4         for v in G[u]:
5
6 def scc_kosaraju(G):
7     def visit(u, add_to_stack):
8         visited[u] = True
9         if not add_to_stack:
10            sccs[-1].append(u)
11        for v in G[u]:
12            if not visited[v]:
13                visit(v, add_to_stack)
14        if add_to_stack:
15            stack.append(v)
16    sccs = [[]]
17    visited = dict.fromkeys(G.keys(), False)
18    stack = list()
19    for u in G:
20        if not visited[u]:
21            visit(u, add_to_stack=True)
22    G = transpose(G)
23    visited = dict.fromkeys(G.keys(), False)
24    while stack:
25        u = stack.pop()
26        if not visited[u]:
27            visit(u, add_to_stack=False)

```



```
28         if stack:
29             sccs.append([])
30     return sccs
```

## 1.2 Data Structures

## 1.3 Linux

### Common Commands

1. grep
2. awk
3. xargs
4. find
5. cut

### Software & Packages

1. Vim
2. Tmux
3. Ranger
4. Autojump
5. Tldr
6. Jq
7. Ccat



## 2 Machine Learning

### 2.1 Supervised

#### 2.1.1 Ordinary Least Squares (OLS)

#### 2.1.2 Generalized Linear Model(GLM)

#### 2.1.3 Logistic Regression

#### 2.1.4 Linear Discriminant Analysis

#### 2.1.5 Support Vector Machines

#### 2.1.6 K-Nearest Neighbors

#### 2.1.7 Gaussian Process

#### 2.1.8 K-Nearest Neighbors

#### 2.1.9 Decision Trees

#### 2.1.10 Random Forest

#### 2.1.11 Gaussian Process

#### 2.1.12 Naive Bayes

#### 2.1.13 Kalman Filter? dunno where this should go yet... maybe estimation section?

### 2.2 Unsupervised

#### 2.2.1 Gaussian Mixture Models

#### 2.2.2 K-Means

#### 2.2.3 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

#### 2.2.4 Spectral Clustering

#### 2.2.5 Hierarchical Clustering

#### 2.2.6 Factor Analysis

#### 2.2.7 Independent Component Analysis

#### 2.2.8 Principal Component Analysis

#### 2.2.9 Non-Negative Matrix Factorization (NMF)

#### 2.2.10 Latent Dirichlet Allocation (LDA)

#### 2.2.11 Outlier Detection?

11

## 3 Deep Learning

### 3.1 Convolutional Networks

### 3.2 Recurrent Networks

#### 3.2.1 Long Short-Term Memory (LSTM)