Machine Learning

- Artificial Intelligence (AI) & Machine Learning (ML) & Deep Learning (DL)
- AI ~ anything related to making computers do stuff that are traditionally done by humans; sorting, gaming, etc.
- ML ~ algorithms that learn models from data; neural nets, SVMs, etc.
- DL ~ application of multi-layer neural nets to learning tasks.
✓ How do we decide the best class?
✓ Experience the past (training) and decide accordingly (query).

Plot your experience.
✓ Mystery flower above (???) lands closer to reds, so decision: red.
Machine Learning using Neural Networks (NNs)

- Neural nets can do this classification for us w/o any plotting or such.

<table>
<thead>
<tr>
<th>color</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
<td>2</td>
<td>5.5</td>
<td>1</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1.5</td>
<td>1.5</td>
<td>1</td>
<td>.5</td>
<td>.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Feed the input (width, length) to our net (bottom) and get an output as their weighted combination (top). If closer to 1, net tells us it is red.
- Currently net is wrong (‘cos 2 & 1 from a blue flower). Adjust weights.
Weights of the NN

- Output is a weighted \((w_1\text{ and } w_2)\) combination of the input.

- Adjust these weights and the bias term \((b)\) to make your net behave the way you want.

- We want: respect the input-output pairs we provide (train with):
  
  - \(2 \& 1 \rightarrow\) Blue \((\text{so output } \leq 0.5)\), \(5.5 \& 1 \rightarrow\) Red \((\text{so output } > 0.5)\), etc.

<table>
<thead>
<tr>
<th>color</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
<td>2</td>
<td>5.5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1.5</td>
<td>1.5</td>
<td>1</td>
<td>.5</td>
<td>.5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Weights of the NN

✓ Squash the values to be in $[0,1]$: 

\[ \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

\[ \text{sigmoid}(w_1 \cdot m_1 + w_2 \cdot m_2 + b) \]
Weights of the NN

- Weights and biases start randomly (to be adjusted later).
  - \( w_1 = .5, w_2 = .2, b = .3 \) \( \Rightarrow \) \( \text{NN}(2,1) = \text{sigmoid}(1.5) = .8 \)
  - NN thinks it is red; we’d have preferred output to be closer to 0.
  - Solution: adjust weights & biases (via Backpropagation method).
    - Cost function: \((\text{prediction} - \text{target})^2 = (0.8 - 0)^2\)
    - Since prediction depends on weights & biases variables, take partial derivative w.r.t. those (gradient descent) and get the adjustment that minimizes the cost.
Weights of the NN

✓ Weights and biases start randomly (to be adjusted later).

<table>
<thead>
<tr>
<th>color</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>⬜️</th>
<th>????</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
<td>2</td>
<td>5.5</td>
<td>1</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1.5</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>.5</td>
<td>.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

✓ Once all weights & biases are adjusted based on the observed data, we essentially constructed our model (NN).

✓ Feed the parameters of the new flower to this constructed model in order to classify it instantly (and hopefully accurately).
Graph

✓ Graph: set of vertices and edges that model many problems in CS.

✓ Footballers (vertices) are connected (edges) if they played at the same team anytime in their careers.

✓ People are connected if they are friends, e.g., Facebook network.
Graph Coloring

- Assignment of colors to vertices s.t. neighbor verts’ve different colors.

- Use as few colors as possible (chromatic number).

- Why do we care?
Graph Coloring for Scheduling

- Series of taxi journeys with a start time (filled) and an end time (empty).
- A taxi cannot be used on another journey until it returns.
- 10 taxis obviously suffice to serve these requests but expensive 😞.
- Can do with just 3 😊.
- Graph: journey (vertex), overlapping in time (edge).
Graph Coloring for Scheduling

- Series of flights with a start time (filled) and an end time (empty).
- A gate cannot be used while occupied by a plane.
- 10 gates obviously suffice to serve these requests but expensive 😞.
- Can do with just 3 😊. Min # of gates for these flights is 3.
- Graph: flight (vertex), overlapping in time (edge).
Graph Coloring for Scheduling

- Schedule exams for courses.
- Two courses clash if some student taking them both.
- 9 timeslots obviously suffice to serve these requests but expensive 😞.
- Can do with just 4 😊.
- Graph: course (vertex), clashing (edge).
Graph Coloring for Scheduling

- Separate cages in a zoo.
- Two species may not get along together.
- Graph: animals (vertex), hating (edge). Min # of cages.
Graph Coloring for Scheduling

- Design seating plans for weddings.
- Some people do not want to seat together (drama).
- 9 tables obviously suffice to serve these requests but expensive 😞.
- Can do with just 2 🙂.
- Graph: party (vertex), hating each other (edge).
Graph Coloring for Sudoku

- Solve Sudoku puzzles.
- Fill in the blank cells s.t. each row, col, and 2x2 box has 1-4 just once.
- Graph: cell (vertex), same row, col, or box (edge).
- 4-coloring of this graph corresponds to a Sudoku solution.
- Some cells filled already (clues) = some vertices already colored for u.
Graph Coloring for Maps

- Color maps to separate neighboring regions robustly.
- Theorem: 4 colors suffice for all possible maps.
- Graph: region (vertex), neighboring (edge).

Suboptimal: 5 colors used.
Graph Coloring Algorithms

✓ So how to solve this problem?
✓ Exact solution: Check each of the $k^n$ assignments of $k$ colors to $n$ vertices for legality. Repeat for $k = 1, 2, \ldots, n-1$.
✓ Too slow 'cos this is a brute-force exponential solution.
  ✓ Growth-rates of functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>10</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\log_2 n$</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>$n$</td>
<td>$10^2$</td>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^5$</td>
<td>$10^6$</td>
<td></td>
</tr>
<tr>
<td>$n \cdot \log_2 n$</td>
<td>30</td>
<td>664</td>
<td>9,965</td>
<td>$10^5$</td>
<td>$10^6$</td>
<td>$10^7$</td>
</tr>
<tr>
<td>$n^2$</td>
<td>$10^2$</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>$10^8$</td>
<td>$10^{10}$</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>$n^3$</td>
<td>$10^3$</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^{12}$</td>
<td>$10^{15}$</td>
<td>$10^{18}$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$10^3$</td>
<td>$10^{30}$</td>
<td>$10^{301}$</td>
<td>$10^{3,010}$</td>
<td>$10^{30,103}$</td>
<td>$10^{301,030}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Instructions</th>
<th>Instr/Second</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^2)$ algorithm</td>
<td>$2n^2$</td>
<td>$10^9$</td>
<td>2000 secs.</td>
</tr>
<tr>
<td>$O(n \log n)$ algorithm</td>
<td>$100 n \log n$</td>
<td>$10^7$</td>
<td>60 secs.</td>
</tr>
</tbody>
</table>
✓ So how to solve this problem?
✓ Approximate solution: Based on heuristics. No optimality guarantees.
✓ This is where machine learning comes in.
  ✓ Some heuristics goods for some graphs.
  ✓ Train a neural network: input graph G1, output Heuristic2.
    input graph G7, output Heuristic1.
    ...
  input graph G166, output Heuristic 2.
  ...
✓ Given a query graph, decide the best heuristic for it and apply it.
Graph Coloring Algorithms

- So how to solve this problem?

- Previously, we used flower features (width, length) to flower color (red, blue) mappings to train our NN. Then, decided the color of a new query point using this NN.

- Now, use graph features* to preferred heuristic (H1, H2) maps to train the NN and decide the better heuristic for a new query graph.

* 13 features measured on graphs:

1. no. of nodes: \( n \)
2. no. of edges: \( m \)
3,4. ratio: \( \frac{n}{m}, \frac{m}{n} \)
5. density: \( \frac{\frac{m}{2}}{n \cdot (n-1)} \)

6-13. nodes degree statistics: min, max, mean, median, \( Q_{0.25} \), \( Q_{0.75} \), variation coefficient, entropy

** Heuristic preference: run both heuristics on each training graph and pick the one using fewer colors (or taking less time).
Graph Coloring Algorithms

- So how to solve this problem?
- Heuristic 1: order vertices arbitrarily $v_1, v_2, .., v_n$. You have available colors $c_1, c_2, .., c_n$.
- For $i=1$ to $n$: Color $v_i$ with the lowest legal color $c_j$ //make it optimal by calling this loop $n!$ times for each possible ordering. ($O(n)\to O(n!)$).
So how to solve this problem?

- **Heuristic 1:** order vertices arbitrarily $v_1, v_2, \ldots, v_n$. You have available colors $c_1, c_2, \ldots, c_n$.

  - For $i=1$ to $n$: Color $v_i$ with the lowest legal color $c_j$ //make it optimal by calling this loop $n!$ times for each possible ordering. ($O(n) \rightarrow O(n!)$).

- **Bad ordering:** left-right-down-left-right-down-.. $\rightarrow$ $n/2$ colors 😞
- **Good ordering:** left-down-left-down-..right-down-.. $\rightarrow$ 2 colors 😊

- Always optimal regardless of ordering (2 colors):
Graph Coloring Algorithms

- So how to solve this problem?
- Heuristic 1: order vertices arbitrarily \( v_1, v_2, \ldots, v_n \). You have available colors \( c_1, c_2, \ldots, c_n \).
- For \( i=1 \) to \( n \): Color \( v_i \) with the lowest legal color \( c_j \) //make it optimal by calling this loop \( n! \) times for each possible ordering. (\( O(n!) \rightarrow O(n!) \)).
- Upper bound on \# of colors to be used: \( d+1 \), if max degree is \( d \).
- Proof:
  - Basis: 1-vertex graph (max degree is \( d=0 \)) requires \( 0+1=1 \) color. Done.
  - Induction: Assume statement is True for all \( n \)-vertex graphs. Show also True for \( n+1 \)-vertex graphs. Here I show: \( v_1, v_2, v_3, \ldots, v_n, v_{n+1} \).
  - **Red subgraph** has \( n \) vertices and max degree \( \leq d \), so by induction it uses at most \( d+1 \) colors.
  - For \( v_{n+1} \), even if all its neighbors (at most \( d \) neighbors) have different colors (worst-case), pick the \((d+1)\)th color for \( v_{n+1} \). Done.
Graph Coloring Algorithms

✓ So how to solve this problem?
✓ Heuristic 2: Choose the uncolored vertex w/ the highest # of different neighbor colors and color it legally. Break ties by choosing the vertex w/ the highest degree.
✓ Behaves better than Heuristic 1 but still no optimality guarantees.
So how to solve this problem?

Heuristic 2: Choose the uncolored vertex with the highest number of different neighbor colors and color it legally. Break ties by choosing the vertex with the highest degree.

Behaves better than Heuristic 1 but still no optimality guarantees.

Always optimal regardless of ordering (3 colors, 2 colors):