Going nonparametric:
Nearest neighbor methods for regression and classification

Locality sensitive hashing for approximate NN search...cont’d
Complexity of brute-force search

Given a query point, scan through each point
- $O(N)$ distance computations per 1-NN query!
- $O(N \log k)$ per $k$-NN query!

What if $N$ is huge???
(and many queries)

Moving away from exact NN search

- Approximate neighbor finding...
  - Don’t find exact neighbor, but that’s okay for many applications

Out of millions of articles, do we need the closest article or just one that’s pretty similar?
Do we even fully trust our measure of similarity???

- Focus on methods that provide good probabilistic guarantees on approximation
### Using score for NN search

#### 2D Data

<table>
<thead>
<tr>
<th>x</th>
<th>Sign(Score)</th>
<th>Bin index</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 5]</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>[1, 3]</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>[3, 0]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Using score for NN search

- Candidate neighbors if Score(x) < 0
- Only search here for queries with Score(x) < 0
- Query point x

#### Using score for NN search

<table>
<thead>
<tr>
<th>Bin</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>List containing indices of datapoints:</td>
<td>{1, 2, 4, 7, ...}</td>
<td>{3, 5, 6, 8, ...}</td>
</tr>
</tbody>
</table>

**Note:**

- Candidate neighbors if Score(x) < 0
- Only search here for queries with Score(x) > 0
- Search for NN amongst this set
Three potential issues with simple approach

1. **Challenging to find good line**
2. **Poor quality solution:**
   - Points close together get split into separate bins
3. **Large computational cost:**
   - Bins might contain many points, so still searching over large set for each NN query

---

How to define the line?

**Crazy idea:**
Define line randomly!

```
1.0 #awesome - 1.5 #awful = 0
```
How bad can a random line be?

Goal: If \( x, y \) are close (according to cosine similarity), want binned values to be the same.

- Bins are the same
- If \( \theta_{xy} \) is small (\( x, y \) close), unlikely to be placed into separate bins

Three potential issues with simple approach

1. Challenging to find good line
2. Poor quality solution:
   - Points close together get split into separate bins
3. Large computational cost:
   - Bins might contain many points, so still searching over large set for each NN query

<table>
<thead>
<tr>
<th>Bin</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>List containing indices of datapoints:</td>
<td>{1, 2, 4, 7, ...}</td>
<td>{3, 5, 6, 8, ...}</td>
</tr>
</tbody>
</table>
Improving efficiency:
Reducing # points examined per query

Reducing search cost through more bins

Example:
line 1: score(x)<0 => 0
line 2: score(x)>0 => 1
line 3: score(x)<0 => 0
bin: [0 1 1]
Using score for NN search

<table>
<thead>
<tr>
<th>2D Data</th>
<th>Sign (Score_1)</th>
<th>Bin 1 index</th>
<th>Sign (Score_2)</th>
<th>Bin 2 index</th>
<th>Sign (Score_3)</th>
<th>Bin 3 index</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1 = [0, 5]</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>x_2 = [1, 3]</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>x_3 = [3, 0]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data indices:</td>
<td>(1,2)</td>
<td>--</td>
<td>(4,8,11)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>(3,5,6)</td>
</tr>
</tbody>
</table>

search for NN amongst this set

Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data indices:</td>
<td>(1,2)</td>
<td>--</td>
<td>(4,8,11)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>(3,5,6)</td>
</tr>
</tbody>
</table>

Query point here, but is NN? Not necessarily

Even worse than before...Each line can split pts. Sacrificing accuracy for speed
### Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th>Bin index</th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data indices</td>
<td>(1,2)</td>
<td>--</td>
<td>(4,8,11)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>(3,5,6)</td>
</tr>
</tbody>
</table>

- **Next closest bins** (flip 1 bit)

### Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th>Bin index</th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data indices</td>
<td>(1,2)</td>
<td>--</td>
<td>(4,8,11)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>(3,5,6)</td>
</tr>
</tbody>
</table>

- **Further bin** (flip 2 bits)
Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Data indices:</td>
<td>(1,2)</td>
<td>--</td>
<td>(4,8,11)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>(3,5,6)</td>
</tr>
</tbody>
</table>

Quality of retrieved NN can only improve with searching more bins

**Algorithm:**
Continue searching until computational budget is reached or quality of NN good enough

**LSH recap**

- Draw $h$ random lines
- Compute "score" for each point under each line and translate to binary index
- Use $h$-bit binary vector per data point as bin index
- Create hash table
- For each query point $x$, search $\text{bin}(x)$, then neighboring bins until time limit
Moving to higher dimensions $d$

Draw random planes

$$\text{Score}(x) = v_1 \#\text{awesome} + v_2 \#\text{awful} + v_3 \#\text{great}$$
Cost of binning points in d-dim

\[
\text{Score}(x) = \sum_{i=1}^{d} \begin{cases} 
  v_1^i & \text{#awesome} \\
  v_2^i & \text{#awful} \\
  v_3^i & \text{#great}
\end{cases}
\]

Per data point, need \(d\) multiplies to determine bin index per plane.

One-time cost offset if many queries of fixed dataset.

Summary for retrieval using nearest neighbors and locality sensitive hashing.
What you can do now...

- Implement nearest neighbor search for retrieval tasks
- Contrast document representations (e.g., raw word counts, tf-idf,...)
  - Emphasize important words using tf-idf
- Contrast methods for measuring similarity between two documents
  - Euclidean vs. weighted Euclidean
  - Cosine similarity vs. similarity via unnormalized inner product
- Describe complexity of brute force search
- Implement LSH for approximate nearest neighbor search

Motivating NN methods for regression:
Fit globally vs. fit locally
Parametric models of $f(x)$

\[ y \text{ vs. price ($)} \]

\[ x \text{ vs. sq.ft.} \]
Parametric models of $f(x)$

$y$ vs. $x$

Parametric models of $f(x)$

$y$ vs. $x$
f(x) is not really a polynomial

What alternative do we have?

If we:
- Want to allow flexibility in f(x) having local structure
- Don’t want to infer “structural breaks”

What’s a simple option we have?
- Assuming we have plenty of data...
Simplest approach: Nearest neighbor *regression*

Fit locally to each data point

Predicted value = “closest” $y_i$

1 nearest neighbor (1-NN) regression
What people do naturally...

Real estate agent assesses value by finding sale of most similar house

![House images with different prices]

$ = ??? \quad $ = 850k

1-NN regression more formally

Dataset of (house, $) pairs: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

Query point: $x_q$

1. Find “closest” $x_i$ in dataset

   $$x_{NN} = \min_j \text{distance}(x_j, x_q)$$

2. Predict

   $$\hat{y}_q = y_{NN}$$

   Sales price of pink house

Here, this is the closest datapoint

Here, this is the closest datapoint

Here, this is the closest datapoint

Here, this is the closest datapoint

Here, this is the closest datapoint

Here, this is the closest datapoint

Here, this is the closest datapoint
Visualizing 1-NN in multiple dimensions

**Voronoi tessellation (or diagram):**
- Divide space into N regions, each containing 1 datapoint
- Defined such that any $x$ in region is “closest” to region’s datapoint

Don’t explicitly form!

Different distance metrics lead to different predictive surfaces

- Euclidean distance
- Manhattan distance
Can 1-NN be used for classification?

Yes!!
Just predict class of neighbor

1-NN algorithm
Performing 1-NN search

- Query house:
- Dataset:
- Specify: Distance metric
- Output: Most similar house

1-NN algorithm

Initialize Dist2NN = \( \infty \), \( \hat{q} = \emptyset \)

For \( i = 1, 2, \ldots, N \)

Compute: \( \delta = \text{distance}(\hat{q}_i, q) \)

If \( \delta < \text{Dist2NN} \)

set \( \hat{q} = \hat{q}_i \)

set \( \text{Dist2NN} = \delta \)

Return most similar house

©2018 Emily Fox
STAT/CSE 416: Intro to Machine Learning
1-NN in practice

Fit looks good for data dense in x and low noise

Sensitive to regions with little data

Not great at interpolating over large regions...
Also sensitive to noise in data

Fits can look quite wild... Overfitting?

k-Nearest neighbors
Get more “comps”

More reliable estimate if you base estimate off of a larger set of comparable homes

$k$-NN regression more formally

Dataset of (house, $\$) pairs: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

Query point: $x_q$

1. Find $k$ closest $x_i$ in dataset

$\{x_{\text{NN}_1}, \ldots, x_{\text{NN}_k}\}$ such that for any $x_i$ not in nearest $k$-nbr set, distance $(x_i, x_q) \geq \text{distance}(x_{\text{NN}_k}, x_q)$

2. Predict

$$\hat{y}_q = \frac{1}{k} \sum_{j=1}^{k} y_{\text{NN}_j}$$
Performing k-NN search

- **Query house:**

- **Dataset:**

- **Specify:** Distance metric
- **Output:** Most similar houses

---

**k-NN algorithm**

Initialize \( \text{Dist2kNN} = \text{sort}(\delta_1, \ldots, \delta_k) \)

For \( i = k+1, \ldots, N \)

Compute: \( \delta = \text{distance}(\text{house}_i, \text{query house}_q) \)

If \( \delta < \text{Dist2kNN}[k] \)

find \( j \) such that \( \delta > \text{Dist2kNN}[j-1] \) but \( \delta < \text{Dist2kNN}[j] \)

remove furthest house and shift queue:

\[
\text{Dist2kNN}[j+1:k] = \text{Dist2kNN}[j:k-1]
\]

set \( \text{Dist2kNN}[j] = \delta \) and \( \text{closest houses to query house} = \text{house}_i \)

Return \( k \) most similar houses
k-NN in practice

Nearest Neighbors Kernel (K = 30)

- Nearest neighbor (for \( K = 30 \)) of \( x_0 \)
- Current pt I’m trying to fit \( f(x_0) = \frac{1}{k} \sum_{j=1}^{K} y_j \)
- Much more reasonable fit in the presence of noise

Boundary & sparse region issues
k-NN in practice

Discontinuities! Neighbor either in or out

Issues with discontinuities

Overall predictive accuracy might be okay, but...

For example, in housing application:
  - If you are a buyer or seller, this matters
  - Can be a jump in estimated value of house going just from 2640 sq.ft. to 2641 sq.ft.
  - Don’t really believe this type of fit
Weighted k-nearest neighbors

Weigh more similar houses more than those less similar in list of k-NN

Predict:

\[ \hat{y}_q = \frac{c_{q_{NN1}}y_{NN1} + c_{q_{NN2}}y_{NN2} + c_{q_{NN3}}y_{NN3} + \ldots + c_{q_{NNk}}y_{NNk}}{\sum_{j=1}^{k} c_{q_{NNj}}} \]
How to define weights?

Want weight $c_{qNNj}$ to be small when

\[ \text{distance}(x_{NNj}, x_q) \text{ large} \]

and $c_{qNNj}$ to be large when

\[ \text{distance}(x_{NNj}, x_q) \text{ small} \]

Define:

\[ c_{qNNj} = \text{Kernel}(|x_{NNj} - x_q|) \]

**Simple method:**

\[ c_{qNNj} = \frac{1}{\text{distance}(x_{NNj}, x_q)} \]

**Kernel weights for d=1**

Define: $c_{qNNj} = \text{Kernel}(|x_{NNj} - x_q|)$

**Gaussian kernel:**

\[ \text{Kernel}_\lambda(|x_i - x_q|) = \exp(-|x_i - x_q|^2/\lambda) \]

**Note:** never exactly 0!
Kernel weights for $d \geq 1$

Define: $c_{qNNj} = \text{Kernel}_\lambda(\text{distance}(x_{NNj}, x_q))$

Kernel regression
**Weighted k-NN**

Weigh more similar houses more than those less similar in list of k-NN

Predict:

\[
\hat{y}_q = \frac{c_{qNN1}y_{NN1} + c_{qNN2}y_{NN2} + c_{qNN3}y_{NN3} + \ldots + c_{qNNk}y_{NNk}}{\sum_{j=1}^{k} c_{qNNj}}
\]

**Kernel regression**

Instead of just weighting NN, weight all points

Predict:

\[
\hat{y}_q = \frac{\sum_{i=1}^{N} c_{qi}y_i}{\sum_{i=1}^{N} c_{qi}} = \frac{\sum_{i=1}^{N} \text{Kernel}_\lambda(\text{distance}(x_i, x_q)) * y_i}{\sum_{i=1}^{N} \text{Kernel}_\lambda(\text{distance}(x_i, x_q))}
\]

Nadaraya-Watson kernel weighted average
Kernel regression in practice

Kernel has bounded support...
Only subset of data needed to compute local fit

Choice of bandwidth $\lambda$

Often, choice of kernel matters much less than choice of $\lambda$

$$\lambda = 0.04$$

$$\lambda = 0.2$$

$$\lambda = 0.4$$

Boxcar kernel
Choosing $\lambda$ (or $k$ in $k$-NN)

How to choose? Same story as always...

Cross Validation