**k Nearest Neighbors and Feature Scaling**

**Nearest Neighbors Algorithm**

- Store all the training data as feature vectors.
- Prediction for new, test data point: return the label of the closest training point.

*(you are the company you keep…)*

What is the predicted color for a new point (-2, -2)? Or for (2, 2)?

**k Nearest Neighbors Algorithm**

- Choose some integer value of $k$ (say, 3).
- Compute the $k$ closest training points to the test data point.
- Return the majority label.

What is the predicted color for a new point (-1.1, 1.7)?

**Effect of increasing $k$: smoother decision boundaries**
Choosing $k$

- $k$ is a free "hyperparameter" of the algorithm. How do we choose it?
- One option: try different values of $k$ when evaluating on test data
- Rather than split data into two parts, training and test, we split data into three parts, training and validation and test.
  - Use the validation data as "pseudo-test data" to tune (choose best) $k$
  - Do final evaluation on the test data only once

Distance Measure in 2D

$$distance(Point 1, Point 2) = \sqrt{(3.8 - 2.6)^2 + (5.4 - 2.6)^2}$$
Distance Measure in 2D - $L^2$ Norm

Point 1: 3.8, 5.4
Point 2: 2.6, 2.6
Point 3: 3.1, 1.5
Point 4: 2.1, 0.5

$$\text{distance}(\text{Point } a, \text{ Point } b) = \sqrt{|a_1 - b_1|^2 + |a_2 - b_2|^2}$$

Distance Measure in 2D - $L^1$ Norm

$\text{distance}(\text{Point } a, \text{ Point } b) = |a_1 - b_1| + |a_2 - b_2|$
Distance Measure in 2D - $L^\infty$ Norm

Point 1 | 3.8 | 5.4
Point 2 | 2.6 | 2.6
Point 3 | 3.1 | 1.5
Point 4 | 2.1 | 0.5

$\text{distance}(\text{Point } a, \text{Point } b) = \sqrt{|a_1 - b_1|^\infty + |a_2 - b_2|^\infty} = \max\{|a_1 - b_1|, |a_2 - b_2|\}$

Distance Measure in 3D

Point 1 | 3.8 | 5.4 | 4.7
Point 2 | 2.6 | 2.6 | 2.6
Point 3 | 3.1 | 1.5 | 2.2
Point 4 | 2.1 | 0.5 | 1.2

$\text{distance}(\text{Point 1, Point 2}) = \sqrt{(3.8 - 2.6)^2 + (5.4 - 2.6)^2 + (4.7 - 2.6)^2}$
**Distance Measure in 3D**

Distance measure in 3D is given by:

\[ \text{distance}(\text{Point } a, \text{Point } b) = \sqrt{|a_1 - b_1|^2 + |a_2 - b_2|^2 + |a_3 - b_3|^2} \]

**Distance Measure in High Dimensions**

Distance measure in high dimensions is given by:

\[ \text{distance}(\text{Point } a, \text{Point } b) = \sqrt{\sum_{i=1}^{d} |a_i - b_i|^2} \]

**kNN Complexity**

- Given \( n \) training examples and \( d \) features
- **Training** step
  - Time: approximately zero; just store the data points
  - Space: size of training data (\( n \times d \))
- **Testing** step (for each test example)
  - Time?

**Feature Scaling**

<table>
<thead>
<tr>
<th>Student</th>
<th>GPA</th>
<th>Standardized Test Score</th>
<th>Accept?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>0.0</td>
<td>1110</td>
<td>0</td>
</tr>
<tr>
<td>Student 2</td>
<td>3.8</td>
<td>1500</td>
<td>1</td>
</tr>
<tr>
<td>Student 3</td>
<td>3.9</td>
<td>1300</td>
<td>?</td>
</tr>
</tbody>
</table>
Feature Scaling

- Compute the mean (i.e., average) for each of the features in the training data and subtract this mean from each feature value.

For each of the $1 \leq i \leq n$ training examples and $1 \leq j \leq d$ features, we subtract the mean: $x_{ij} = x_{ij} - \mu_j$

where the mean of the $j^{th}$ feature is $\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$

- Data will then be centered around zero.

\[ \text{distance}(\text{Student 1, Student 3}) = \sqrt{[0.0 - 3.9]^2 + [1110 - 1300]^2} \]
\[ = \sqrt{15.21 + 36100} \]

\[ \text{distance}(\text{Student 2, Student 3}) = \sqrt{[3.8 - 3.9]^2 + [1500 - 1300]^2} \]
\[ = \sqrt{0.01 + 40000} \]
Feature Scaling

- Compute the standard deviation for each of the features in the training data and divide each feature value by this standard deviation.

For each of the $1 \leq i \leq n$ training examples and $1 \leq j \leq d$ features, we divide by the standard deviation:

$$ x_{ij} = \frac{x_{ij}}{\sigma_j} $$

where the standard deviation of the $j^{th}$ feature is

$$ \sigma_j = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \mu_j)^2} $$

- Data will then have comparable scale.

Feature Scaling - Test Data

- When scaling the training data, we store the mean and standard deviation values that we compute for each feature as part of the scaling process.

- When given a testing example, we need to make sure that it is on a comparable scale as the training data. Thus, we scale it using the stored mean and standard deviation values.

For the $i^{th}$ testing example, we scale each of its $1 \leq j \leq d$ features by subtracting the $j^{th}$ mean ($\mu_j$) and dividing by the $j^{th}$ standard deviation ($\sigma_j$):

$$ x_{ij} = \frac{(x_{ij} - \mu_j)}{\sigma_j} $$

Pros and Cons of kNN

**Pros**

- Simple and intuitive
- Can be used with multiple classes (not just 2)
- Data do not have to be linearly separable

**Cons**

- Need to store large full training data
- Test time is SLOOOOWW
  - Prefer to pay for expensive training in exchange for fast prediction
Looking ahead

- kNN is an instance-based classifier: must carry around training data (waste of space)
- Training easy
- Testing hard

Future methods will be

- Parametric classifiers: compute a small "model" and then throw away training data
- Training hard
- Testing easy

Looking ahead: linear classifiers

- **Training:** find a dividing "hyperplane" between two classes
- **Testing:** check which side of hyperplane the new point falls in

Overview

- **ML Algorithms**
  - Supervised Learning
  - Unsupervised Learning
  - Non-Parametric
    - Decision Trees
    - kNN
    - Support Vector Machines
    - Collaborative Filtering
  - Parametric
    - Regression Models
    - Linear Classifiers
    - Non-Linear Classifiers
    - Neural Networks
    - Hidden Markov Models
  - Non-Parametric
    - Gaussian Mixture Models
    - Dimensionality Reduction