Machine Learning

Machine learning is the study of writing programs that automatically learn to make useful inferences from implicit structure and patterns in data.

As an example (use this example throughout the recitation), we may be given data of a number of different people, with the following information:

[height (ft), age (yrs), gender, weight (lb), eye color, speaks English]

Person 1: [6.2, 24, M, 190, Blue, Yes]
Person 2: [5.3, 28, F, 120, Brown, Yes]
Person 1000: [5.5, 54, M, 120, Brown, No]

These examples are referred to as training data, a term we’ve already seen in the context of curve fitting. Each example (person) is represented by a feature vector, an ordered list of information where each element describes a different characteristic. Note that feature vectors are almost always an incomplete representation - in this case, it isn’t possible to describe every single aspect of a human.

There are two broad subclasses of machine learning algorithms that we can use to learn things from this data:

1. **Supervised learning**: A class of problems where each data example comes with an output value or label. The goal is to be able to predict the label of a future example given the feature vector (maybe use the Person example to make this clear)
   
   a. If we are given whether each person is American (label = 1) or not (label = -1), we can try to learn a model of what features are characteristic of being American. This is referred to as classification, b/c we are learning a model that tries to separate those examples that are American from those that aren’t.
b. If we are given the number of hours of exercise each person does per week, we could learn a model that will predict the # of exercise hours of a new person given the features. This is called **regression**, a type of problem we’ve already seen before in this class (curve fitting). Put simply, the output variable in regression takes on continuous values while the output variable in classification is one of a discrete set of class labels.

2. **Unsupervised learning**: Sometimes data points are unlabeled and we want to try and find regularities (or anomalies) in the data. Clustering is an important technique for doing this.

*Extra note*: There is also a subfield of machine learning in which you have some labeled and some unlabeled examples (semi-supervised learning).

The three components that are generally in all machine learning work:
1. Data representation (feature vectors)
2. Metric for assessing goodness of the model
3. Optimization method for learning the model

**Clustering**

Goal of clustering: Automatically segment data into groups of similar points.

When and why would we want to do this?
1. Organizing data
2. Understanding hidden structure in some data
3. Representing high-dimensional data in a low dimensional space. This may not be as intuitive. The idea is that once you find clusters, each data point can then be described by just the cluster it belongs to instead of by its representation in the feature space.

Examples:
1. Animals based on physical features
2. Genes based on expression profile
3. Facebook users by interests

Good clusters have:
- low intra-cluster dissimilarity (points within the same cluster are close together or similar)
- high inter-cluster dissimilarity (points from different clusters are far apart)
Let’s now look at clustering as an optimization problem, where we want to minimize/maximize an objective function.

For each cluster \( c \): \( \text{variance}(c) = \sum_{x \in c} (\text{mean}(c) - x)^2 \)

Lower variance is better - means that the points in the cluster are all about the same (close to average).

For a cluster set \( S \): \( \text{badness}(S) = \sum_{c \in S} \text{variance}(c) \)

badness\( (S) \), which mentions intra-cluster dissimilarity, should be as low as possible. For the trivial case in which each point is its own cluster, \( \text{badness}(S) = 0 \). That doesn't help us at all, so we generally add constraints, such as:

1. Set a maximum # of clusters \( K \)
2. Set a maximum distance between clusters

But computing the optimal solution is computationally intensive! Instead, we will resort to greedy algorithms, such as the hierarchical and k-means clustering algorithms. We will cover k-means in a couple of weeks and we will focus on hierarchical clustering for this recitation.

**Hierarchical Clustering**

Given a set of \( N \) data points, first compute an \( N \times N \) similarity matrix based on “distance”. Then:

1. Assign each item to a separate cluster \( (N \text{ total clusters} - \text{we will eventually get down to } K) \)
2. Find the two most similar clusters. This takes \( O(N^2) \) time, since you have to look at every pair. Evaluate similarity using distance metric (below).
3. Merge the two clusters you found in step 2. Now you have one fewer cluster than before.
4. Repeat steps 2-3 until termination (look at note 2 below for termination criteria).

How fast is the hierarchical clustering algorithm? Steps 2-3 take O(N^2) time, and will need to be repeated ~O(N) times. This means the overall complexity is O(N^3), which can be very slow for large datasets.

Why is the algorithm greedy?
Answer: Because in each step, we find the closest two points and merge them.

Notes:
1. Depending on how big N is, the N x N matrix can be quite large and may not fit easily in memory. Either another algorithm (e.g. k-means) should be used or you can’t precompute the entire matrix at once.
2. What are reasonable termination strategies? One is to specify the # of clusters K beforehand. Another is to stop the process when the next smallest distance between clusters is above a prespecified threshold (can show on dendrogram).

Computing distances between clusters
There are different definitions of “distance” between clusters that we can use for hierarchical clustering:

1. single_linkage (aka minimum linkage): shortest distance between any point in cluster A to any point in cluster B.
   - Good local coherence, meaning two data points that are close together will likely be in the same cluster. This may lead to the chaining phenomenon, i.e. the tendency to create one mega-cluster.

2. complete_linkage (aka maximum linkage): maximum/longest distance between any point in A to any point in B
   - Good global cluster quality...it makes sure that two points that are far apart won’t be in the same cluster

3. average_linkage: average distance between any point in A to any point in B

Example: Clustering People by Height

We have 6 people with the following heights in ft. :
5, 7, 5.1, 5.3, 6, 6.4

Here is a distance matrix, with the differences in height:

<table>
<thead>
<tr>
<th>People</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.1</td>
<td>0.3</td>
<td>1</td>
<td>1.4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.9</td>
<td>1.7</td>
<td>1</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>
Let’s use the single-linkage criterion. We can visualize the clustering process using a diagram called the dendrogram:

![Dendrogram Diagram](image)

The y-axis is the distance between clusters, and the x-axis shows each example. The dendrogram shows the progression of merges, and you can easily tell which elements belong to the same set.

If we set the threshold at 0.5, we will have 3 clusters.
If we set the threshold at 0.3, we will have 4 clusters.

**Multiple Features and Scaling**
So far, we’ve assumed a one-dimensional feature of distance, i.e. just difference in height. But what if we want to cluster on multiple features, e.g. height and weight?
We build a feature vector \(<\text{ftr1}, \text{ftr2}, \text{ftr3}, \ldots>\) for each person, and use the Minkowski metric to calculate distance:

\[
dist(X_1, X_2, p) = \left( \sum_{k=1}^{\text{dim}(X_1)} \left| X_{1_k} - X_{2_k} \right|^p \right)^{\frac{1}{p}}
\]

Usually \(p = 1\) (Manhattan) or 2 (Euclidean).

But features may have different units, with very different ranges (e.g. height (ft) and weight (lbs))! Two people who differ in weight by 10 lbs will sway the distance more than two people who differ in height by 1 ft, which is probably not right.

We need to scale the features to the same range so that distance is equally affected by each of them. Usually, this means normalizing each feature to a mean of 0 and stddev of 1:

```python
def scaleFeatures(vals):
    result = pylab.array(vals)
    mean = sum(result)/float(len(result))
    result = result -mean
    sd= stdDev(result)
    result = result/sd
    return result
```