Announcements

- Homework 9 is available, Due March 11.
- Course grade based on top 8 of 9 homeworks. All weighted equally.
- Remaining lectures
  - Optimization to linear programming and beyond

Dimension Reduction

- A key problem in working with large scale data sets
  - Can we reduce representation size at the expense of a small error

Warmup for dimension reduction for $\mathbb{R}^n$

- Consider the distance function $D(x,y) = 0$ if $x = y$, $D(x,y) = 1$ if $x \neq y$
- Suppose we have a domain $U$ and want to answer distance queries between a set of $n$ elements
- Natural solution is to use $\log_2 |U|$ bits to describe the elements
- Can we use less space if we want to approximately answer distance queries

Scenario: We have a database of objects (e.g. people), and a field with domain $U$ (e.g. favorite movie) – we are interested in reducing the space required to store the field info and to answer the query of whether or not $X$ and $Y$ have the same favorite movie.

Of course this is going to be hashing

- Choose a good hash function $h: U \rightarrow 2^{2^2}$
- Let $f_i(x) = h(x) \mod 2$
- 1 bit representation
  - If $x = y$, then $f_i(x) = f_i(y)$
  - If $x \neq y$, then $\Pr[f_i(x) = f_i(y)] \leq 1/2$
  - Property preserved with probability at least 50%
  - Repeat with $k$ independent has functions $h_1, \ldots, h_k$
    - If $x = y$, then $f_i(x) = f_i(y)$ for all $i = 1, \ldots, k$
    - If $x \neq y$, then $\Pr[f_i(x) = f_i(y) \text{ for all } i = 1, \ldots, k] \leq 2^-k$
- To achieve error of $\delta$, we need to use $k = \lceil \log_2 1/\delta \rceil$

Random Projections for $L_2$ Distance in $\mathbb{R}^n$

- Johnson-Lindenstrauss Transform
  - Extensions of Lipschitz mappings into a Hilbert space, William Johnson and Joram Lindenstrauss, 1984
  - Pure mathematics result that crossed over to Computer Science
  - Project from $\mathbb{R}^n$ to a random $\mathbb{R}^K$ dimensional subspace where $K$ is $O(\varepsilon^{-2} \log N)$ and distances are preserved to a factor of $1 + \varepsilon$
  - In practice, $K = 100$
Projections

• Projection onto a line
• Inner Product
• If \( b \) is a unit vector than \( a \cdot b \) gives the position of \( a \) when projected onto \( b \)
• Project \([4, -1, 3]\) onto \([2, 1, 1]\)

Gaussian Distribution

• \( N(\mu, \sigma^2) \) - Normal distribution with mean \( \mu \) and variance \( \sigma^2 \)
• \( X_1 \) and \( X_2 \) are independent random variables with distribution \( N(\mu_1, \sigma_1^2) \) and \( N(\mu_2, \sigma_2^2) \) then \( X_1 + X_2 \) has distribution \( N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \)

Aside: Generating Gaussians

• Box-Muller methods generates a pair of independent Gaussian RVs from two random points from \([0,1)\)
• Mapping of unit square to independent Gaussians
• Many languages have a Gaussian generator (Matlab, Python, Java)
• Web sites will generate them for you

Random Projection

• Objects are points in \( n \) dimensional Euclidean space \( \mathbb{R}^n \)
• Choose random vector \( r = (r_1, \ldots, r_n) \in \mathbb{R}^n \)
• Real valued function \( f_r : \mathbb{R}^n \rightarrow \mathbb{R} \)
• Random linear combination of components of \( x \)

Unbiased Estimator of Squared L_2 Distance

• For \( x, y \) in \( \mathbb{R}^n \), \( f_r((x)-f_r(y))^2 \) is an unbiased estimator of \( |x - y|^2 \)
• Fix \( x, y \) in \( \mathbb{R}^n \)
• \( f_r(x) - f_r(y) = \sum_{j=1}^{n} x_j r_j - \sum_{j=1}^{n} y_j r_j = \sum_{j=1}^{n} (x_j - y_j) r_j \)
• \( r_j \) is a Gaussian with mean zero and variance 1, \( (x_j), (y_j), (r_j) \) is a Gaussian with mean zero and variance \( |x_j - y_j|^2 \)
• Right-hand side is a Gaussian with mean zero and variance \( \sum_{j=1}^{n} (x_j - y_j)^2 = |x - y|^2 \)

Cont.

• By definition \( \text{Var}(X) = E[(X - E(X))^2] \), so \( \text{Var}(X) = E(X^2) \) when \( E(X) = 0 \)
• Taking \( X \) as the random variable \( f_r(x) - f_r(y) \)
• \( E[(f_r(x) - f_r(y))^2] = |x - y|^2 \)
• Estimator of the squared L_2 distance between \( x \) and \( y \)
Independent Trials

• Pick d vectors \(r_1, \ldots, r_d\).
• Each vector is drawn i.i.d. from a standard Gaussian.
• Given points \(x, y\), we get d independent estimates of \(|x - y|^2\).

“One can figure out exactly how large d needs to be to achieve a target approximation.”

For a set of k points in n dimensions, to preserve all \(k(k-1)/2\) interpoint Euclidean distances up to a \(1 \pm \varepsilon\) factor, set \(d = \Theta(\varepsilon^{-2} \log k)\).

Johnson-Lindenstrauss Transform

• JL transform maps from \(\mathbb{R}^n\) to \(\mathbb{R}^d\), where d is selected based on desired accuracy.
• The JL transform is represented as a \(d \times n\) matrix \(A\) where each of the \(dn\) entries is chosen i.i.d. from a standard Gaussian distribution.
• Mapping from \(n\) vectors to \(d\) vectors is defined \(x \rightarrow Ax / \sqrt{d}\).
• \(1/\sqrt{d}\) factor scales to be an average over \(d\) estimates.

Simplification

• D. Achlioptas (2003). Similar results hold if matrix entries are chosen uniformly from \((-1,1)\) or from \((-\sqrt{3}, 0, \sqrt{3})\) with probability 1/6, 2/3, 1/6 respectively.
• Proof:

Applications

• High dimensional data sets
• Facebook
• Friend neighborhood
• Stories followed
• Biochemistry
• Candidate compounds for pharmaceuticals
• Youtube
• Videos watched
• Phone metadata
• Numbers called

Range queries

• Queries such as “k-Nearest Neighbors”, all points within distance \(B\), count points within distance \(B\).
• Standard approach – reduce dimension and search using k-D trees*
• Results subject to errors by factors of \(1 \pm \varepsilon\).
• k-Nearest Neighbors
  • Given query point \(y\), return \(k\) points within \(1+\varepsilon\)B of \(y\), where \(B\) is the k-NN distance from \(y\).
  • Points with distance \(B\)
    • Given query point \(y\), return a set of points which contains all points within distance \((1-\varepsilon)B\) of \(y\) and no point of distance greater than \((1+\varepsilon)B\) of \(y\).

k-means clustering

• Given \(S\), a set of \(n\) points in \(\mathbb{R}^m\), find \(k\) representative points in \(\mathbb{R}^m\) that that best partition the data into clusters.
• Application – use these points for classification – a new point finds its nearest neighbor among the \(k\) points. Rocchio Algorithm.
• Partition the space into Voronoi cells.
• k-clustering of \(S\) is a partition into \(k\) subsets.
  • The cluster-variance is the sum of the squares of the distances of each point to the respective center of its cluster.
  • The k-means clustering of a set \(S\) is the k-clustering that minimizes the cluster-variance.
• Finding the optimal k-means cluster is NP-Hard, but we will ignore that.
Higher dimensional clustering

- It gets harder to draw!
- Same idea works in high dimensions
- Results show that k-means clustering can be done after dimension reduction, greatly improving performance of constructing and using clustering (at the expense of 1+ε error)
- Heuristic algorithms are used to construct a k-means clustering (with common confusion between the algorithm and the definition of clustering)

Lloyd’s Algorithm (Stuart Lloyd, Bell Labs, 1957)

- Iterative algorithm that (usually) converges to a good approximation.
- Pick an initial clustering
- Repeat until tired
  - Compute centers of clusters
  - Reassign points to closest center

Lloyd’s Algorithm

```c
for (int i = 0; i < N; i++) {
    CS.AddPoint(i, i%K);
}
```

```c
int count = 0;
while (count < TOO_LONG) {
    CS.SetCenters();
    bool moved = false;
    for (int i = 0; i < N; i++) {
        int g = CS.Group(i);
        double d_min = dist(P[i], CS.Center(g));
        for (j = 0; j < K; j++) {
            if (j == CS.Group(i)) continue;
            double d = dist(P[i], CS.Center(g));
            if (d < d_min) {
                g = j;
                d_min = d;
            }
        }
        if (g != CS.Group(i)) {
            CS.MovePoint(i, g);
            moved = true;
        }
    }
    count++;
    if (!moved) break;
}
```

CS is a ClusterSet which associates points with clusters and maintains the centers of the clusters.

Methods
- AddPoint(int i, int g)
- MovePoint(int i, int g)
- SetCenters()
- Group(int i)
- Center(int g)