Case Study 2: Document Retrieval

Finding Similar Documents Using Nearest Neighbors

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Emily Fox
January 22\textsuperscript{nd}, 2013

©Emily Fox 2013

Nearest Neighbor with KD Trees

- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

©Emily Fox 2013
Nearest Neighbor with KD Trees

- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Complexity

- For (nearly) balanced, binary trees...
  - Construction
    - Size: $2N - 1 \rightarrow O(N)$
    - Depth: $O(\log N)$
    - Median + send points left right: $O(N)$ at every tree level
    - Construction time: $O(N \log N) \leftarrow$ (smart)
  - 1-NN query
    - Traverse down tree to starting point: $O(\log N)$
    - Maximum backtrack and traverse: $O(N)$ worst case
    - Complexity range: $O(\log N) \rightarrow O(N)$

- Under some assumptions on distribution of points, we get $O(\log N)$ but exponential in $d$ (see citations in reading)
Inspections vs. $N$ and $d$

- \( \log N \)
- \( \text{Exponential} \)

\[ (x^i \in \mathbb{R}^d) \]

K-NN with KD Trees

- Exactly the same algorithm, but maintain distance as distance to furthest of current $k$ nearest neighbors
- Complexity is: \( \mathcal{O}(k \log N) \)
Approximate K-NN with KD Trees

- Before: Prune when distance to bounding box > \( r \)
- Now: Prune when distance to bounding box > \( \frac{r}{\alpha} \)
- Will prune more than allowed, but can guarantee that if we return a neighbor at distance \( r \), then there is no neighbor closer than \( \frac{r}{\alpha} \).
- In practice this bound is loose...Can be closer to optimal.
- Saves lots of search time at little cost in quality of nearest neighbor.

Wrapping Up – Important Points

- kd-trees
  - Tons of variants
    - On construction of trees (heuristics for splitting, stopping, representing branches...)
    - Other representational data structures for fast NN search (e.g., ball trees,...)

- Nearest Neighbor Search
  - Distance metric and data representation are crucial to answer returned

- For both...
  - High dimensional spaces are hard!
    - Large \( d \)
      - Number of kd-tree searches can be exponential in dimension
        - Rule of thumb... \( N \gg 2^d \)... Typically useless.
      - Distances are sensitive to irrelevant features
        - Most dimensions are just noise → Everything equidistant (i.e., everything is far away)
        - Need technique to learn what features are important for your task
What you need to know

- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large $N$
- $kd$-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using tree
  - Complexity of construction and query
  - Challenges with large $d$
Using Hashing to Find Neighbors

- KD-trees are cool, but...
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data
- Approximate neighbor finding...
  - Don't find exact neighbor, but that's OK for many apps, especially with Big Data
- What if we could use hash functions:
  - Hash elements into buckets:
    - Look for neighbors that fall in same bucket as \( x \):
  - But, by design...

Locality Sensitive Hashing (LSH)

- A LSH function \( h \) satisfies (for example), for some some similarity function \( d \), for \( r > 0 \), \( \alpha > 1 \):
  - \( d(x, x') \leq r \), then \( P(h(x) = h(x')) \) is high
  - \( d(x, x') > \alpha r \), then \( P(h(x) = h(x')) \) is low
  - (in between, not sure about probability)

©Carlos Guestrin 2013
Random Projection Illustration

- Pick a random vector \( v \):  
  - Independent Gaussian coordinates  
  - Preserves separability for most vectors  
  - Gets better with more random vectors

Multiple Random Projections: Approximating Dot Products

- Pick \( m \) random vectors \( v^{(i)} \):  
  - Independent Gaussian coordinates  
  - Approximate dot products:  
  - Cheaper, e.g., learn in smaller \( m \) dimensional space  
  - Only need logarithmic number of dimensions!  
  - \( N \) data points, approximate dot-product within \( \epsilon > 0 \):  
    \[
    m = O\left( \frac{\log N}{\epsilon^2} \right)
    \]
    if \( N \) is big, \( m \) is small, but only need \( \log N \) random vectors

- But all sparsity is lost
LSH Example: Sparser Random Projection for Dot products

- Pick random vectors $\mathbf{v}^i$
- Simple 0/1 projection: $h_i(x) = \begin{cases} 1 & \text{if } \text{sign}(\mathbf{v}^i \cdot \mathbf{x}) \geq 0 \\ 0 & \text{if } \text{sign}(\mathbf{v}^i \cdot \mathbf{x}) < 0 \end{cases}$
- Now, each vector is approximated by a bit-vector $\phi(x) = (0, 0, 1, 0, 1, 1, 1, 0)$
- Dot-product approximation:

$$\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \approx 1 - 2 \frac{\text{HammingDistance}(\phi(x), \phi(y))}{m} = 1 - 2 \frac{\|\phi(x) - \phi(y)\|_2^2}{m}$$

LSH for Approximate Neighbor Finding

- Very similar elements fall in exactly same bin:
- And, nearby bins are also nearby:
- Simple neighbor finding with LSH:
  - For bins $b$ of increasing hamming distance to $h(x)$:
    - Look for neighbors of $x$ in bin $b$
    - Stop when run out of time
- Pick $m$ such that $N/2^m$ is “smallish”
Hash Kernels: Even Sparser LSH for Learning

- Two big problems with random projections:
  - Data is sparse, but random projection can be a lot less sparse
  - You have to sample many huge random projection vectors
  - And, we still have the problem with new dimensions, e.g., new words

- Hash Kernels: Very simple, but powerful idea: combine sketching for learning with random projections

- Pick 2 hash functions:
  - \( h \): Just like in Min-Count hashing
  - \( \xi \): Sign hash function
    - Removes the bias found in Min-Count hashing (see homework)

- Define a "kernel", a projection \( \phi \) for \( x \):
  - for each non-zero element \( x_j \)
    - add bin \( h(j) \)
    - \( x_j \) contributes \( h(j) \)

\[
\phi(x) = \sum_{j: h(j) = i} \xi(j) x_j
\]

Hash Kernels, Random Projections and Sparsity

- Hash Kernel as a random projection:
  - \( \psi = (0 0 0 1 0 0 0) \) has \( h(\cdot) = \) (1 0 0 0 1 0 0 1)
  - \( \phi(x) = \sum \xi(j) x_j \)

- Random projection vector for coordinate \( i \) of \( \psi \):
  - mostly zero
  - non-zero for \( j: h(j) = i \)

- Implicitly define projection by \( h \) and \( \xi \), so no need to compute a priori and automatically deal with new dimensions

- Sparsity of \( \psi \), if \( x \) has \( s \) non-zero coordinates:
  - how many times does \( y_j \) "show up" in \( \psi(x) \) one!
  - \( h(j) \)
  - thus if sparsity of \( x \leq s \), sparsity of \( \psi(x) \)
Hash Kernels Preserve Dot Products

- Hash kernels provide unbiased estimate of dot-products!
  \[ E_{h,i} [\phi(x) \cdot \phi(y)] = x \cdot y \]
  **proof:** by homework
- Variance decreases as \(O(1/m)\) gets better with more dims
- Choosing \(m\)? For \(\epsilon > 0\), if
  \[ m = O\left(\frac{\log N}{\epsilon^2}\right) \]

Under certain conditions...
Then, with probability at least 1-\(\delta\):
\[
(1 - \epsilon)||x - x'||_2^2 \leq ||\phi(x) - \phi(x')||_2^2 \leq (1 + \epsilon)||x - x'||_2^2
\]

Learning With Hash Kernels

- Given hash kernel of dimension \(m\), specified by \(h\) and \(\xi\)
  - Learn \(m\) dimensional weight vector
- Observe data point \(x\)
  - Dimension does not need to be specified a priori!
- Compute \(\phi(x)\):
  - Initialize \(\phi(x)\)
  - For non-zero entries \(j\) of \(x_j\):
    \[
    \phi_{h,j} = \{ (j) : x_j \}
    \]
  - E.g., \(j = h(u) \cdot \xi, \{ (u) = -1 \}
  \[ \phi_x = x \cdot w \]
- Use normal update as if observation were \(\phi(x)\), e.g., for LR using SGD:
  \[
  w_i^{t+1} \leftarrow w_i^t + \eta_t \left\{ -\lambda w_i^t + \phi_i(x^{(t)})[y^{(t)} - P(Y = 1|\phi(x^{(t)}), w^{(t)})] \right\}
  \]
  \[
  P(Y = 1|\phi(x^{(t)}), w^{(t)}) = \frac{\exp(\phi(x^{(t)}) \cdot w^{(t)})}{1 + \exp(\phi(x^{(t)}) \cdot w^{(t)})}
  \]
Interesting Application of Hash Kernels: Multi-Task Learning

Personalized click estimation for many users:
- One global click prediction vector $w$: predict using $w \cdot x$
  - But... people are unique
- A click prediction vector $w_u$ per user $u$: predict with $w_u \cdot x$
  - But... people are lazy

Multi-task learning: Simultaneously solve multiple learning related problems:
- Use information from one learning problem to inform the others

In our simple example, learn both a global $w$ and one $w_u$ per user:
- Prediction for user $u$: $(w + w_u) \cdot x = w \cdot x + w_u \cdot x$
  - If we know little about user $u$: usually $w \cdot x$
  - After a lot of data from user $u$: using $w + w_u$ as your vector

Problems with Simple Multi-Task Learning

- Dealing with new user annoying, just like dealing with new words in vocabulary

- Dimensionality of joint parameter space is HUGE, e.g. personalized email spam classification from Weinberger et al.:
  - 3.2M emails
  - 40M unique tokens in vocabulary
  - 430K users
  - 16T parameters needed for personalized classification!
Hash Kernels for Multi-Task Learning

- Simple, pretty solution with hash kernels:
  - Very multi-task learning as (sparse) learning problem with (huge) joint data point \( z \) for point \( x \) and user \( u \):
    \[
    z_{(x,u)} = (x_1, \ldots, x_k, u_1, \ldots, u_m)
    \]
  - Estimating click probability as desired:
    \[
    w = (w_1, w_2, \ldots, w_k, w_{k+1}, \ldots, w_{k+m})
    \]
    \[
    z_{(w)} = w \cdot x + w_{k+1} u
    \]
  - Address huge dimensionality, new words, and new users using hash kernels:
    \[
    \phi(x, u) \text{ just like with hash kernels}
    \]
    \[
    \phi = \sum_{i} \{ (i) x_j \}
    \]
    - Desired effect achieved if \( j \) includes both
      - just word (for global \( w \))
      - word, user (for personalized \( w_{k+1} \))

Simple Trick for Forming Projection \( \phi(x, u) \)

- Observe data point \( x \) for user \( u \)
  - Dimension does not need to be specified a priori and user can be unknown!
- Compute \( \phi(x, u) \):
  - Initialize \( \phi(x, u) = 0 \)
  - For non-zero entries \( j \) of \( x \):
    - E.g., \( j = \text{'Obamacare'} \)
    - Need two contributions to \( \phi \):
      - Global contribution
      - Personalized Contribution
    - Simply:
      \[
      \phi_h^{(x_{Obamacare})} \quad \phi_h^{(x_{Obamacare} - u_x B)}
      \]
    - Learn as usual using \( \phi(x, u) \) instead of \( \phi(x) \) in update function
Results from Weinberger et al. on Spam Classification: Effect of $m$

Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (global-hashed) converges relatively soon, showing that the distortion error $\epsilon_d$ vanishes. The personalized classifier results in an average improvement of up to 30%.

Results from Weinberger et al. on Spam Classification: Illustrating Multi-Task Effect

Figure 3. Results for users clustered by training emails. For example, the bucket $[8,15]$ consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up to 65% reduction in spam), even users that did not contribute to the training corpus at all obtain almost 20% spam-reduction.
What you need to know

- Locality-Sensitive Hashing (LSH): nearby points hash to the same or nearby bins
  - LSH use random projections
    - Only $O(\log N/\varepsilon^2)$ vectors needed
    - But vectors and results are not sparse
  - Use LSH for nearest neighbors by mapping elements into bins
    - Bin index is defined by bit vector from LSH
    - Find nearest neighbors by going through bins

- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash function
    - Can even use one hash function, and take least significant bit to define $\xi$
  - Quickly generate projection $\phi(x)$
  - Learn in projected space

- Multi-task learning:
  - Solve many related learning problems simultaneously
  - Very easy to implement with hash kernels
  - Significantly improve accuracy in some problems
    - If there is enough data from individual users