Reflection Week 6

Summary

This week's lectures explore various algorithms in machine learning, especially natural language processing and information retrieval, including embeddings, distances, k-means algorithms, k-neighbor algorithms, among others. Also introduced are concepts of hashing and clustering.

Concepts

- Bag of Words: a representation model in which each document become a W-dimension vector where W is the number of words in the entire corpus of documents, and each value $x_i[j]$ represents the number of times word j appears in the document i.
  - Pro: simple to describe and compute
  - Con: common words dominate while uncommon words uniquely define a document
- TF-IDF stands for term frequency-inverse document frequency:
  - term frequency = word count of a certain word
  - inverse document frequency = $\log(\frac{\#docs}{1+\#docs\ using\ words})$

  Multiply TF & IDF gives the final weight. Common words that appear in every document (do not uniquely define any specific document) will have a small IDF, making the TF-IDF small. Unique words being common locally (uniquely define a document, like special terminologies) and rare globally (do not appear in others) will have higher weights.
- Distance concepts (smaller distance is better to find two close vectors):
  - Euclidian distance = L-2 norm
  - Manhattan distance = L-1 norm
  - Weighted Euclidian distance

$$distance(x_i, x_q) = \sqrt{\sum_{j=1}^{D} a_j^2 (x_i[j] - x_q[j])^2}$$

- Similarity (bigger is better):

$$x_i^T x_q = \sum_{j=1}^{D} x_i[j] x_q[j]$$

- Cosine similarity is just normalized similarity: $\frac{x_i^T x_q}{||x_i||_2 ||x_q||_2}$
- Normalization is not desired when comparing documents of different sizes since it ignore lengths, so we can just cap the maximum word count. We can also use multiple distance metric and combine them using defined weights
- Nearest Neighbor Algorithms (k-NN):

  Basic implementation steps:

  For a query point x_q and training data $\mathcal{D} = (x_i, y_i)_{i=1}^n$ (n points, y can be either labels (classification) or numbers (regression)), calculate distances between this point and other points in the dataset: $(x^{NN_1}, y^{NN_1}), \dots, (x^{NN_k}, y^{NN_k}) = NearestNeighborAlgorithm(x_q, \mathcal{D}, \dots)$, which the y values as the weight. Further processing may be required for different variations of the algorithm

    - 1-NN algorithm: easy to visualize but will be inaccurate if data is sparse and can cause overfitting, since it relies too much on a single data point (susceptible to data noise)
    - k-NN algorithm: estimations are less volatile, but needs to determine ideal k value and still appears to have discontinuties. Weight is further processed by $\hat{y}_q = \frac{1}{k}\sum_{j=1}^{k} y^{NN_j}$ to prevent over-dependence
    - Weighted k-NN algorithm: Weight is further processed by:

      $$\hat{y}_q = \frac{\sum_{j=1}^{k} c_{q,NN_j} \times y^{NN_j}}{\sum_{j=1}^{k} \times c_{q,NN_j}}$$

      with c_{q,NN_j} defined with a function called a kernel to turn the distance into weight the satisfies the properties:

        - small if the distance is large
        - large if the distance is small
- Local sensitive hashing (LSH):
    - break the data into smaller bins based on how close they are to each other
    - to find the nearest neighbor, place the point into a bin and do an exact nearest neighbor search for the points in the bin
    - based on the number of bins we have, we can implement bin indexing based on some chosen lines, and store points in hash table indexed by all bin indexes, with bins including arrays of 1 and 0 (bit values).
    - find nearby bins to look for neighbors by flipping indices (bits).

  Implementation:

  1. Preprocessing:
      - Draw h lines randomly with $h \approx \log(\mathcal{D})$
      - For each data point, compute $Score(x_i)$
      - Translate scores into binary indices, then use them to store points in a hash table
  2. Querying:
      - For query point, compute its score for each of the lines
      - Translate scores into binary indices and look up all data points that have the same key in the hash table
      - Perform exact nearest neighbor search in this bin
      - (Optional) search for values in nearby bins
- Clustering is the process of finding clusers and assigning examples to a particular cluster. We can use the k-means algorithm to perform clustering:

  Implementation:

  0. Initialize cluster centers randomly

  Repeat until convergence:

  1. Assign each example to its closest cluster centroid by calculating distances
  2. Update the centroids to be the average of all the points assigned to that cluster

Concerns

- Why do we have to specify a hash table in LSH? Is it because of the bitwise nature of the algorithm?
- What does the λ in the kernel do?