Lecture 12 - Modelling Prices

Summary

Predicting the price of an item based using the prices of similar items is something data mining is often used for. This is clearly a regression problem, although the underlying function could potentially be extremely complex. You've probably seen the **K Nearest Neighbours** (KNN) algorithm used for classification, but in this lecture you'll see how we can use it for regression. We'll look at a number of extensions of KNN and then look at how some of the computational problems associated with KNN-like methods can be mitigated.

Key points

KNN for classification and regression

- KNN is a simple idea:
  - Items represented by vectors in a feature-space should share similar target attributes (either classes or a numeric value) to other items located *close-by* in that space
  - For classification we find the K-closest items and select the output class from the **mode** of the classes of the similar items
  - For regression we can find the K-closest items and select the output value from the **mean** of the values of the similar items

Choosing K:

- Choosing the correct K value can be very important
  - especially in regression problems where there might be a lot of noise in the target values of the training data
  - Too small and we might overfit to any noise
  - Too big and we smooth too much...
- Setting K must be done empirically to maximise performance (e.g. minimise residual error in regression; maximise accuracy in classification)
  - Typically cross-validation is used to get robust performance measurements
  - Optimisation could be manual or automatic

Weighted KNN

- Rather than taking the average value of the neighbours (or mode for classification), weight neighbours based on their distance
  - Intuition: neighbours further away are likely to be less similar, so should have less effect
- Common weighting schemes include:
  - Inverse
  - Subtraction
  - Gaussian
- Best weighting scheme for particular task must be determined empirically
  - Again, cross-validation & optimisation

Dealing with Heterogeneous Variables

- In most datasets the variables/features don't have the same range or scale of values
    - But if we're computing distances this is obviously important - features with bigger ranges would have more of an impact
- What about entirely irrelevant variables?
    - Might force things to be far apart even though they should be considered to be similar...
- Normalisation could be the solution:
    - Bring all features to same range
    - But is this optimal?
        - might it be better to choose scale factors for each feature that together optimise the performance?
        - Could also use this to perform feature selection by setting weight to 0 for certain features

KNN Problems

- KNN is computationally expensive if there are:
    - Lots of training examples
    - Many dimensions
- However, on the flip side,
    - More examples *generally* mean better accuracy
    - More dimensions *generally* give more descriptive power
        - unless dimensions are highly correlated or irrelevant...
- Approximate Nearest Neighbour (NN) approaches and dimensionality reduction can help make KNN tractable with lots of high-dimensional data

Fast Approximate Nearest Neighbours

- There are a number of possible techniques for speeding-up nearest-neighbour search in high dimensional spaces:
    - Tree structures: **K-d Trees**
        - A k-d tree is a binary tree structure
        - Each node splits a specific dimension of the space in two
        - The leaf nodes store a number of points corresponding to the points that have made it down the tree to that point
        - Fast nearest-neighbour search can be achieved by walking down the tree until a leaf is hit.
            - Brute force search can be used to select the neighbour from the points in the leaf; unfortunately, this isn't guaranteed to actually be the closest, so you have to back-track up the tree looking in down the other paths to different leaf nodes until you can be assured that you've checked all the leaves that can possibly contain the neighbour (this is still just a small subset of all the leaves in the tree)
    - Hashing
        - Some recently introduced techniques allow feature vectors to be hashed with special hashing schemes that allow vectors that are spatially similar (i.e. in the sense of Euclidean or other distance/similarity measures) to have similar hash codes!
        - These are called **Locality Sensitive Hash** (LSH) Functions
            - Input vectors hashed so that similar items map to the same *buckets* with high probability

        - number of buckets is much smaller than the set of input items
        - LSH differs from conventional and cryptographic hash functions because it aims to maximise the probability of a *collision* for similar items
    - There are many possible *families* of LSH functions
        - An LSH family is defined for a
            - metric space $\mathbf{M}=(M,d)$
            - a threshold $R>0$
            - and an approximation factor $c>1$
        - The family $\mathbf{F}$ is a family of functions $h : \mathbf{M} \rightarrow S$ which map elements from the metric space to a bucket $s \in S$.
        - $\mathbf{F}$ satisfies the following conditions for any two points $p, q \in M$, using a function $h \in \mathbf{F}$ which is chosen uniformly at random:
            - if $d(p, q) \leq R$, then $h(p) = h(q)$ (i.e. p and q collide) with probability at least P_1,
            - if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most P_2.
        - A family is **interesting** when $P_1 > P_2$
    - For example, LSH functions can be formulated using p-Stable distributions such that the underlying metric is L1 or L2
    - Can use LSH as a basis for NN search using standard hash tables
        - Combine responses of a list of LSH's into a vector of integers
            - Store in a hash table of <int[], Set of points>
        - Use multiple hash tables to better ensure probability of collision
- Sketching
    - Sketching concatenates binary hashes into a bit string
    - With the correct LSH function, the Hamming distance between a pair of sketches is proportional to the Euclidean distance between the original vectors
        - Can easily compress features to relatively small number of bits
        - Hamming distance computation is cheap and can be performed using lookup tables and bitwise operations

Dimensionality reduction revisited

- Already looked at a few techniques a few lectures ago
    - Some of those (MDS, SOM) not really suitable for this in terms of doing NN analysis…
- What about PCA?
    - Johnson-Lindenstrauss lemma: "if points in a vector space are of sufficiently high dimension, then they may be projected into a suitable lower-dimensional space in a way which approximately preserves the distances between the points"
    - PCA preserves distances, but it might not preserve class separation
        - It could in the worse case make two classes that were separable in a high dimensional space inseparable in the lower dimensional space
    - Rather than choosing a lower dimensional basis using the eigenvectors, why not choose the basis to project to randomly?
        - Pick a random unit vector
            - Pick a random unit vector orthogonal to any vectors already selected
            - Repeat until we have the required number of dimension
        - High probability of preserving distances
        - Potentially less likelihood of choosing a direction that looses separability

Further Reading

- Chapter 8 of "Programming Collective Intelligence" gives a good overview of regression KNN, weighting, cross-validation and optimisation
- The original paper of LSH using p-stable distributions is quite accessible:
  - Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. Datar et al. In Proceedings of the twentieth annual symposium on Computational geometry. 2004. http://people.csail.mit.edu/mirrokni/pstable.ps (http://people.csail.mit.edu/mirrokni/pstable.ps)
- Wikipedia is a good starting point to reading about general ideas related to approximate NN and dimensionality reduction:
  - https://en.wikipedia.org/wiki/Random_projection (https://en.wikipedia.org/wiki/Random_projection)
  - https://en.wikipedia.org/wiki/Locality-sensitive_hashing (https://en.wikipedia.org/wiki/Locality-sensitive_hashing)
  - https://en.wikipedia.org/wiki/Stable_distribution (https://en.wikipedia.org/wiki/Stable_distribution)
  - https://en.wikipedia.org/wiki/Dimensionality_reduction (https://en.wikipedia.org/wiki/Dimensionality_reduction)