**Machine Learning**

**Session 9**

1. What do we mean by ‘Features’
   1. Input database columns, aka Attributes, Dimensions.
   2. What data to include as the input to your learning procedure
      1. Sometimes choice is out of your scope or obvious from domain/business setting
      2. Sometimes choice is an opportunity for engineering / intuition
      3. (Given that more attributes potentially provide more information, but potentially increase overfitting, memory and slow the computation).
2. Feature Engineering
   1. Input and Output for Machine Learning Algorithm
      1. {xi =[x1,…,xd], yi }
   2. How to convert real life data:
      1. Into xi =[x1,…,xd], yi ?
      2. What to store in your database?
   3. Domain Specific
      1. Human Expertise
   4. It may make sense for your input to be some transformation of the raw data
      1. Because your data may not be fixed length
      2. Because the right non-linear transformation can make learning easier
      3. Because the right low-dimensional transform could help avoid over-fitting.
   5. Designing this transformation:
      1. Feature engineering
3. Feature Engineering: Examples
   1. E.g., Audio or accelerometer data
      1. Full data is waveform
      2. Probably don’t want to use directly: Variable and high dimensions
      3. (E.g., 5sec/44KhZ/stereo: Half million columns.)
   2. Features:
      1. High-amplitude count (1d) … Loud noise detector.
      2. Zero-crossing count(1d) … Activity recognition.
      3. Fourier Coefficients (e.g., 128d) … Music / Speech recognition
   3. Case Study @ EECS:
      1. Activity + identity recognition on mobile phone accelerometer
   4. E.g., Images.
      1. Full data is all the pixels
   5. Features:
      1. Average brightness (1d) e.g., Day vs Night.
      2. Color histogram (3 – 1000dims) e.g., Recognize objects
      3. Histogram of Gradients (~128dim) e.g., Detect objects
      4. Raw Pixels (10000dim+) e.g., Face recognition
   6. Internet Advertising
      1. Full data is everything on your facebook profile.
         1. Should we show an add about premium baby-items?
         2. Engineer a feature aggregating high income related likes+posts & a feature aggregating baby/mother related likes+posts
      2. Could result in a simpler & more efficient model than if you threw everything in?
   7. Text
      1. The most common choice for text is “Bag of Words”
      2. E.g., Raw data:
         1. “John likes to watch movies. Mary likes movies too.”
            1. John: 1, likes: 2, to: 1, ….
         2. “John also likes to watch football games.”
            1. John: 1, also: 1, likes: 1,…
      3. Result is a “bag of words” vector for each text document
         1. Length: # of words in the dictionary.
         2. Row: Frequency of each word in a document
         3. Sum of a row: Number of words in corresponding document
      4. Can also use bi-grams, trigrams, n-grams
4. Feature Engineering
   1. Input may be some transformation of the raw data
      1. Because your data may not be fixed length
         1. Because the right non-linear transformation make the problem easier
         2. Because the right low-dimensional transform could help avoid over-fitting.
   2. Sometimes cleverly derived features can simplify learning.
      1. E.g., House price database has length+width of room.
      2. => Linear regressor on area=L\*W, non-linear regressor on length & width.
   3. But can have more features than original data
   4. Dichotomy between designing exactly the right feature (#2) and designing very many features (#1) above.
      1. If we include/make many features in the hope of finding a good one….
      2. We may not know which are relevant
      3. Risk of over-fitting
5. Many Dimensions
   1. Suppose we are given or have designed our features. Then…
   2. We often end up with many dimensions
      1. Because we over-killed on feature engineering
      2. Because it’s a problem where we have very little prior knowledge, so had no choice but to include everything.
         1. E.g., drug discovery, genome analysis
6. Why Reduce Dimensions?
   1. “Curse of Dimensionality”
   2. Irrelevant Data
   3. Computation Time
   4. Visualization
   5. Interpretation
   6. Many applications have > 106 features (columns)
7. Why Reduce Dimensions? “Curse of Dimensionality”
   1. Human intuition breaks down in high dimensions
      1. E.g., Gaussian, Cube, Sphere-Cube.
      2. Everything is similarly far away
   2. So do many machine learning algorithms…
      1. Slow
      2. Inaccurate
      3. Makes over-fitting very easy, so poor generalization
   3. E.g.,
      1. KNN: Not robust to high dimensions.
      2. Linear Regression. Need data > dimensions.
8. Why Reduce Dimensions? Irrelevant Data
   1. Curse of Dimensionality especially dangerous if
      1. Many weakly relevant dimensions
      2. Some very relevant, but many irrelevant dimensions
   2. Irrelevant dimensions, e.g.,:
      1. Given article content: Classify sports versus technology.
      2. Given someone’s facebook profile, what product should I advertise to them?
   3. Computation Time
      1. We have seen:
         1. Train: Regression O(d^2\*n+d^3)
         2. Test: Regression, MaxEnt: O(nd), KNN O(nd).
         3. (An O(d3) method will be 8x faster with 0.5 the features!)
      2. “Big Data” / Web-scale
         1. N=10^6, d=10^6
         2. => Reducing dimensions is critical
      3. Embedded systems & mobile apps
      4. Real-time apps  
           
         Fig of buses and sheep separated.
   4. Interpretation
      1. Sometimes finding good dimensions is the fundamental aim
      2. E.g.: What causes a program to crash?
         1. Features are aspects of a single program execution
            1. Which branches were taken?
            2. What values did functions return?
         2. Classifier F(Trace): Crash or Not
         3. Features that predict crashes well are probably bugs
      3. E.g.: What causes lung cancer?
         1. Features are aspects of a patient’s medical history
         2. Binary response variable: did the patient develop lung cancer?
         3. Want to legislate against features that predict lung cancer.
   5. Two categories of ways to reduce dimensions….
      1. Feature Selection
         1. Pick a good subset of features (attributes, columns), ignore the others (typically supervised)
      2. Dimensionality Reduction by Linear Projection
         1. Transform linear combination of all features to a smaller set of features (typically unsupervised)
   6. A Taxonomy
      1. A flowchart with the following structure:
         1. Machine Learning (root)
         2. Left node: supervised
         3. Two types: classification and regression
         4. Both point to feature selection.
         5. The right side is: unsupervised
         6. Point to clustering & density estimation, and dimension reduction.
9. Feature Selection Methods
   1. Three main types
      1. Filtering
      2. Wrapper
      3. Embedded
   2. Formally:
      1. Want to learn y=f(x) – x=[x1,..xj ,..xd]
      2. Suspect not all xj are relevant
      3. Task: Find the relevant subset
      4. Challenge: there are 2d subsets!
10. Filtering
    1. Assign a score to each feature: sj =score(j)
       1. Sort features j by score, and pick the top K or top K%.
    2. Common scoring methods
       1. Correlation between Xj and Y
       2. Estimate the mutual information between Xj and Y:  
            
          I(X;Y) = double sum of (p(x,y) \* log((p(x,y)) / (p(x)\*p(y))))
       3. x2 test of statistical independence between Xj and Y.
       4. Domain Specific.
          1. Text: Ignore words such as “the”, “it”
    3. Advantages:
       1. Very fast
       2. Simple to apply.
    4. Disadvantages?
       1. Doesn’t take into account feature interaction
       2. => Apparently useless features can be useful when grouped together
          1. It will miss these
    5. Practical:
       1. Use light filtering as an initial step if training time is a big issue.
11. Wrapper
    1. Filter ignores features that can be useful in conjunction
       1. Also doesn’t account for limitations / power of learning algorithm.
    2. Wrapper methods:
       1. For each subset of features:
          1. Retrain learning algorithm on chosen subset
          2. Evaluate learning algorithm on validation data
       2. Pick the subset which has highest validation performance
    3. Issue:
       1. Repeatedly retraining is costly
       2. There are exponentially many (2d) subsets of features.
12. Wrapper Methods: Greedy Search
    1. Forward selection:
       1. Initialize no feats: fs={}
       2. Do:
          1. Try all unused features s
          2. Find s\* to add that improves performance the most
          3. Add feature s\* to fs.
       3. While: performance improving
    2. Backward selection:
       1. Initialize fs={1,..,d}
       2. Do:
          1. Try removing each feature in fs
          2. Find s\* to remove which improves performance the most
          3. Remove s \* from fs.
       3. While: performance improving.
13. Embedded
    1. Wrapper methods:
       1. Advantage: can be applied to any model (model agnostic)
       2. Disadvantage: suffer from repeated re-train cost and sub-optimality (greedy).
    2. In some special cases, feature selection can be built into a particular learning algorithm
       1. Model specific
       2. May be more efficient / optimal
14. Embedded Methods
    1. We have seen regularization, e.g., MaxEnt and regression
       1. Find w=argmin E(w,D)
       2. This is known as L2 regularization because it penalizes the squared weights:  
            
          E\_{MCLR} (w, D) = - (sum of (log p(y\_i given x\_i) + (lambda \* w^T \* w)))  
          E\_{MSER} (w, D) = sum of (y\_i – f(x\_i)^2) + (lambda \* w^T \* w))
       3. Suppose some dimension j of x is totally irrelevant
          1. Suppose we remove it (setting wj =0)
          2. No effect on the first term
          3. Improves the second term
       4. => Good regularization can help with feature selection.
          1. But how to achieve it?
       5. L2 regularizer: “Ridge”
          1. Fast and easy, but weak feature selection   
               
             R\_2(w) = sum of (w\_j^2)
       6. L0 regularizer is ideal
          1. But very slow optimise (NP hard)
             1. (because not differentiable)  
                  
                R0(w) = sum of I (w\_j != 0)
       7. L1 regularizer: “Lasso”
          1. Commonly chosen tradeoff.
          2. Reasonably easy, reasonably quick  
               
             R\_1(w) = sum of (|w\_j|)
    2. Summary
       1. Sometimes we want to engineer new features:
          1. Raw data may not be suitable (e.g., variable length)
          2. A good derived feature may simplify the problem.
          3. A suitable set of features may be lower dimensional than raw data
       2. Sometimes we want to select features:
          1. When there are many potential inputs, and little domain knowledge to select/engineer them
          2. When there are resource constraints (large scale/embedded)
          3. When we engineered many features in the hope of finding a good one
          4. When the feature selection is itself the goal
15. From Feature Selection to Dimensionality Reduction
    1. So far we selected a subset of good columns (feature selection)
       1. We loose everything in the discarded columns.
    2. Sometimes we want to “compress” all the columns into a smaller number, but loosing the least possible information.
16. Dimensionality Reduction: Overview
    1. Data x, |x|=d
    2. Derived Features z, |z|=k, k < D
       1. z1=F1(x) = x1+x2
       2. z2=F2(x) = 2x3-x1-x2
    3. Feature Selection
       1. z1=x1 – z2=x3
    4. Dimensionality Reduction:
       1. How to find a good linear combination of features?
       2. Restrict ourselves to linear combinations for now
       3. Supervised: Find a linear combination that helps achieve a task.
       4. Unsupervised: Find a linear combination according to some other criteria
17. Dimensionality Reduction: Linear
    1. Linear combinations of features can be expressed as a matrix multiply
       1. z=Ux
    2. E.g., U is a binary row
       1. z is a subset of x according to ones in U
    3. E.g., U is a list of 1s
       1. z is the sum of the elements in x
    4. Lots more options…
       1. So how to find a “good” matrix U? Ideas?
       2. Pick U that explains the data well / looses little information
18. Geometric Intuition
    1. d=2, k=1
    2. Which axis do we project to?
       1. z=x1?
       2. z=x2?  
            
          Fig of a cluster of points on a xy axis graph.
19. Dimensionality Reduction: Residual Error
    1. Can measure the residual error of projecting to a particular axis?  
         
       Fig of a graph where the points error is measured as their x-y distance from the line of best fit.
20. Dimensionality Reduction: Principal Components Analysis (PCA)
    1. PCA Objective
       1. Project to the axis with minimum residual error
    2. Encoder: z=U^Tx
    3. Decoder: x=Uz
    4. Find matrix U for minimum error:  
         
       E(U) = sum of (x\_i – x\_j)^2 = sum of (U\*U^T\*x\_i – x\_j)^2
    5. Same as maximize variance of z
21. Dimensionality Reduction: Principal Components Analysis (PCA)
    1. How to solve PCA?
       1. Turns out the right basis is given by the eigenvectors of the covariance matrix. So: [U,V]=eigenvalue of (XX^T)
       2. Faster version without explicit covariance: [U,S,V]=svd(X)
          1. Rows of U are the basis
          2. Diagonal of S are the eigenvalues
          3. Pick the first k rows
       3. Encode z=U(1:k,:)x
       4. Decode x=U(1:k,:)^T \* z
       5. Note:
          1. Using all the Evs will store the exact data
22. PCA: How to choose the number of dimensions
    1. How to choose?
    2. Each eigenvalue tells you what fraction of the variance/error is accounted for.
       1. Strategy 1: Pick k dimensions.
    3. If you plot the eigenvalues, you typically get
       1. Strategy 2: Take a number of eigenvalues such that you account for P% of the variance (E.g., P=99%)
23. PCA: Pitfalls
    1. You must subtract the mean of the data first
    2. NOT scale invariant, need to rescale first.
    3. ‘Traditional’ implementation computes Cov(X)=X\*X^T which is already O(nd^2)
       1. SVD can compute top k singulars in O(ndk)
    4. Second order statistics / Gaussianity assumption. This can hide many interesting patterns
    5. Non-linear manifolds are not covered
    6. Not discriminative
       1. The information your problem needs could be in a low-variance dimension
24. PCA: Versus Linear Regression
    1. Regression: Predict a special output variable (y) given others (x)
    2. PCA: No special variable, model all the data (x) with maximum fidelity
    3. Error on y only versus error on all x.
25. PCA Example: Economics
    1. Many Economic Statistics
       1. What are the underlying factors?
       2. Reduce to 2 dimensions and plot…
          1. Reveals aggregate “Wealth” + “Size” factors
26. PCA Example: Politics
    1. Opinion database
       1. Rows: People
       2. Columns: Opinions (Immigration, Crime, Tax, Welfare, Drugs, etc)
    2. PCA -> 1D
       1. Rows: People
       2. Column: Left<->Right:
          1. “Lib Dem <-> Conservative”
    3. PCA -> 2D
       1. Rows: People
       2. Columns:
          1. Economic & Social views
27. PCA Examples: Eigen-faces
    1. Each face image is a database row
       1. E.g. 100x100=10000 columns.
       2. What if we require to store each face image in only
28. PCA Examples: Eigen-faces
    1. Each face image is a database row
    2. First dimensions correspond to lighting, thereafter face, hair structure
    3. Extensively used for face recogntion
       1. Speed, memory, accuracy
       2. Tuck away lighting…
    4. Example of reconstruction with increasing number of PCs
    5. Connection to general Image Compression
       1. Linear vs Non-linear (eg DCT)
       2. Not perceptually motivated
       3. So works, but not great •
    6. But good for revealing structure
29. PCA Recap
    1. Plus
       1. Standard tool.
       2. Available in most languages / toolkits.
       3. Fairly Fast and robust. •
    2. Minus
       1. Assumes linear
       2. Assumes orthogonal dimensions
       3. Assumes Gaussian
       4. Not “discriminatively trained”
30. Beyond PCA
    1. Non-negative matrix factorization
       1. Bases have to be positive
    2. Linear Discriminant Analysis & Partial Least Squares
       1. Find a lower-dimension projection that help separate classes
    3. Factor Analysis
       1. Don’t assume the dimensions are orthogonal
    4. Independent Component Analysis
       1. Look for the most independent basis (e.g., blind-source separation)
    5. Non-linear methods: Isomap, LLE, etc