**Dimensionality Reduction (PCA)**

**Why should we look at dimensionality reduction?**

-Speeds up algorithms

-Reduces space used by data for them

-Dimensionality reduction can improve how we display information in a tractable manner for human consumption

-Often helps to develop algorithms if we can understand our data better

**What is dimensionality reduction?**

-You've collected many features - maybe more than you need

-Redundant data set - different units for same attribute

-Reduce data to 1D (2D->1D)

-Data redundancy can happen when different teams are working independently

-Gives lossy compression, but an acceptable loss (probably)

**Principle Component Analysis (PCA): Problem Formulation**

-For the problem of dimensionality reduction the most commonly used algorithm is PCA

-Say we have a 2D data set which we wish to reduce to 1D

-find a single line onto which to project this data

How do we determine this line?

-The distance between each point and the projected version should be small (projection error)

-PCA tries to find the surface (a straight line in this case) which has the minimum projection error

**You should normally do mean normalization and feature scaling on your data before PCA**

**A more formal description is**

* For 2D-1D, we must find a vector u(1), which is of some dimensionality
* Onto which you can project the data so as to minimize the projection error

 u(1) can be positive or negative (-u(1)) which makes no difference

**In the more general case**

* To reduce from nD to kD we
  + Find *k* vectors (u(1), u(2), ... u(k)) onto which to project the data to minimize the projection error
  + So lots of vectors onto which we project the data
  + Find a set of vectors which we project the data onto the linear subspace spanned by that set of vectors
    - We can define a point in a plane with k vectors
* e.g. 3D->2D
  + Find pair of vectors which define a 2D plane (surface) onto which you're going to project your data
  + Much like the "shallow box" example in compression, we're trying to create the shallowest box possible (by defining two of it's three dimensions, so the box' depth is minimized)

**How does PCA relate to linear regression?**

PCA is **not** linear regression

 For linear regression, fitting a straight line to minimize the **straight line** between a point and a squared line

* NB - **VERTICAL distance** between point

 For PCA minimizing the magnitude of the shortest **orthogonal distance**

* Gives very different effects
* With linear regression we're trying to predict "y"
* With PCA there is no "y" - instead we have a list of features and all features are treated equally
  + If we have 3D dimensional data 3D->2D
    - Have 3 features treated symmetrically

**PCA Algorithm**

Before applying PCA must do data preprocessing

* Given a set of m unlabeled examples we must do
  + **Mean normalization**
    - Replace each xji with xj - μj,
      * In other words, determine the mean of each feature set, and then for each feature subtract the mean from the value, so we re-scale the mean to be 0
  + **Feature scaling (depending on data)**
    - If features have very different scales then scale so they all have a comparable range of values
      * e.g. xji is set to (xj - μj) / sj
        + Where sjis some measure of the range, so could be

Biggest - smallest

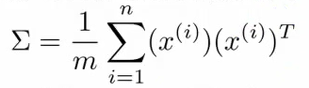
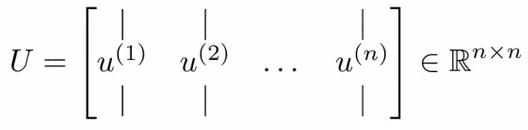
Standard deviation (more commonly)

With preprocessing done, PCA finds the lower dimensional sub-space which minimizes the sum of the square

For 2D->1D we'd be doing something like this

* Compute the **u vectors**
  + The new planes
* Need to compute the **z vectors**
  + z vectors are the new, lower dimensionality feature vectors

**Algorithm description**

* Reducing data from *n*-dimensional to k-dimensional
  + Compute the covariance matrix  
    
    - This is commonly denoted as Σ (greek upper case sigma) - NOT summation symbol
    - Σ = sigma
      * This is an [n x n] matrix
        + Remember than xiis a [n x 1] matrix
    - In MATLAB or octave we can implement this as follows;  
      C:\Books\Machine Learning\Machine_learning_complete\Machine_learning_complete\14_Dimensionality_Reduction_files\Image [13].png
  + Compute eigenvectors of matrix Σ
    - **[U,S,V] = svd(sigma)**
      * svd = singular value decomposition
        + More numerically stable than **eig**
      * **eig**= also gives eigenvector
  + U, S and V are matrices
    - U matrix is also an [n x n] matrix
    - Turns out the columns of U are the u vectors we want!
    - So to reduce a system from n-dimensions to k-dimensions
      * Just take the first *k-vectors* from U (first k columns)
    - Just take the first *k-vectors* from U (first k columns)  
      

 Next we need to find some way to change x (which is n dimensional) to z (which is k dimensional)

* (reduce the dimensionality)
* Take first k columns of the u matrix and stack in columns
  + n x k matrix - call this Ureduce
* We calculate z as follows
  + z = (Ureduce)*T* \* x
    - So [k x n] \* [n x 1]
    - Generates a matrix which is
      * k \* 1

 So in summary

* Preprocessing
* Calculate sigma (covariance matrix)
* Calculate eigenvectors with **svd**
* Take k vectors from U (Ureduce= U(:,1:k);)
* Calculate z (z =Ureduce' \* x;)

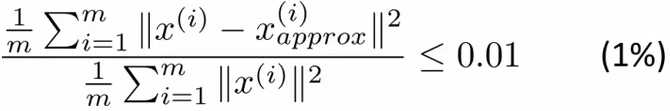
**Choosing the number of Principle Components**

 How do we chose *k*?

* k = number of **principle components**
* Guidelines about how to chose k for PCA

 To chose k think about how PCA works

* PCA tries to minimize averaged squared projection error  
  C:\Books\Machine Learning\Machine_learning_complete\Machine_learning_complete\14_Dimensionality_Reduction_files\Image [17].png
* Total variation in data can be defined as the average over data saying how far are the training examples from the origin  
  C:\Books\Machine Learning\Machine_learning_complete\Machine_learning_complete\14_Dimensionality_Reduction_files\Image [18].png

 When we're choosing k typical to use something like this  


* Ratio between averaged squared projection error with total variation in data
  + Want ratio to be small - means we retain 99% of the variance
* If it's small (0) then this is because the numerator is small
  + The numerator is small when xi = xapproxi
    - i.e. we lose very little information in the dimensionality reduction, so when we decompress we regenerate the same data

 So we chose k in terms of this ratio

 Often can significantly reduce data dimensionality while retaining the variance

**Speeding up supervised learning algorithms**

* Say you have a supervised learning problem
  + Input x and y
    - x is a 10 000 dimensional feature vector
    - e.g. 100 x 100 images = 10 000 pixels
    - Such a huge feature vector will make the algorithm slow
  + With PCA we can reduce the dimensionality and make it tractable
  + How
    - 1) Extract xs
      * So we now have an unlabeled training set
    - 2) Apply PCA to x vectors
      * So we now have a reduced dimensional feature vector z
    - 3) This gives you a new training set
      * Each vector can be re-associated with the label
    - 4) Take the reduced dimensionality data set and feed to a learning algorithm
      * Use y as labels and z as feature vector
    - 5) If you have a new example map from higher dimensionality vector to lower dimensionality vector, then feed into learning algorithm
* PCA maps one vector to a lower dimensionality vector
  + x -> z
  + Defined by PCA **only** on the training set
  + The mapping computes a set of parameters
    - Feature scaling values
    - Ureduce
      * Parameter learned by PCA
      * Should be obtained only by determining PCA on your training set
  + So we use those learned parameters for our
    - Cross validation data
    - Test set
* Typically you can reduce data dimensionality by 5-10x without a major hit to algorithm