share.coursera.org

ML:Dimensionality Reduction - Coursera

- We may want to reduce the dimension of our features if we have a lot of redundant data.
  - To do this, we find two highly correlated features, plot them, and make a new line that seems to describe both features accurately. We place all the new features on this single line.

Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

Note: in dimensionality reduction, we are reducing our features rather than our number of examples. Our variable m will stay the same size; n, the number of features each example from $x^{(1)}$ to $x^{(m)}$ carries, will be reduced.

It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to 3 or less in order to plot it.

We need to find new features, z_1, z_2 (and perhaps z_3) that can effectively **summarize** all the other features.

Example: hundreds of features related to a country's economic system may all be combined into one feature that you call "Economic Activity."

The most popular dimensionality reduction algorithm is *Principal Component Analysis* (PCA)

Problem formulation

Given two features, x_1 and x_2, we want to find a single line that effectively describes both features at once. We then map our old features onto this new line to get a new single feature.

The same can be done with three features, where we map them to a plane.

The **goal of PCA** is to **reduce** the average of all the distances of every feature to the projection line. This is the **projection error**.

> Reduce from 2d to 1d: find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

The more general case is as follows:

> Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data so as to minimize the projection error.

If we are converting from 3d to 2d, we will project our data onto two directions (a plane), so k will be 2.

PCA is not linear regression

- In linear regression, we are minimizing the **squared error** from every point to our predictor line. These are vertical distances.

- In PCA, we are minimizing the **shortest distance**, or shortest *orthogonal* distances, to our data points.

More generally, in linear regression we are taking all our examples in x and applying the parameters in Θ to predict y.

In PCA, we are taking a number of features $x_1, x_2, \dots, x_n$, and finding a closest common dataset among them. We aren't trying to predict any result and we aren't applying any theta weights to the features.

Before we can apply PCA, there is a data pre-processing step we must perform:

Data preprocessing

Given training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Preprocess (feature scaling/mean normalization):

$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

Replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$

If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

Above, we first subtract the mean of each feature from the original feature. Then we scale all the features ($x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$)

We can define specifically what it means to reduce from 2d to 1d data as follows:

$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$

The z values are all real numbers and are the projections of our features onto $u^{(1)}$.

So, PCA has two tasks: figure out $u^{(1)}, \dots, u^{(k)}$ and also to find $z_1, z_2, \dots, z_m$.

The mathematical proof for the following procedure is complicated and beyond the scope of this course.

1. Compute "covariance matrix"

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

This can be vectorized in Octave as:

```
Sigma = (1/m) * X' * X;
```

We denote the covariance matrix with a capital sigma (which happens to be the same symbol for summation, confusingly---they represent entirely different things).

Note that $x^{(i)}$ is an $n \times 1$ vector, $(x^{(i)})^T$ is an $1 \times n$ vector and X is a $m \times n$ matrix (row-wise stored examples). The product of those will be an $n \times n$ matrix, which are the dimensions of Σ.

2. Compute "eigenvectors" of covariance matrix Σ

```
[U,S,V] = svd(Sigma);
```

svd() is the 'singular value decomposition', a built-in Octave function.

What we actually want out of svd() is the 'U' matrix of the Sigma covariance matrix: $U \in \mathbb{R}^{n \times n}$. U contains $u^{(1)}, \ldots,$ $u^{(n)}$, which is exactly what we want.

3. Take the first k columns of the U matrix and compute z

We'll assign the first k columns of U to a variable called 'Ureduce'. This will be an $n \times k$ matrix. We compute z with:

$$z^{(i)} = Ureduce^T \cdot x^{(i)}$$

$Ureduce^T$ will have dimensions $k \times n$ while $x^{(i)}$ will have dimensions $n \times 1$. The product $Ureduce^T \cdot x^{(i)}$ will have dimensions $k \times 1$.

To summarize, the whole algorithm in octave is roughly:

```
Sigma = (1/m) * X' * X;  % compute the
covariance matrix
[U,S,V] = svd(Sigma);    % compute our projected
directions
Ureduce = U(:,1:k);      % take the first k
directions
Z = X * Ureduce;         % compute the projected
data points
```

If we use PCA to compress our data, how can we uncompress our data, or go back to our original number of features?

To go from 1-dimension back to 2d we do: $z \in \mathbb{R} \rightarrow x \in \mathbb{R}^2$.

We can do this with the equation: $x_{approx}^{(1)} = U_{reduce} \cdot z^{(1)}$.

Note that we can only get approximations of our original data.

Note: It turns out that the U matrix has the special property that it is a Unitary Matrix. One of the special properties of a Unitary Matrix is:

$U^{-1} = U^*$ where the "*" means "conjugate transpose".

Since we are dealing with real numbers here, this is equivalent to:

$U^{-1} = U^T$ So we could compute the inverse and use that, but it would be a waste of energy and compute cycles.

How do we choose k, also called the *number of principal components*? Recall that k is the dimension we are reducing to.

One way to choose k is by using the following formula:

> Given the average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - x_{approx}^{(i)}||^2$
>
> Also given the total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2$
>
> Choose k to be the smallest value such that: $\dfrac{\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)} - x_{approx}^{(i)}\right|\right|^2}{\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)}\right|\right|^2} \leq 0.01$ (1%)

In other words, the squared projection error divided by the total variation should be less than one percent, so that **99% of the variance is retained**.

Algorithm for choosing k

1. Try PCA with $k = 1, 2, \dots$
2. Compute U_{reduce}, z, x
3. Check the formula given above that 99% of the variance is retained. If not, go to step one and increase k.

This procedure would actually be horribly inefficient. In Octave, we will call svd:

```
[U,S,V] = svd(Sigma)
```

Which gives us a matrix S. We can actually check for 99% of retained variance using the S matrix as follows:

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

The most common use of PCA is to speed up supervised learning.

Given a training set with a large number of features (e.g. $x^{(1)}, \dots, x^{(m)} \in \mathbb{R}^{10000}$) we can use PCA to reduce the number of features in each example of the training set (e.g. $z^{(1)}, \dots, z^{(m)} \in \mathbb{R}^{1000}$).

Note that we should define the PCA reduction from $x^{(i)}$ to $z^{(i)}$ **only on the training set** and not on the cross-validation or test sets. You can apply the mapping $z^{(i)}$ to your cross-validation and test sets after it is defined on the training set.

Applications

- Compressions
  - Reduce space of data

  - Speed up algorithm

- Visualization of data
  - Choose k = 2 or k = 3

Bad use of PCA: trying to prevent overfitting. We might think that reducing the features with PCA would be an effective way to address overfitting. It might work, but is not recommended because it does not consider the values of our results y. Using just regularization will be at least as effective.

Don't assume you need to do PCA. **Try your full machine learning algorithm without PCA first.** Then use PCA if you find that you need it.

Next: Anomaly Detection Back to Index: Main