Principal component analysis

The technique of principal component analysis (PCA) can be conceptualized in the following manner: First, we find the direction along which the data varies the most (analogous to the semi-major axis of an ellipse). That is the first principal component. We then 'subtract off' that dimension of variation from the data and, in the reduced data, find the direction along which the data varies the most. This is the second principal component. Repeating this process, we end up with p principal components for a dataset of dimension p (containing p variables).

PCA is actually ordinarily calculated through the computation of the singular value decomposition of the covariance matrix of the data, but the above procedure better illustrates the intuition behind PCA. The first principal component (PC1) is the dimension which accounts for as much variation as possible, PC2 accounts for as much variation as possible after PC1, PC3 accounts for as much variation as possible after PC1 and PC2, and so on and so forth. PCA can also be thought of as a rotation of the coordinate axes.

- Read this StackExchange answer explaining the intuition behind PCA.

In the following, we will implement our own version of PCA using the method of successive principal component extraction described above and compare our answer to the PCA method implemented in R's `prcomp()`.

For illustration, let us consider a matrix of example data `X = matrix(1:100, nrow=10)`. First, `scale()` the matrix. (If we want to analyze the dimensions of greatest variation, it's important to first center our data and to scale the variables so that choice of units does not affect our end result.) We can get the results of PCA by simply running `prcomp(X)`, with the principal components being stored in the `$rotation` variable of the `prcomp()` output. Each principal component is represented by a vector pointing along the dimension corresponding to that PC.

We want to implement our own method for extracting the principal components, which will also be represented by such vectors. In order to check our results, we need a way to see if the `prcomp()` PCs point in the same directions as our PCs. To determine if two vectors point in the same direction, we can't just compare the elements of the vectors; for example, $(1, 1)$ and $(10, 10)$ point in the same direction but are clearly quite different on an element-by-element basis.

The key is to look at the dot product of the two vectors, given by $\mathbf{v} \cdot \mathbf{w} = \sum_i v_i w_i$. It is also given by the identity $\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \, \|\mathbf{w}\| \cos\theta$, where $\|\mathbf{v}\|$ is the *magnitude* of the vector $\mathbf{v}$ (the square root of the sum of the square of each element of $\mathbf{v}$) and θ is the *angle* between the two vectors $\mathbf{v}$ and $\mathbf{w}$. If the angle between two vectors is close to either 0 or 180, then they point along the same direction (an angle of 180 means that one is flipped relative to the other, like $(1, 1)$ and $(-1, -1)$).

- Write a function `norm(v)` which calculates the norm of `v`, `dot_prod(v,`

w) which calculates the dot product of v and w, and angle(v, w) which calculates the angle between v and w. You may find the inverse cosine function acos() useful. Return the angle in degrees rather than radians.

Next, given a matrix of data **X**, the vector representing the dimension of greatest variation in the data is given by the vector **w** which *maximizes* the expression $(\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w})$ / $(\mathbf{w}^\top \mathbf{w})$, where $\mathbf{X}^\top$ represents the transpose of the matrix **X**.

- Write a function objective(X, w) which calculates the value of the above expression for a given matrix X and vector w. Since it is an expression which we wish to maximize, it is called an objective function.

Add the following code into your script:

```
objective_X = function(X) {
  function (w) {
    objective(X, w)
  }
}
```

The above function, objective_X(), will return a *function of* w *only* which evaluates the objective function at that value of w. The purpose of objective_X() is as follows: Often, we will want to evaluate the objective function for a given data matrix X many different times for different values of w. As such, we can set obj = objective_X(X), and whenever we want to evaluate the objective function for different vectors w we just call obj(w).

- Write a function extract_pc(X) which uses optim() to maximize the objective function in order to find a vector corresponding to the first principal component of the data matrix X. Keep the following in mind:
  - By default, optim() tries to *minimize* the objective function; read the documentation to figure out how to change this to maximization.
  - Change the default behavior of optim() so that the maximum number of iterations is 10000.
  - For an initial starting point (the par parameter), you can use a vector of all 1s
- Write a function prcomp_pc(X, k) which scales X and then uses prcomp() to extract and return the kth principal component of X.