customer_segments

Hamilton Hoxie Ackerman
hoxiea@gmail.com

January 21, 2016

1 Creating Customer Segments

In this project, you will analyze a dataset containing annual spending amounts for internal structure, to understand the variation in the different types of customers that a wholesale distributor interacts with.

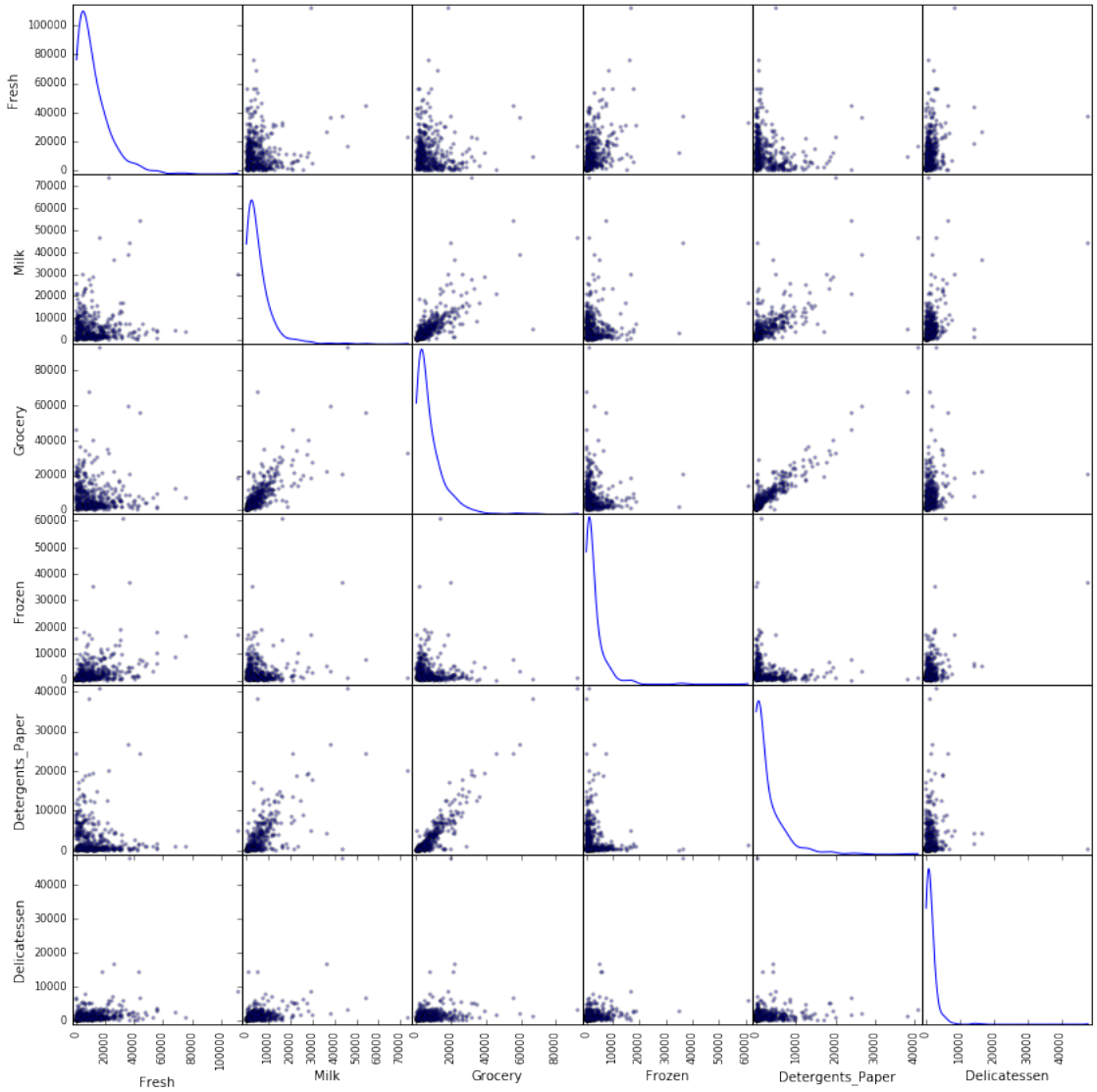
```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import pylab as pl

        %matplotlib inline   # Tell iPython to include plots inline in the notebook

        # Read dataset
        data = pd.read_csv("wholesale-customers.csv")
        print "Dataset has {} rows, {} columns".format(*data.shape)
        print data.head()  # print the first 5 rows
```

```
Dataset has 440 rows, 6 columns
   Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicatessen
0  12669  9656     7561     214              2674          1338
1   7057  9810     9568    1762              3293          1776
2   6353  8808     7684    2405              3516          7844
3  13265  1196     4221    6404               507          1788
4  22615  5410     7198    3915              1777          5185
```

1.1 Data Exploration

Before diving into PCA/ICA, I wanted to get a visual feel for the data. Each variable seems to span a fairly impressive range of values: Fresh, for example, takes on values between 3 and 100,000+! All the variables, in fact, span four or five orders of magnitude, so let's take a look and see how they're distributed across those ranges. Let's also try to visually understand the relationships between pairs of variables. Sounds like a job for a scatterplot matrix.

```
In [52]: from pandas.tools.plotting import scatter_matrix
         _ = scatter_matrix(data, alpha=0.3, figsize=(14,14), diagonal="kde")
```



Interesting! On the diagonal, we see that every variable is highly skewed right, with almost non-existant tails for the top 80+% of the ranges.

On the off-diagonals (the upper triangle and the lower triangle are symmetric across the diagonal), we see that: - some variables have relatively strong positive relationships (Grocery & Detergents_Paper, Milk & Grocery) - some variables have what appear to be non-linear inverse relationships (Fresh & Milk, Fresh & Grocery, Fresh & Detergents_Paper, Grocery & Frozen) - some variables have no obvious relationship at all (Fresh & Frozen, Fresh & Delicatessen, Detergents_Paper and Delicatessen).

For future reference, here's the correlation matrix for our data:

```
In [243]: print data.corr().applymap(lambda x: "{0:.3f}".format(x))    # map makes it all fit on one lin

Fresh   Milk Grocery  Frozen Detergents_Paper Delicatessen
Fresh              1.000  0.101  -0.012   0.346           -0.102        0.245
Milk               0.101  1.000   0.728   0.124            0.662        0.406
Grocery           -0.012  0.728   1.000  -0.040            0.925        0.205
Frozen             0.346  0.124  -0.040   1.000           -0.132        0.391
```

```
Detergents_Paper  -0.102  0.662   0.925  -0.132            1.000         0.069
Delicatessen       0.245  0.406   0.205   0.391            0.069         1.000
```

We see indeed that the pairwise correlations among Grocery, Detergents_Paper, and Milk are the strongest linear associations in this data, all positive. There are also weaker positive correlations among the other three features: Fresh, Frozen, and Delicatessen.

1.2 Feature Transformation

1) In this section you will be using PCA and ICA to start to understand the structure of the data. Before doing any computations, what do you think will show up in your computations? List one or two ideas for what might show up as the first PCA dimensions, or what type of vectors will show up as ICA dimensions.

PCA (Principal Components Analysis) is a mathematical projection of data onto a (typically) lower-dimensional space, where the axes of the new space are orthogonal (as is required to be axes) and represent the directions of maximal variance in the data. Given m factors, finding all m principal components amounts to a linear transformation of your data into a new feature space of the same dimension, from which your original data can be reconstructed and the new axes are the directions of maximal variation. If you take $k < m$ principal components, then you've projected your data into the k-dimensional space that minimizes the amount of information lost, i.e. you've come up with the best lower-dimensional representation of your data that you can, from an information theory perspective. This makes PCA quite useful for feature reduction, especially if a relatively small proportion of the principal components explain a relatively large amount of total variability in your data. The principal components themselves can also have informative interpretations in the context of the problem.

As we learned in the video lectures, PCA features tend to capture global concepts. It's not 100% clear what global concept the first principal component is most likely to capture, but if it's really going to try to maximize variance explained, then it could favor larger loadings for the variables with greater ranges: Fresh, Grocery, and Milk all have ranges of 70,000+, whereas Frozen, Detergents_Paper, and Delicatessen have ranges closer to 40,000-60,000. So perhaps the first component will capture something related to the magnitude of these variables, which would be some measure of how large or small the buyer is? This would be great, given that this project started because we didn't have a good understanding of the various types of buyers we supplied.

ICA (Independent Components Analysis), on the other hand, attempts to discover a different kind of underlying structure in the data. It assumes that the data were generated by some statistically independent, non-Gaussian signals that are hidden but responsible for generating the data that we observed. Then it decomposes your data into independent non-Gaussian signals as best it can, so that you can try to better understand the structure in your data.

In the video lectures, we learned that ICA tends to indentify more local features than PCA does: in image recognition, for example, the components are apparently often edges, since those are the somewhat independent components that work together to produce the images. In the case of our distribution data, I could imagine a few underlying concepts that would work together to generate purchases: the type of store (convenience store versus chain grocery store versus neighborhood grocery store versus coffee shop, etc.), the amount of business that the store does (small versus large), and maybe an urban/suburban/rural distinction.

1.2.1 PCA

```
In [56]: # Apply PCA with the same number of dimensions as variables in the dataset
         from sklearn.decomposition import PCA
         pca = PCA()   # default n_components == min(n_samples, n_features)
         pca.fit(data)

         # Print the components and the amount of variance in the data contained in each dimension
         print pca.components_
         print "Var, Each:      ", pca.explained_variance_ratio_
         print "Var, Cumulative:", np.cumsum(pca.explained_variance_ratio_)
```

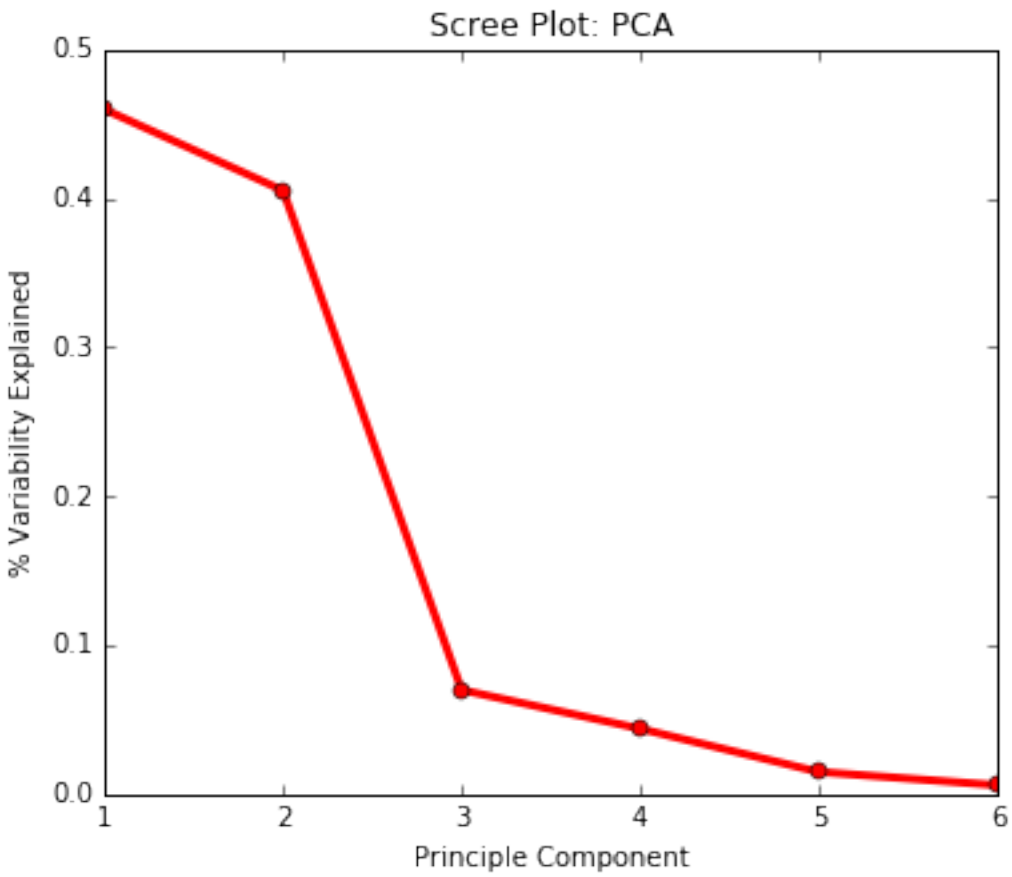
```
[[-0.97653685 -0.12118407 -0.06154039 -0.15236462  0.00705417 -0.06810471]
 [-0.11061386  0.51580216  0.76460638 -0.01872345  0.36535076  0.05707921]
 [-0.17855726  0.50988675 -0.27578088  0.71420037 -0.20440987  0.28321747]
 [-0.04187648 -0.64564047  0.37546049  0.64629232  0.14938013 -0.02039579]
 [ 0.015986    0.20323566 -0.1602915   0.22018612  0.20793016 -0.91707659]
 [-0.01576316  0.03349187  0.41093894 -0.01328898 -0.87128428 -0.26541687]]
Var, Each:       [ 0.45961362  0.40517227  0.07003008  0.04402344  0.01502212  0.00613848]
Var, Cumulative: [ 0.45961362  0.86478588  0.93481597  0.97883941  0.99386152  1.        ]
```

2) How quickly does the variance drop off by dimension? If you were to use PCA on this dataset, how many dimensions would you choose for your analysis? Why?

Well, in `explained_variance_ratio_`, we see that the first component captures 45.96% of the total variability in our original data, the second component captures 40.52% (86.48% cumulative), the third captures 7.00% of the data (93.48% cumulative), etc.

This dropoff is often visualized via a *scree plot*:

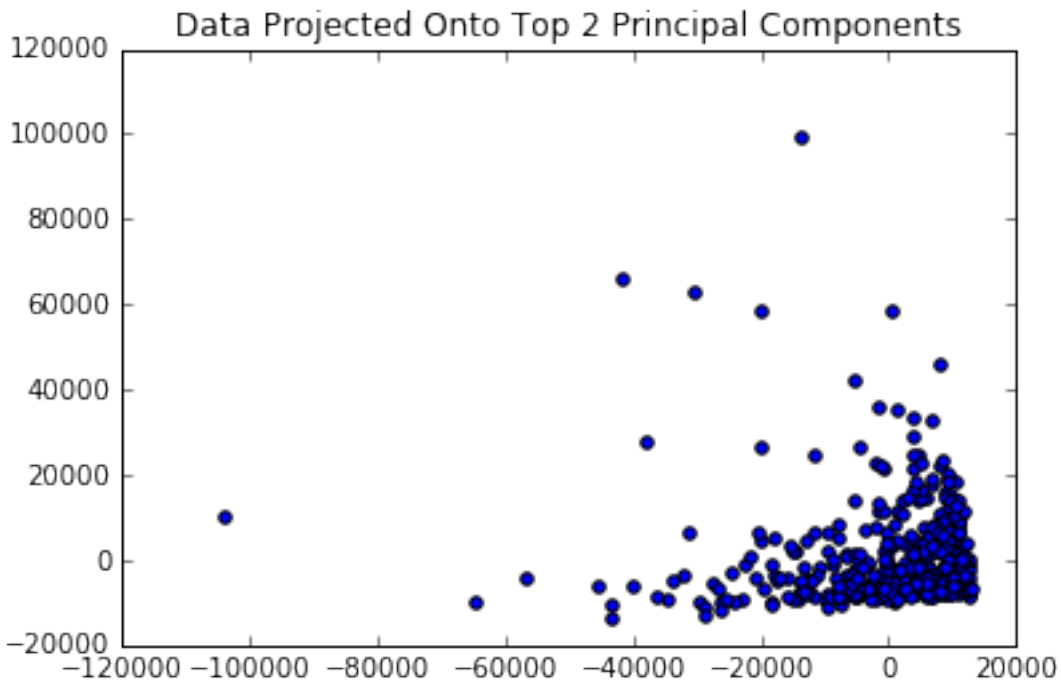
```
In [133]: # Inspired by https://stats.stackexchange.com/questions/12819/how-to-draw-a-scree-plot-in-pyt
          def scree_plot(fitted_pca, title=None):
              num_pcs = len(fitted_pca.explained_variance_ratio_)
              plt.figure(figsize=(6, 5))
              plt.plot(np.arange(num_pcs) + 1, fitted_pca.explained_variance_ratio_, 'ro-', linewidth=3
              if title is None:
                  plt.title('Scree Plot: PCA')
              else:
                  plt.title(title)
              plt.xlabel('Principle Component')
              plt.ylabel('% Variability Explained')
              plt.show()

          scree_plot(pca)
```



If I were going to use PCA to reduce the dimensionality of this dataset, I would probably begin by using the first two principal components. These components capture 86.48% of the total variability. (In contrast, if information was uniformly distributed among the six original features, we would capture only $2/6 = 33.33\%$ of total variability in the first two components.) You could make the argument to also use the third principal component, increasing our total-explained-variability to 93.48%, but the advantage of sticking to the first two is that easier and more intuitive to visualize things in two dimensions than in three.

In fact, let's take a look at the data, projected into the space defined by the top 2 principal components:

```
In [60]: pca_first2 = PCA(2)
         data_proj_top2 = pca_first2.fit_transform(data)
         plt.figure()
         plt.scatter(data_proj_top2[:,0], data_proj_top2[:,1])
         plt.title("Data Projected Onto Top 2 Principal Components")
         plt.show()
```



Interesting. I was hoping that some more visually distinct groups might emerge, but there are really only two relatively poorly defined groups that are obvious upon first glance: those in the main clump, and those scattered away from the clump. We'll explore this distribution of data more thoroughly later on.

3) What do the dimensions seem to represent? How can you use this information?

```
In [172]: def format_components(fitted_with_components_, columns, round=3, scale=1.0):
              for i, comp in enumerate(fitted_with_components_.components_):
                  markdown = ["Comp {}:".format(i+1)]
                  for col, val in zip(columns, comp):
                      if val > 0:
                          markdown.append(" + {:.{round}}\*{}".format(val*scale, col, round=round))
                      else:
                          markdown.append(" - {:.{round}}\*{}".format(abs(val*scale), col, round=round))
                  print "".join(markdown)

          # format_components(pca, data.columns.values)     # used in Markdown below
```

To answer this question, let's first spell out what these dimensions actually mean. Looking at the `pca.components_` above, and emphasizing the three coefficients with the largest magnitude for each principal component, we have:

PC1 = **-0.976*Fresh** - **0.121*Milk** - 0.062*Grocery - **0.152*Frozen** + 0.007*Detergents_Paper - 0.068*Delicatessen

The fact that Fresh dominates this linear combination so much (coefficient with magnitude 0.976, versus the next-largest coefficient of 0.152 on Frozen) tells us that the axis of maximal variation is dominated mostly by the effects of Fresh. So if you needed a single dimension on which to understand variations between customers, you should look at their Fresh value.

(It's probably not a coincidence that the variable with the largest range (Fresh) formed the basis of the direction of maximal variability in this data. If one variable spans a much larger range of values than other, there's likely to be more variability in the larger direction, so that variable will dominate the PC loadings.

It's common to normalize your data before performing PCA, though we weren't asked to do so in this project and there were forum questions that had the results of unnormalized PCA, so I didn't.)

PC2 = -0.111*Fresh + **0.516*Milk** + **0.765*Grocery** - 0.018*Frozen + **0.365*Detergents_Paper** - 0.057*Delicatessen

The second principal component is essentially a weighted average of Milk, Grocery, and Detergents_Paper, where Grocery gets a little bit more weight and Detergents_Paper gets a little bit less weight. These are exactly the three variables that were relatively highly positively correlated with each above. The fact that correlations between these variables is high means that, if we know the value for one of the variables, we can infer with good accuracy the value of the other variables. So in some sense, there's some redundancy there; in the extreme case, where two variables were perfectly correlated with each other, the second variable would be totally redundant.

As PCA tries to boil down our data into the directions of maximal variance, it's apparently noticed that these three variables are somewhat redundant, and that the axis of variation that they specify happens to capture a lot of variability in the data. So this second component tries to capture the variability given by these three variables in our original data.

The "larger range <-> larger coefficient" effect could be at play here, too: Grocery has the largest range of these three variables (~92k), followed by Milk (~73k) and finally by Detergents_Paper (~40k), and that happens to be the ranking of variable coefficient magnitudes as well.

We can use this information to better understand the structure of our data. Going into this exercise, it wasn't clear how our six variables were related or which variables were more important in understanding the essence of variation between customers. By examinings the first two principal components, which capture 86.48% of all variation in our data, we see that knowing where a customer lies on the Fresh spectrum actually tells us quite a bit about the customer, and furthermore that the variables Milk, Grocery, and Detergents_Paper capture similar notions and are next-most important in capturing variability in the data.

1.2.2 PCA: Take 2 (Normalized Data)

After performing the above analysis, I read the chapter covering unsupervised learning of "An Introduction to Statistical Learning" by James et al. In Section 10.2.3, the authors say:

> Because it is undesirable for the principal components obtained to depend on an arbitrary choice of scaling, we typically scale each variable to have standard devation one before we perform PCA.

They also provide an example of a dataset with four features. Running PCA on the unscaled data produces components that are highly skewed towards the features with the most and second-most variance, which is what I noted was happening in my unnormalized PCA above.

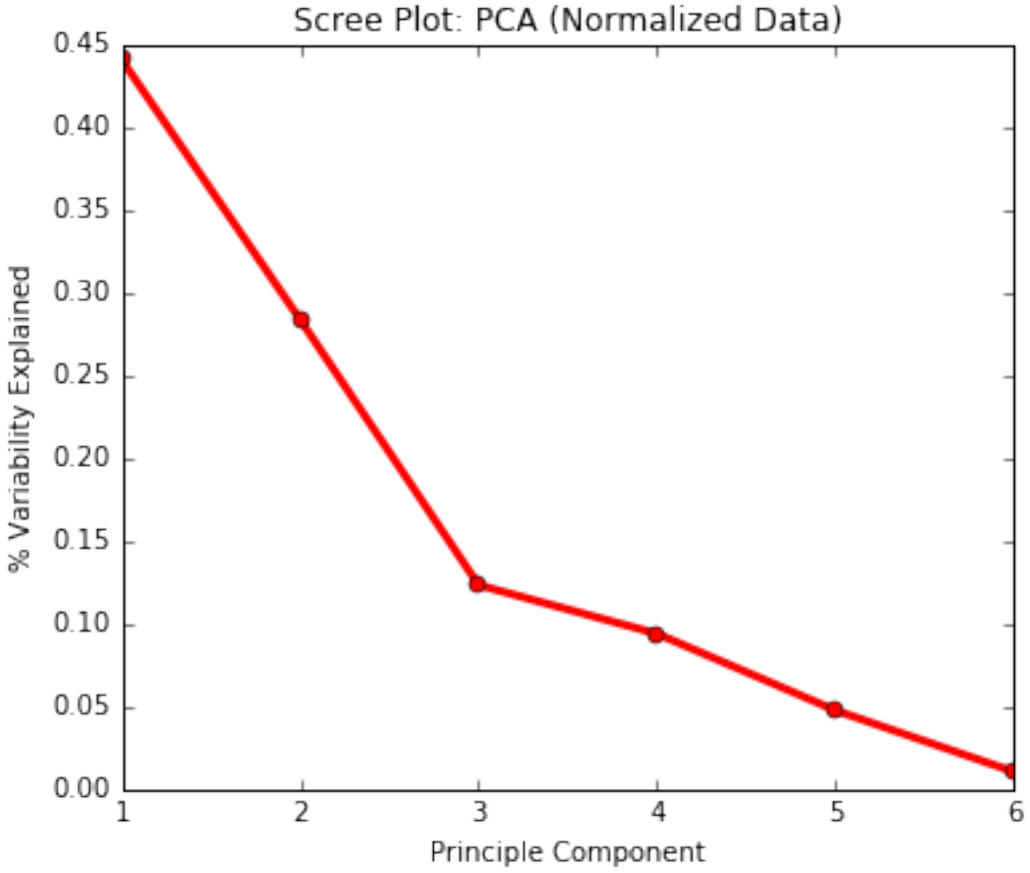
It was mentioned in the forums that, because the units of our customer variables are all the same, we shouldn't scale the variables before running PCA [1]. I can certainly appreciate that argument; we're comparing apples with apples in terms of the units, and the fact that our first principal component above was highly aligned with the variable with the greatest range actually does tell us something about our raw data. That said, in the AISL example, three of the four variables had the same units, and yet the variable with the greatest range still dominated the principal components when they ran it on unscaled data.

With all this in mind, I decided to normalize the data (mean=0, SD=1) and rerun PCA to see how the results changed.

In [135]:
```
from sklearn import preprocessing

data_normalized = preprocessing.scale(data)
pca_normalized = PCA()   # default n_components == min(n_samples, n_features)
pca_normalized.fit(data_normalized)

# Print the components and the amount of variance in the data contained in each dimension
print pca_normalized.components_
```

```
        print "Var, Each (Norm.):      ", pca_normalized.explained_variance_ratio_
        print "Var, Cumulative (Norm.):", np.cumsum(pca_normalized.explained_variance_ratio_)
        scree_plot(pca_normalized, "Scree Plot: PCA (Normalized Data)")
```

```
[[-0.04288396 -0.54511832 -0.57925635 -0.05118859 -0.5486402  -0.24868198]
 [-0.52793212 -0.08316765  0.14608818 -0.61127764  0.25523316 -0.50420705]
 [-0.81225657  0.06038798 -0.10838401  0.17838615 -0.13619225  0.52390412]
 [-0.23668559 -0.08718991  0.10598745  0.76868266  0.17174406 -0.55206472]
 [ 0.04868278 -0.82657929  0.31499943  0.02793224  0.33964012  0.31470051]
 [ 0.03602539  0.03804019 -0.72174458  0.01563715  0.68589373  0.07513412]]
Var, Each (Norm.):       [ 0.44082893  0.283764    0.12334413  0.09395504  0.04761272  0.01049519]
Var, Cumulative (Norm.): [ 0.44082893  0.72459292  0.84793705  0.94189209  0.98950481  1.        ]
```



After standardizing our variables so that they're centered (mean = 0) and have equal variation (SD = 1), we're (unsurprisingly) able to explain less of the variation in our data than we were previously with each number of principal components.

# Principal Components	Variation Explained, Unnormalized	Variation Explained, Normalized
1	45.96%	44.08%
2	86.48%	72.46%
3	93.48%	84.79%
4	97.88%	94.19%
5	99.39%	98.95%

Using 2 components still accounts for a reasonable amount of the total variability in the data (72.46%), but the constant slope of the scree plot implies that if you're going to use 2, you might as well use 3.

Let's take a look at the components and see if any meaning emerges:

```
In [168]: # format_components(pca_normalized, data.columns.values, 2)   # used in Markdown below
```

(The three coefficients with greatest magnitude are in bold.)

Comp 1: - 0.043*Fresh - 0.55***Milk** - 0.58***Grocery** - 0.051*Frozen - 0.55***Detergents_Paper** - 0.25*Delicatessen

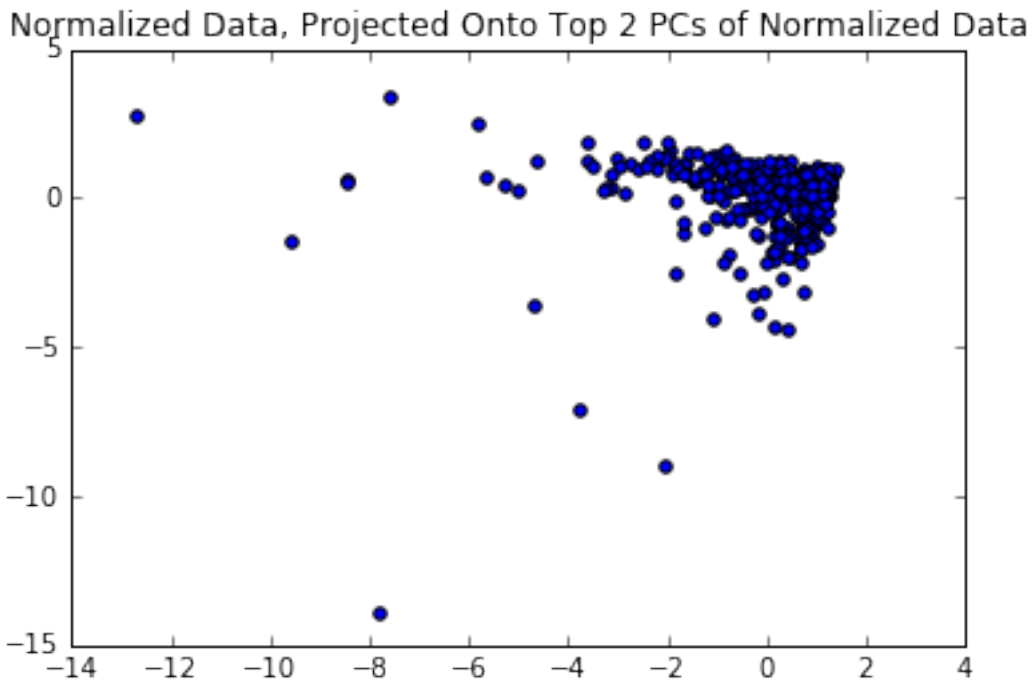
In the first component, we see that Fresh and Frozen are irrelevant, and that the axis of greatest variation is essentially a (negative) average of Milk, Grocery, and Detergents_Paper. These were the three variables that were selected to form the second principle component when our data was unnormalized; as noted in that discussion, these three variables are relatively highly positively correlated with each other. Correlation is unaffected by shifting and scaling, so that correlation has remained, and PCA in the normalized case has recognized the factor that these three variables are capturing and reduced their input down to a single dimension. This combination of features could reflect how much of a large grocery store a customer is: smaller convenience stores are unlikely to stock Detergents + Paper products, and probably won't go big on Groceries either.

Comp 2: - 0.53***Fresh** - 0.083*Milk + 0.15*Grocery - 0.61***Frozen** + 0.26*Detergents_Paper - 0.5***Delicatessen**

Interestingly, the second principal component is essentially a (negative) average of the three variables that were ignored by the first component: Fresh, Frozen, and Delicatessen. (There's also a non-trivial contribution by Detergents_Paper, but it's of a lesser magnitude.) Examining the correlation matrix above, we see that these three variables form the next-most correlated set of three variables, so again, PCA is recognizing the redundancy of individual variables and making composite variables out of related features. This feature could be picking up something along the lines of a corner store or deli.

Finally, we can again project our data onto the top components:

```
In [174]: data_normalized_proj_top2 = pca_first2.fit_transform(data_normalized)
          plt.figure()
          plt.scatter(data_proj_top2[:,0], data_proj_top2[:,1])
          plt.title("Normalized Data, Projected Onto Top 2 PCs of Normalized Data")
          plt.show()
```



This plot is regrettably similar to the previous projection plot. Instead of nicely contained clusters, we see the blob and the fringe. It'll be interesting to see what clustering algorithms pick up in these data.

1.2.3 ICA

```
In [177]: # Fit an ICA model to the data
          from sklearn.decomposition import FastICA

          data_centered = preprocessing.scale(data, with_std=False)  # Adjust the data to have center a
          np.testing.assert_array_almost_equal(data_centered.mean(), [0] * data_centered.shape[1])

          ica = FastICA()
          ica.fit(data_centered)

          # Print the independent components
          ica_components = (ica.components_ * 1e6).round(4)       # round to make everything fit on one
          print ica_components

          format_components(ica, data.columns.values, scale=1e6)
```

```
[[ -0.3865  -0.2195  -0.6006  -0.5221   0.51    18.0925]
 [ -0.1537  -9.8451   5.8107   0.3637  -3.3179   6.0571]
 [ -3.9759   0.8594   0.6261   0.6773  -2.0649   1.0424]
 [  0.8652   0.1405  -0.7741 -11.1462   0.5549   5.9522]
 [ -0.2995   2.3087  12.0564  -1.463  -28.2058  -5.7316]
 [ -0.2104   1.8852  -6.4311  -0.4106   0.8179   1.4563]]
Comp 1: - 0.386\*Fresh - 0.22\*Milk - 0.601\*Grocery - 0.522\*Frozen + 0.51\*Detergents_Paper + 18.1\*De
Comp 2: - 0.154\*Fresh - 9.85\*Milk + 5.81\*Grocery + 0.364\*Frozen - 3.32\*Detergents_Paper + 6.06\*Del
Comp 3: - 3.98\*Fresh + 0.859\*Milk + 0.626\*Grocery + 0.677\*Frozen - 2.06\*Detergents_Paper + 1.04\*De
Comp 4: + 0.865\*Fresh + 0.14\*Milk - 0.774\*Grocery - 11.1\*Frozen + 0.555\*Detergents_Paper + 5.95\*De
```

```
Comp 5: - 0.3\*Fresh + 2.31\*Milk + 12.1\*Grocery - 1.46\*Frozen - 28.2\*Detergents_Paper - 5.73\*Delica
Comp 6: - 0.21\*Fresh + 1.89\*Milk - 6.43\*Grocery - 0.411\*Frozen + 0.818\*Detergents_Paper + 1.46\*Del
```

4) For each vector in the ICA decomposition, write a sentence or two explaining what sort of object or property it corresponds to. What could these components be used for?

The six components are spelled out below; coefficients with relatively large magnitudes are emphasized. Four are interpreted.

Comp 1: - 0.386*Fresh - 0.22*Milk - 0.601*Grocery - 0.522*Frozen + 0.51*Detergents_Paper + **18.1*Delicatessen**

The relatively massive loading on Delicatessen suggests that we've captured the concept of a "deli" with this component.

Comp 2: - 0.154*Fresh **- 9.85*Milk** + **5.81*Grocery** + 0.364*Frozen - 3.32*Detergents_Paper + **6.06*Delicatessen**

This seems to be some kind of general-purpose grocery store: Milk, Groceries, Deli, and Detergents_Paper to a lesser degree.

Comp 4: + 0.865*Fresh + 0.14*Milk - 0.774*Grocery **- 11.1*Frozen** + 0.555*Detergents_Paper + **5.95*Delicatessen**

With its large emphasis on frozen foods and deli meats, this could be some kind of corner/convenience store.

Comp 5: - 0.3*Fresh + 2.31*Milk **+ 12.1*Grocery** - 1.46*Frozen **- 28.2*Detergents_Paper** - 5.73*Delicatessen

This is the only component with an appreciable loading for Detergents_Paper; that plus Grocery suggests some kind of grocery store that stocks many cleaning and paper goods.

1.3 Clustering

In this section you will choose either K Means clustering or Gaussian Mixed Models clustering, which implements expectation-maximization. Then you will sample elements from the clusters to understand their significance.

1.3.1 Choose a Cluster Type

5) What are the advantages of using K Means clustering or Gaussian Mixture Models?

Overview of K Means and Gaussian Mixture Models K Means (via the standard algorithm, Lloyd's algorithm) and Gaussian Mixture Models (via EM) are both iterative clustering algorithms.

Both algorithms require you, the user, to specify the number of clusters k ahead of time. Given that *a priori* specification, the algorithms then group the data into k clusters as best they can. How they "group" data differs between the two procedures:

K Means performs *hard* clustering: once the algorithm converges, each observation has been definitively assigned to belong to a single cluster. It's purely algorithmic, in the sense that there are no underlying probability distributions in K Means; it just finds k cluster centroids that minimize the sum of Euclidean distances from the observations to their closest centroid. All observations closest to a given centroid form a cluster. (Finding the centroid arrangement that globally minimizes this sum is technically an NP-hard problem; the algorithms we use are heuristics that provide approximations to the global optimum.)

GMMs perform *soft* clustering: it assumes that the data were generated by k underlying Gaussian distributions (multivariate in our case) and estimates the mean vectors and covariance matrices that will maximize the likelihood of the observed data. Once the algorithm converges, each observation has a certain probability of belonging to each of the k distributions, and cluster assignments can be made accordingly.

Both algorithms are iterative, and the iterations for both are quite similar. Both typically begin with a random initialization: the choice of the cluster centroids in K Means, and the choices of the mean vectors and covariance matrices. Then: - In K Means, observations are then assigned to the cluster that they're closest to (in Euclidean distance). Then, given those assignments, the centroids are updated to minimize the sum of Euclidean distances of each point to its centroid. These two steps alternate until the algorithm converges

and the centroids no longer change. - In GMMs, the conditional probability of each point belonging to each each Gaussian is calculated, given the current parameter estimates. Then given those probabilities, the centers and spreads are updated as weighted averages of the points, where the weights are the fixed cluster probabilities. These two steps alternate (this is the Expectation-Maximization (EM) algorithm) until the parameters don't change.

Neither algorithm is guaranteed to find the globally optimal clustering on a given run; due to the initialization process, these are technically random algorithms, and different initializations may produce different clusterings. It's common practice to run these procedures multiple times with different initializations to compare outcomes and see how reliable they are.

Advantages of K Means or Gaussian Mixture Models In some sense, soft clustering provides at least as much information as hard clustering: an observation with a cluster probability near 1 in a GMM is essentially as good as a definitive assignment in K Means, but the potential for an observation to have reasonable probabilities of belonging to multiple clusters captures the fact that some points might not fit squarely into one of the k clusters. As was mentioned in the video lectures, it also captures the fact that many random variables have infinite support, and therefore you can never know with 100% certainty which infinitely-supported random variable might have generated a given observation.

I think it unlikely that the highly skewed univariate distributions seen in the scatterplot matrix above could be mixtures of just a few Gaussian variables, which would be implied by the assumption that there are multivariate Gaussian subpopulations underlying our data, but Gaussian Mixture Models are known to work in practice even if the Normality assumption is violated. **To gain the benefits of soft clustering, I decided to use Gaussian Mixture Models.**

6) Below is some starter code to help you visualize some cluster data. The visualization is based on this demo from the sklearn documentation.

```
In [178]: # Import clustering modules
          from sklearn.mixture import GMM
```

```
In [67]: # First we reduce the data to two dimensions using PCA to capture variation
         reduced_data = data_proj_top2   # Did this already above
         print reduced_data[:10]  # print upto 10 elements
```

```
[[  -650.02212207   1585.51909007]
 [  4426.80497937   4042.45150884]
 [  4841.9987068    2578.762176  ]
 [  -990.34643689  -6279.80599663]
 [-10657.99873116  -2159.72581518]
 [  2765.96159271   -959.87072713]
 [   715.55089221  -2013.00226567]
 [  4474.58366697   1429.49697204]
 [  6712.09539718  -2205.90915598]
 [  4823.63435407  13480.55920489]]
```

```
In [239]: def plot_GMM_clusters(reduced_data, num_clusters):
              """
              Fit a GMM with num_clusters clusters to reduced_data
              Then plot, with regions colored
              Assumes that reduced_data is two-dimensional
              Returns the fitted GMM object
              """
              from sklearn.mixture import GMM

              clusters = GMM(num_clusters, covariance_type='full')
              clusters.fit(reduced_data)
```

```
    # Plot the decision boundary by building a mesh grid to populate a graph.
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    hx = (x_max-x_min)/1000.
    hy = (y_max-y_min)/1000.
    xx, yy = np.meshgrid(np.arange(x_min, x_max, hx), np.arange(y_min, y_max, hy))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = clusters.predict(np.c_[xx.ravel(), yy.ravel()])
    centroids = clusters.means_

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('Clustering on the wholesale grocery dataset (PCA-reduced data)\n'
              'Centroids are marked with white cross')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()

    return clusters
```

Before we run plot_GMM_clusters to examine the clusters that Gaussian Mixture Models pick up, we need to answer the question...

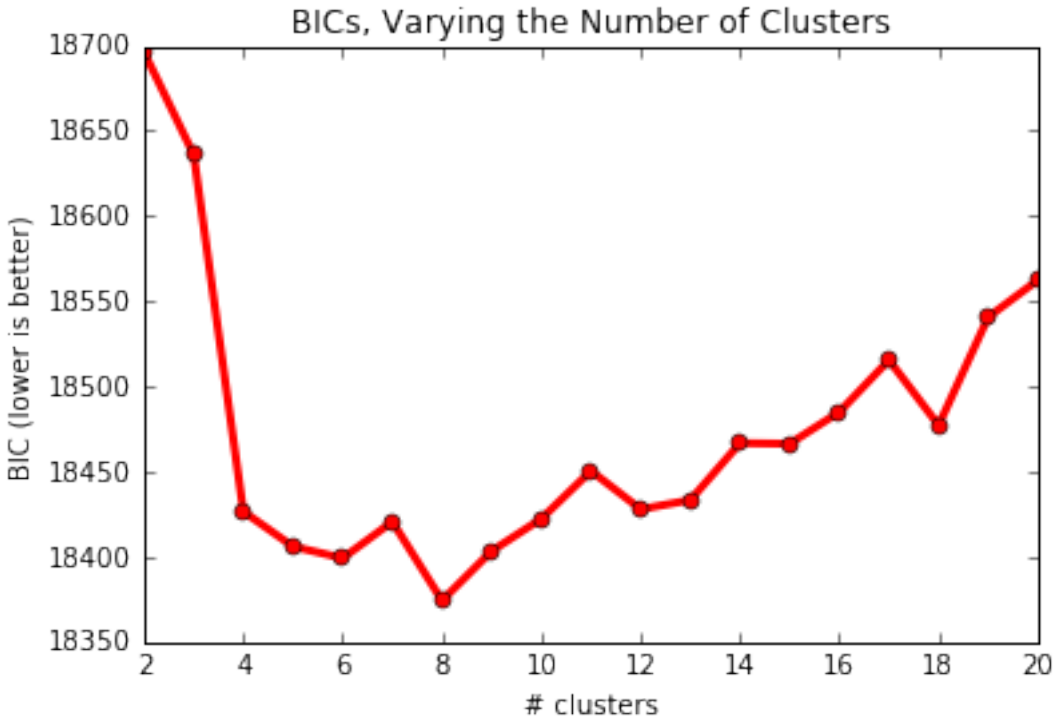
1.3.2 How Many Clusters?

One of the most crucial choices you make when clustering is the number of clusters that you tell these algorithms to form. There are many different heuristics that have been proposed over the years to help guide this choice. Based on the excellent discussion given here reinforced by the Clustering chapter of Elements of Statistical Learning (Hastie et al. 2009), I decided to use the Bayesian Information Criterion as my clustering criterion. According to Hastie et al., "choosing the model with minimum BIC is equivalent to choosing the model with largest (approximate) posterior probability" (p. 234).

Implementing this idea was fairly straightforward: for the number of clusters k in [2..20], I fit a GMM, saved the corresponding BIC, and then plotted those BICs as a function of k.

```
In [190]: # Try to figure out how many clusters should be used, via Bayesian Information Criterion (BIC
          bics = []
          min_k = 2
          max_k = 21
          for k in range(min_k, max_k):
              clusters = GMM(k, covariance_type='full')
              clusters.fit(reduced_data)
```

```
    bics.append(clusters.bic(reduced_data))

plt.plot(np.arange(min_k, max_k), bics, 'ro-', linewidth=3)
plt.title('BICs, Varying the Number of Clusters')
plt.xlabel('# clusters')
plt.ylabel('BIC (lower is better)')
plt.show()
```



It seems like k=6 and k=8 are both potentially interesting values to consider. Let's take a look...

```
In [241]: gmm8 = plot_GMM_clusters(reduced_data, 8)
```


```
In [240]: gmm6 = plot_GMM_clusters(reduced_data, 6)
```

Of these two, I prefer the six-cluster interpretation better: the clusters essentially split the data into {low, medium, high} values for the first and second principal component. (In the 8-cluster case, there's that

weird cluster all the way over to the left side of the image that doesn't seem to be capturing much. And reducing the number of clusters to 7 actually doesn't eliminate the outlier.) We'll explore these clusters in the next section.

7) What are the central objects in each cluster? Describe them as customers.

Here are the actual central objects:

```
In [242]: print gmm6.means_

[[ -7562.29478011  -2500.09919329]
 [  9437.91056691   3606.28289813]
 [-47375.4884958   -7149.56893789]
 [-20788.64888205  43077.28006753]
 [  2670.89267311  14851.23087061]
 [  2902.8610809   -7126.2024584 ]]
```

To describe these points as customers, we need to refer back to the principal components (from the unnormalized data): PC1 was dominated by Fresh, while PC2 was essentially a weighted average of Grocery, Milk, and Detergents_Paper (all fairly correlated variables).

Since our units were monetary, that suggests that variation horizontally indicates how much Fresh business you represent, with greater values to the left and smaller values to the right, since the Fresh coefficient was negative. The pink cluster, for example, are high-Fresh customers, and Fresh consumption drops off as we move right through the blue and red clusters. Note that these groups are all low consumers of Grocery, Milk, and/or Detergents_Paper, and that there really aren't any stores that score highly on both axes; I found this interesting, given the existence of Walmart Superstores and the like.

Variation in the vertical direction represents increasing consumption of Grocery, Milk, and/or Detergents_Paper (since the coefficients on all three variables in the principal component were positive.) So the green cluster is a small store that focuses more on these kinds of goods than it does on Fresh items, with increasing amounts of Grocery/Milk/Detergents_Paper as we move through the purple and orange clusters.

So when the prompt says that "mom and pop shops had a hard time dealing with the change," that could certainly be the case. But this analysis suggests that we can actually even get more precise with our distinctions: in the "small store" category, we actually have two different groups of stores that are differentiated by the amounts of Fresh and Grocery/Milk/Detergents_Paper business. Maybe one kind of small store did fine with the change, but the other suffered; with these clustering insights in hand, we'd be in a much better position to understand what was happening with our customer base.

1.3.3 Conclusions

8) Which of these techniques did you feel gave you the most insight into the data?

I thought that PCA, in conjunction with the GMM clustering algorithm (both discussed in detail above), was the most useful method for better understanding the structure of this dataset. Finding the axes of greatest variation is a generally useful thing to do, and then it was cool to see that one of the best clustering arrangements (via BIC) corresponded to what were essentially low, medium, and high values of either the first or second (orthogonal) principal component. In addition to providing us with a summary of our customer base that's easy to think about, understand, and share, this also gives us the ability to do some great things for our customers, as discussed in the next two answers.

9) How would you use that technique to help the company design new experiments?

In the introduction, we learned that the company made a change that worked well for some of their customers but not others; the conclusion was that they needed to better understand the kinds of customers they actually had, so that it would be easier to understand the effects of future changes on their customer base. PCA + GMM came up with a reasonable partitioning of the space of customers into six different groups, so if I were this company and I was considering a change down the road, I would make sure to consult customers belonging to all of these groups; this should provide much better coverage than a simple random sample would. Or, if I wanted to be more aggressive with a change and implement it with some but not all customers via an A/B test, I would make sure that I was collecting enough data about how customers in all six clusters responded to both A and B.

10) How would you use that data to help you predict future customer needs?

If we want to make predictions about future customers, we now have a more solid framework for doing so. When evaluating the needs of a new customer, we could first figure out which cluster the customer belongs to (by projecting his six main features on the subspace defined by the principal components, and then evaluating the pdf of each multivariate Gaussian cluster at that point to find the relative probability of the customer belonging to each cluster). Once the new customer has been grouped with a cluster, we can then focus on the other customers in that cluster to understand what might work well for the new customer.

For example, if 50% of customers in the cluster prefer some kind of packaging, delivery service, or billing schedule, versus only 10% of customers in the total population exhibiting that preference, it seems much more likely that our new customer would prefer that offering as well, and we could confidently discuss the option with the customer. On the other hand, if only 3% of customers in the cluster use a feature versus 25% in the general population, then we probably don't need to waste time bringing it up in a meeting. (Note that this is a supervised learning problem; the target is a binary like/dislike.)

This is essentially a recommendation system: with clusters, we suddenly have a concept of "similar customers," so we can make predictions and recommendations based on what's working well for other relevant customers.