customer_segments

February 1, 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.

Instructions:

- Run each code block below by pressing **Shift+Enter**, making sure to implement any steps marked with a TODO.
- Answer each question in the space provided by editing the blocks labeled "Answer:".
- When you are done, submit the completed notebook (.ipynb) with all code blocks executed, as well as a .pdf version (File > Download as).

In [124]:
```
# Import libraries: NumPy, pandas, matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

# 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 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.

Answer: From Rubric: At least one idea for what patterns might arise as components in PCA and ICA has been written.

As the project is to look at annual spending of customers of a wholesale distributor, what should show up are the different groups or types that the customers fall into (if any) based upon their spending patterns ie. what categories of merchandise do they buy from the wholesale distributor. It is likely that a supermarket will buy different types and quantities compared to a restaurant or bar etc.

List one or two ideas for what might show up as the first PCA dimensions: PCA finds directions of maximal variance in the data. From looking at the data displayed above, it would appear that the "Fresh" and "Milk" features are the most likely to be the main principle components as they have the largest range in values. The obtained principal components will point in the direction of maximal variance.

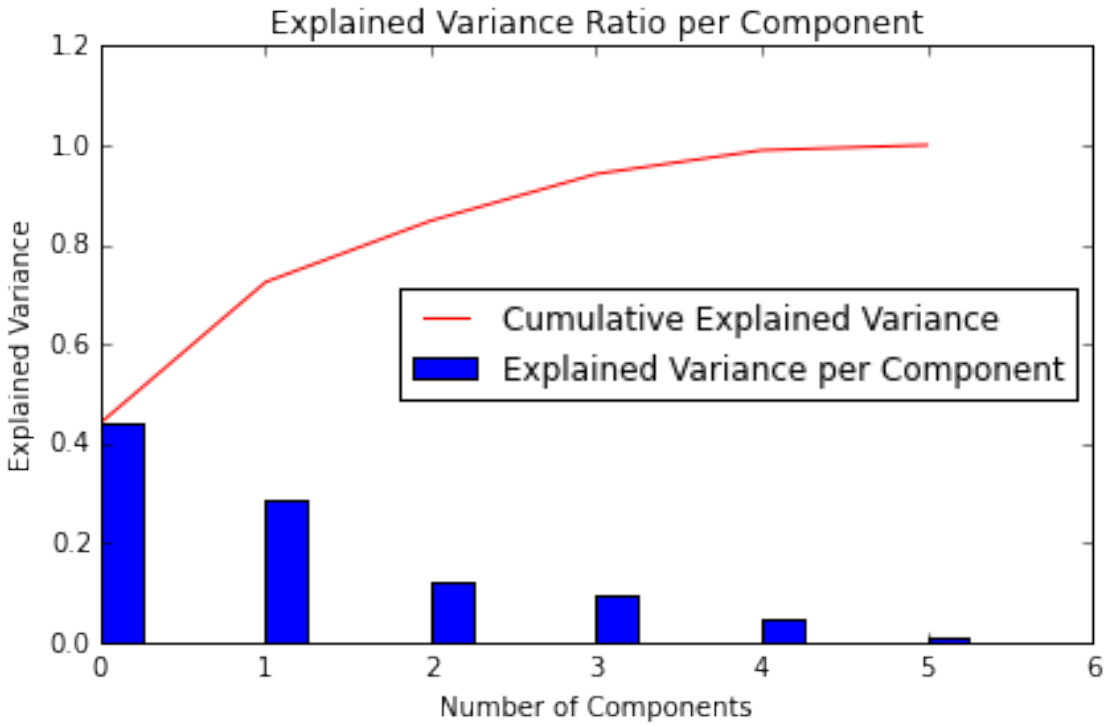
What type of vectors will show up as ICA dimensions ? The component vectors output by ICA will be independent of each other, each of which are composed of relative amounts of each feature. Each component will be unique which may indicate a type or category of customer. Looking at what is purchased in each component may offer some insight as to what type of customer each component represents.

1.1.1 PCA

In [125]:
```
# TODO: Apply PCA with the same number of dimensions as variables in the dataset
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize features by removing the mean and scaling to unit variance
scaled_data = StandardScaler().fit_transform(data)

# build PCA model
pca = PCA()

# apply PCA to scaled data
pca.fit(scaled_data)

# Print the components and the amount of variance in the data contained in each dimension
print "pca.components_ (eigenvectors):"
print pca.components_
print " "
print "pca.explained_variance_ratio_ :"
print pca.explained_variance_ratio_
print " "

# Generate and plot graph of explained variance per component and cummulative explained varia
fig, ax = plt.subplots()
plt.title('Explained Variance Ratio per Component ')
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
comps = [x for x in range(0,6)]
plt.bar(comps, pca.explained_variance_ratio_, 0.25, color='b', label='Explained Variance per
plt.plot(pca.explained_variance_ratio_.cumsum(), color='r', label='Cumulative Explained Varia
plt.legend(loc=7)
plt.tight_layout()
plt.show()

print "Cummulative Sum of Priniple Component Explained Variances:"
[sum(pca.explained_variance_ratio_[:i+1]) for i in xrange(len(pca.explained_variance_ratio_))]
```

```
pca.components_ (eigenvectors):
[[-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]]
```

```
pca.explained_variance_ratio_ :
[ 0.44082893  0.283764    0.12334413  0.09395504  0.04761272  0.01049519]
```




```
Cummulative Sum of Priniple Component Explained Variances:

Out[125]: [0.44082892881128016,
           0.7245929240774498,
           0.84793705304531442,
           0.94189209057502954,
           0.98950481458191641,
           1.0000000000000002]
```

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?

Answer: From Rubric: The variance explained by each dimension is appropriately plotted and explained. A reasonable cutoff point for use has been explained.

The variance drops of quite quickly as you move from the first dimension to the third, the drop being approximately 16%, from both first (44%) to second(28%) and second(28%) to third(12%). After the third principle component the dropoff slows to 3% between the third(12%) and forth(9%) components and 5% between forth(9%) and fifth(4%) and finally 3% between the last two components.

If I were to use PCA on this dataset I would use n_components (number of dimensions) equal to 4. I choose this number as it will account for over 94% of the explained variance in the dataset. Typically when using PCA for dimension reduction you retain the number of features/components that explain at least 90%-95% of the variance.

```
In [126]: # Display each eigenvector and give it a label
          print " "
          print "Eigenvectors for Question 3 Discussion:"
```

```
print "A.",pca.components_[0]
print "B.",pca.components_[1]
print "C.",pca.components_[2]
print "D.",pca.components_[3]
print "E.",pca.components_[4]
print "F.",pca.components_[5]
```

```
Eigenvectors for Question 3 Discussion:
A. [-0.04288396 -0.54511832 -0.57925635 -0.05118859 -0.5486402  -0.24868198]
B. [-0.52793212 -0.08316765  0.14608818 -0.61127764  0.25523316 -0.50420705]
C. [-0.81225657  0.06038798 -0.10838401  0.17838615 -0.13619225  0.52390412]
D. [-0.23668559 -0.08718991  0.10598745  0.76868266  0.17174406 -0.55206472]
E. [ 0.04868278 -0.82657929  0.31499943  0.02793224  0.33964012  0.31470051]
F. [ 0.03602539  0.03804019 -0.72174458  0.01563715  0.68589373  0.07513412]
```

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

Answer: From Rubric: Basis vectors for at least two PCA dimensions have been interpreted correctly and their significance has been discussed.

Using the eigenvectors (see above), I have labelled them A-F.

Eigenvectors are the linear combinations of the original variables, they describe how those variables "contribute" to each factor axis, ie. they show how variable contributions are weighted or linearly transformed in this new space. So taking C as an example, it has heavy weights for "Fresh" (0.81) and "Delicatessen" (0.52) so maybe that customer or group of customers buys mostly fresh food and delicatessen items, perhaps the customers are either restaurants, delicatessens or catering companies. Using B as another example, the heaviest weights are seen for "Frozen" (0.61), "Fresh" (0.52) and "Delicatessen" (0.50), the other categories show moderate weights except for "Milk", this suggests the customer or customer group maybe supermarkets. Without proper domain knowledge one cannot be too specific.

1.1.2 ICA

In [127]:
```
# TODO: Fit an ICA model to the data
# Note: Adjust the data to have center at the origin first!
from sklearn.decomposition import FastICA

# build ICA model, ensure repeatable values
ica = FastICA(random_state=42)

# Standardize features by removing the mean and scaling to unit variance
scaled_data = StandardScaler().fit_transform(data)

# apply ICA to scaled data
S_ = ica.fit(scaled_data)

# Print the independent components
print "ica.components_: "
print ica.components_
```

```
ica.components_:
[[-0.0109083  -0.00108579  0.00730777  0.05405594 -0.00254136 -0.01675677]
 [ 0.00253788 -0.0123283   0.06912878  0.00142375 -0.01374853 -0.00544097]
 [-0.00490605 -0.00153897 -0.00562146 -0.002525    0.00238444  0.05092947]
 [-0.00336282  0.01863001  0.10899024 -0.00723244 -0.13338644 -0.0160228 ]
 [-0.05026646  0.00647203  0.00748246  0.00322414 -0.01147139  0.0027079 ]
 [-0.00193854 -0.07245463  0.05647623  0.0016736  -0.0171404   0.01695592]]
```

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?

Answer: From Rubric: Basis vectors for at least 4 dimensions of ICA have been interpreted correctly and their significance has been discussed.

The goal of ICA is to recover independent sources when given only observations that are unknown mixtures of the unobservable independent source signals. The independent souces are called independent components, they can be used to represent hidden information in the observable data. The components may reflect signals of interest or they may alternatively be dominated by artifacts/noise and its up to the user to determine which are of interest eg. task related, this is where domain knowledge is an advantage.

ICA models observations as a linear combination of latent feature variables or components which are chosen to be as statistically independent as possible. These components or rows of the ica.components matrix, are independent in the sense that the weightings of each feature in a component reflect samplings of independent random variables. The rows of the ica.components matrix describe the contributions of individual feature categories ie. Fresh, Milk, Grocery, Frozen, Detergent/Paper and Delicatessen to each component. Each component contains an expression value for each feature category. ICA produces components with unit variance and zero mean, so the contributions of each feature category are relative.

Now taking each component and looking at feature categories with the largest relative weights I observe the following:

(1) Frozen (0.054), Delicatessen (0.017) and Fresh (0.011), other feature categories are significantly less, with Frozen at least 3x times as large as the next largest. Perhaps this is some sort of convenience store that specialises in frozen foods/meals.

(2) Grocery (0.069), Detergent/Paper (0.014) and Milk (0.012), other feature categories are significantly less, with Grocery almost 5x times as large as the next largest. This could be a normal grocery store or supermarket.

(3) Delicatessen (0.051), other feature categories are significantly less. This could be a delicatessen or sandwich shop.

(4) Detergent/Paper (0.133) and Grocery (0.108), other feature categories are significantly less. Given the much larger weights compared to other components, perhaps this is a bulk/discount retailer.

(5) Fresh (0.050) and Detergent/Paper (0.011), other features categories are significantly less, with Fresh over 4x times as large as the next largest. This could be a restaurant or eatery of some kind or a store that specialises in selling fresh food/produce.

(6) Milk (0.072), Grocery (0.056), with Detergent/Paper and Delicatessen (0.017). Milk and groceries are the major components here, given that Milk is the largest component it could indicate that the customer type might be a neighbourhood convenience store.

From above we can see that these components can be used as away to categorise the different customer types the distributor has. However interpreting these vectors is far from intuitive in the absence of domain knowledge.

1.2 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.

1.2.1 Choose a Cluster Type

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

Answer: From Rubric: Gaussian Mixtures and K Means have been compared. Student makes a choice which is justified based on characteristics of the algorithms.

K-Means is a clustering algorithm ie. a way of partitioning a set of data points into "clusters" or sets of data points which are similiar to one another. It works by iteratively reassigning data points to clusters and computing cluster centers based on the avaerage of the point locations. It is commonly used for vector quantization and as an initialisation for Gaussian Mixture models. The main advantages of K-Means is that it is fast, robust and easy to understand. There are always K clusters, with at least one item in each cluster, the clusters are non-hierarchical and don't overlap. Every member of a cluster is closer to its cluster than any other cluster. Gives best results when dataset points are distinct or well separated from each other.

Gaussian Mixture models are a probabilistic model commonly used for clustering/partitioning a set of data points into a set of clusters, where data points within a cluster are similiar to one another. It assumes that all the data points are generated from a mixture of a finite number of gaussian distributions with unknown parameters. Each cluster can be mathematically represented by a gaussian distribution (continuous). The entire dataset is therefore modelled by a mixture of this distribution. The algorithm that the GMM classifier uses to find the mixture of gaussians that can model the dataset is called EM (expectation-maximisation). Some of the advantages of GMM relate to its probabilistic method for obtaining a classification of the data points, in that this can be used to obtain density estimation for each cluster and the probabilities can be used to interpret suspect classifications. Other advantages include, its fast in that EM is the fastest algorithm for learning mixture models, and the model is flexible in that there are a number of parameters available for model tuning.

I choose K-Means to perform clustering given its speed and ease of use.

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

From Rubric: More than one choice of number of clusters has been tried out. Elements from each cluster have been sampled and interpreted.

I have run K-Means with n_clusters=2 and n_clusters=7, see plot figures 1 and 2 below. I have also run KMeans with n_clusters=7 after applying a logarithmic transformation to the original data, see plot figure 3 below.

```
In [128]: s6_data= {'Centroid_1': [-3.6743, 0.4186],
                    'Centroid_2': [0.4075, -0.0464],
                    'Fresh': [12084, 11257],
                    'Milk': [4152, 20223],
                    'Groc': [5595, 28663],
                    'Frozen': [3080, 2997],
                    'Det/Paper': [1712, 13140],
                    'Deli': [1309, 2414],
                    'Total': [27935, 79666]}
          s6_info = pd.DataFrame(s6_data, columns=['Centroid_1', 'Centroid_2', 'Fresh', 'Milk', 'Groc',
          print "Table of Information for K-Means Plot with n_clusters=2:"
          print s6_info
```

```
Table of Information for K-Means Plot with n_clusters=2:
   Centroid_1  Centroid_2  Fresh   Milk   Groc  Frozen  Det/Paper  Deli  Total
0     -3.6743      0.4075  12084   4152   5595    3080       1712  1309  27935
1      0.4186     -0.0464  11257  20223  28663    2997      13140  2414  79666
```

6) Continued

For the K-Means plot using n_clusters=2 We have a vector for each cluster (the centroids) with weights for each of the two features, as the data has been PCA reduced to 2 principle components. The centroids for each of the two clusters are displayed in the table above along with the average spends in each category and the average total spend. Means were obtained by applying the cluster predictions for each datapoint from KMeans to the original dataset.

The first number for the first cluster centroid ie. 3.6743, is the mean of the first principle component for the observations in that cluster, and looking at the average sales figures we can see that the amount for the first feature 'Fresh' is significantly larger than any other feature. Comparing the spends to the other cluster,

the spending by customers in this cluster is significantly less on average, indicating that these customers are small/medium sized businesses like a local store or restaurant.

Looking at the centroid values for the second cluster we see much lower values for each component compared with the first cluster. This is probably due to the fact that the average spend for 'Fresh' is low relative to 'Milk', 'Groceries' and 'Detergent/Paper' in the second cluster. The second component is quite small, again probably due to more than 50% of the spending being outside the first two features. As mentioned below spending across each category/feature when compared to the first cluster is about the same or significantly higher, indicating that the customers in this clsuter are large, and could be large supermarkets or a chain of shops.

In [129]:
```
# Import clustering modules
from sklearn.cluster import KMeans
from sklearn.mixture import GMM
from sklearn.preprocessing import FunctionTransformer
```

In [130]:
```
# TODO: First we reduce the data to two dimensions using PCA to capture variation

# create a log transformer
transformer = FunctionTransformer(np.log1p)
# apply log transform to original data
t_data = transformer.transform(data)

# # Standardize features by removing the mean and scaling to unit variance
scaled_data = StandardScaler().fit_transform(data)
# apply same standardisation to transformed data
t_scaled_data = StandardScaler().fit_transform(t_data)

# build PCA model
pca = PCA()

# want to keep only 2 features
pca.n_components = 2

# apply PCA to scaled data
reduced_data = pca.fit_transform(scaled_data)
# apply PCA  to transformed scaled data
t_reduced_data = pca.fit_transform(t_scaled_data)

print "reduced_data (first 10 elements):"
print reduced_data[:10]
print " "
print "t_reduced_data (first 10 elements):"
print t_reduced_data[:10] # print upto 10 elements
print " "
print "Eigenvectors for PCA with n_components=2:"
print pca.components_
```

```
reduced_data (first 10 elements):
[[-0.19329055  0.30509996]
 [-0.4344199   0.32841262]
 [-0.81114323 -0.8150957 ]
 [ 0.77864783 -0.65275373]
 [-0.16628726 -1.27143372]
 [ 0.15616993  0.29514099]
 [ 0.3352884   0.52500326]
```

```
 [-0.14058643  0.23099269]
 [ 0.51731954  0.65936283]
 [-1.59210908  0.74101133]]

t_reduced_data (first 10 elements):
[[ 1.38038279 -0.30472683]
 [ 1.43886964  0.53746823]
 [ 1.51345256  1.23256453]
 [-0.8251054   1.21078962]
 [ 0.80289711  1.76604596]
 [ 0.88616546  0.12989418]
 [ 0.60325602 -0.46154812]
 [ 1.15264871  0.60767615]
 [ 0.53001207 -0.63013806]
 [ 2.19686557  0.32604951]]

Eigenvectors for PCA with n_components=2:
[[-0.10524973  0.54232857  0.57147152 -0.13790969  0.55064187  0.21446903]
 [ 0.59035923  0.1329114  -0.00575087  0.58977789 -0.07082164  0.53002246]]
```

In [131]:
```
# TODO: Implement your clustering algorithm here, and fit it to the reduced data for visualiz
# The visualizer below assumes your clustering object is named 'clusters'

# use Bayesian information criterion from GMM to try and get a handle on how many clusters th
# be in the dataset...we are looking for the lowest value...
for i in range(2,20):
    x_clusters = GMM(n_components=i, covariance_type='diag', verbose=0)
    x_clusters.fit(reduced_data)
    print ("i=",i," bic=",x_clusters.bic(reduced_data))

    def Build_Model(n_clusters, reduced_data):
        # build KMeans model, to form n clusters and centroids
        clusters = KMeans(n_clusters=n_clusters)

        # apply KMeans to reduced data (from PCA above)
        clusters.fit_transform(reduced_data)

        # display model parameters
        print " "
        print "KMeans model parameters:"
        print clusters
        # return model to caller
        return clusters
```

```
('i=', 2, ' bic=', 2597.7431641431704)
('i=', 3, ' bic=', 2406.1560411103223)
('i=', 4, ' bic=', 2415.3213551291747)
('i=', 5, ' bic=', 2308.3564090553682)
('i=', 6, ' bic=', 2309.5976540030711)
('i=', 7, ' bic=', 2297.1252271695971)
('i=', 8, ' bic=', 2305.4348680725757)
('i=', 9, ' bic=', 2327.7663897127791)
('i=', 10, ' bic=', 2341.0372215969473)
('i=', 11, ' bic=', 2358.6674783311687)
('i=', 12, ' bic=', 2371.8436082135431)
```

```
('i=', 13, ' bic=', 2397.8991287274857)
('i=', 14, ' bic=', 2408.8267436545325)
('i=', 15, ' bic=', 2432.0038186637016)
('i=', 16, ' bic=', 2452.8014251676314)
('i=', 17, ' bic=', 2478.6887184122402)
('i=', 18, ' bic=', 2511.3370363344884)
('i=', 19, ' bic=', 2537.977292903779)
```

In [16]:
```
def Decision_Boundary(reduced_data, clusters):
    # 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()])
    # return co-ordinates and labels to caller
    return x_min, x_max, y_min, y_max, xx, yy, Z
```

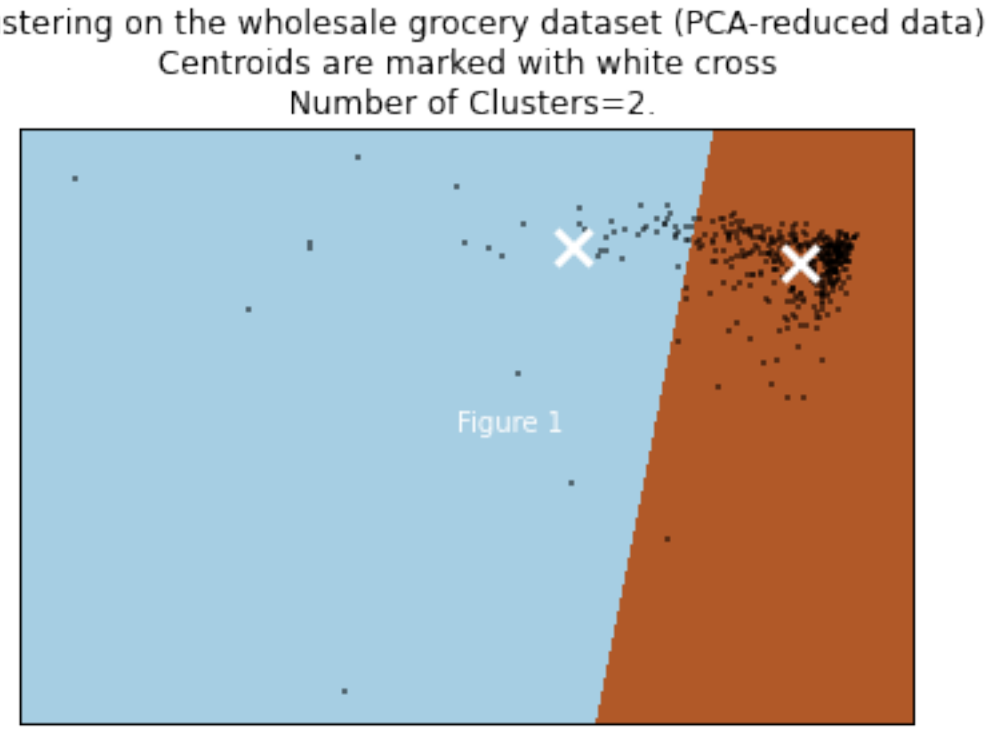
In [19]:
```
# TODO: Find the centroids for KMeans or the cluster means for GMM
def Print_Centroids(clusters):
    # get the co-ordinates of cluster centers
    centroids = clusters.cluster_centers_

    # print cluster centers
    print "Cluster centers:"
    print centroids
    # return centroids to caller
    return centroids
```

In [134]:
```
def Show_Scatter_Plot(x_min, x_max, y_min, y_max, xx, yy, Z, reduced_data, centroids, nClu
# 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\n Number of Clusters=' + nClusters +
    plt.annotate('Figure '+figureCount, xy=(2.45, 2.50),
         xytext=(-5.8, -5.5), color='w')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

```
    def Produce_Stats(data, clusters, reduced_data):
        # create a dataframe from original data
        df = pd.DataFrame(data)
        # ensure new columns not present
        if 'total_spend' in df.columns:
            del df['total_spend']
        if 'cluster' in df.columns:
            del df['cluster']
        # add a column called 'total_spend' from summ of all features totals
        df['total_spend'] = df.sum(axis=1)
        # add a column called cluster, and fill with the predicted cluster the datapoint belo
        df['cluster'] = clusters.fit_predict(reduced_data)
        # display how many datapoints are in each cluster
        print "Number of datapoints in each cluster:"
        print df.cluster.value_counts()
        # for each predicted cluster produce a mean spend for each feature/category
        print " "
        for cluster, features in df.groupby("cluster"):
            print cluster, features.mean()

# keep track of the number of figures for graph annotations
figure_counter = 1
# want to look at n_clusters=2 and n_clusters=7
number_of_clusters = [2, 7]
# for each cluster...
for i in number_of_clusters:
        # Build KMeans model
        clusters = Build_Model(i, reduced_data)
        # Calculate decision boundary
        x_min, x_max, y_min, y_max, xx, yy, Z = Decision_Boundary(reduced_data, clusters)
        # Calculate and display centroids
        centroids = Print_Centroids(clusters)
        # Produce and display a plot
        Show_Scatter_Plot(x_min, x_max, y_min, y_max, xx, yy, Z, reduced_data, centroids, str
        # Calculate and display average spend statistics per cluster
        Produce_Stats(data, clusters, reduced_data)
        # increment count of figures produced
        figure_counter = figure_counter + 1

# want to look at n_clusters=7 only
number_of_clusters = [7]
# for each cluster...
for i in number_of_clusters:
        # Build KMeans model using transformed data
        clusters = Build_Model(i, t_reduced_data)
        # Calculate decision boundary
        x_min, x_max, y_min, y_max, xx, yy, Z = Decision_Boundary(t_reduced_data, clusters)
        # Calculate and display centroids
        centroids = Print_Centroids(clusters)
        # Produce and display a plot
        Show_Scatter_Plot(x_min, x_max, y_min, y_max, xx, yy, Z, t_reduced_data, centroids,
                          str(i) + ", Log Transformed Data", str(figure_counter))
        # Calculate and display average spend statistics per cluster
```

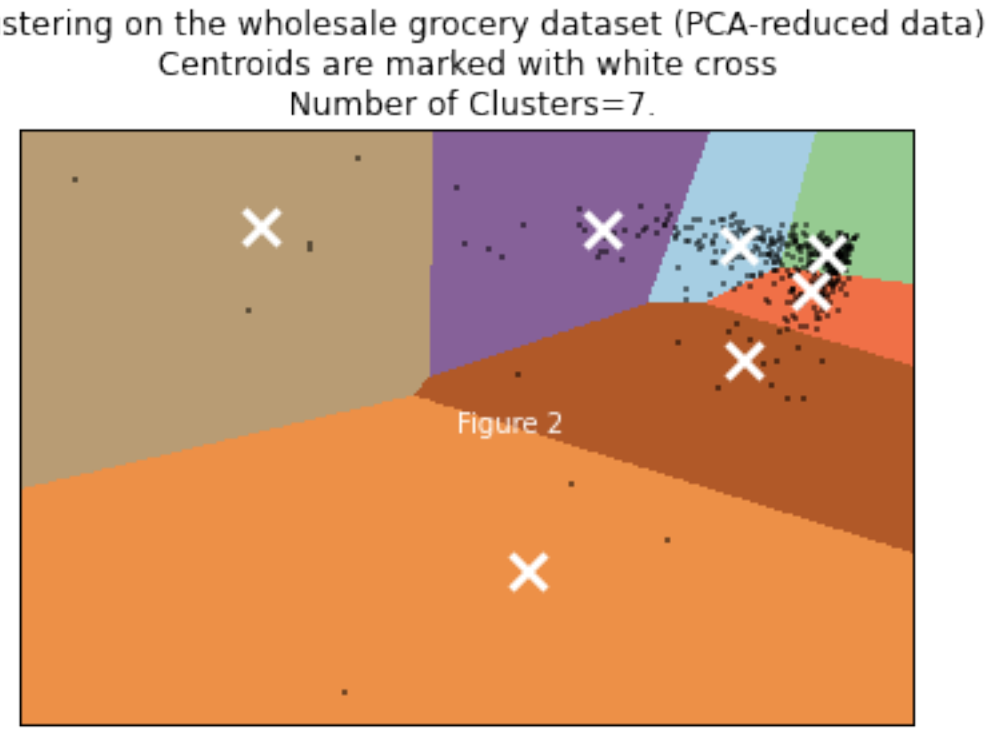
```
Produce_Stats(data, clusters, t_reduced_data)
```

```
KMeans model parameters:
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=2, n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
    verbose=0)
Cluster centers:
[[-3.67428358  0.40751388]
 [ 0.41858927 -0.04642563]]
```




```
Number of datapoints in each cluster:
0    395
1     45
dtype: int64

0 Fresh               12084.959494
Milk                   4152.612658
Grocery                5595.116456
Frozen                 3080.463291
Detergents_Paper       1712.782278
Delicatessen           1309.549367
total_spend           27935.483544
cluster                   0.000000
dtype: float64
1 Fresh               11257.155556
Milk                  20223.888889
```

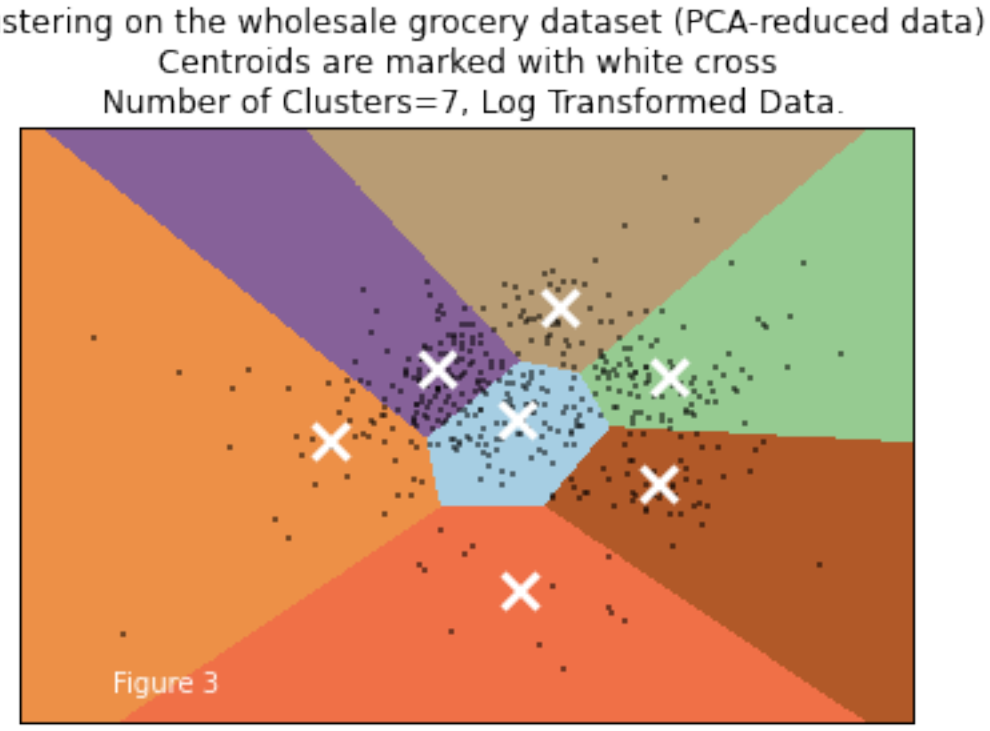
```
Grocery             28633.133333
Frozen               2997.044444
Detergents_Paper    13140.177778
Delicatessen         3414.911111
total_spend         79666.311111
cluster                 1.000000
dtype: float64

KMeans model parameters:
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=7, n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
    verbose=0)
Cluster centers:
[[ -0.71866834   0.5604511 ]
 [  0.88362244   0.31504725]
 [ -9.31832266   1.12533697]
 [  0.58959368  -0.96382991]
 [ -4.51308525 -10.04070335]
 [ -3.16138343   1.06396649]
 [ -0.59839041  -3.15947057]]
```




```
Number of datapoints in each cluster:
3    173
0    105
5     98
1     33
4     21
```

```
6      7
2      3
dtype: int64

0 Fresh               19819.438095
Milk                   3251.066667
Grocery                3517.409524
Frozen                 4868.133333
Detergents_Paper        575.476190
Delicatessen           1494.866667
total_spend           33526.390476
cluster                   0.000000
dtype: float64
1 Fresh                5132.151515
Milk                  15446.696970
Grocery               24137.333333
Frozen                 1602.969697
Detergents_Paper      11325.424242
Delicatessen           1767.909091
total_spend           59412.484848
cluster                   1.000000
dtype: float64
2 Fresh               60571.666667
Milk                  30120.333333
Grocery               17314.666667
Frozen                38049.333333
Detergents_Paper       2153.000000
Delicatessen          20700.666667
total_spend          168909.666667
cluster                   2.000000
dtype: float64
3 Fresh                6911.000000
Milk                   2351.369942
Grocery                3173.566474
Frozen                 1541.618497
Detergents_Paper        782.815029
Delicatessen            695.341040
total_spend           15455.710983
cluster                   3.000000
dtype: float64
4 Fresh               37192.380952
Milk                   6700.857143
Grocery                7634.333333
Frozen                11596.809524
Detergents_Paper       1145.380952
Delicatessen           4485.047619
total_spend           68754.809524
cluster                   4.000000
dtype: float64
5 Fresh                7460.734694
Milk                   8110.204082
Grocery               12025.785714
Frozen                 1482.295918
Detergents_Paper       4839.204082
```

```
Delicatessen         1645.122449
total_spend         35563.346939
cluster                 5.000000
dtype: float64
6 Fresh                20031.285714
Milk                   38084.000000
Grocery                56126.142857
Frozen                  2564.571429
Detergents_Paper       27644.571429
Delicatessen            2548.142857
total_spend           146998.714286
cluster                   6.000000
dtype: float64

KMeans model parameters:
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=7, n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
    verbose=0)
Cluster centers:
[[-0.20224645 -0.36805633]
 [ 1.89790468  0.46769048]
 [ 0.41179199  1.74155997]
 [-0.14478038 -3.620432  ]
 [-2.77512494 -0.75426506]
 [-1.29017565  0.5854917 ]
 [ 1.75604165 -1.55881109]]
```




```
Number of datapoints in each cluster:
```

```
1    112
0     90
3     85
4     52
6     49
5     39
2     13
dtype: int64

0 Fresh                8359.700000
Milk                   3607.077778
Grocery                4436.111111
Frozen                 1373.188889
Detergents_Paper       1096.155556
Delicatessen            953.944444
total_spend           19826.177778
cluster                   0.000000
dtype: float64
1 Fresh               14868.267857
Milk                   1918.848214
Grocery                2325.812500
Frozen                 4385.937500
Detergents_Paper        357.803571
Delicatessen           1034.785714
total_spend           24891.455357
cluster                   1.000000
dtype: float64
2 Fresh                 539.230769
Milk                   3647.461538
Grocery                8160.923077
Frozen                  297.230769
Detergents_Paper       1679.076923
Delicatessen            123.461538
total_spend           14447.384615
cluster                   2.000000
dtype: float64
3 Fresh               10895.211765
Milk                  12918.964706
Grocery               17715.094118
Frozen                 1960.705882
Detergents_Paper       7687.917647
Delicatessen           2260.200000
total_spend           53438.094118
cluster                   3.000000
dtype: float64
4 Fresh                2717.076923
Milk                   8430.096154
Grocery               15496.269231
Frozen                  567.807692
Detergents_Paper       7429.000000
Delicatessen            987.961538
total_spend           35628.211538
cluster                   4.000000
dtype: float64
```

```
5 Fresh                9872.948718
Milk                    812.205128
Grocery                1439.487179
Frozen                 2407.974359
Detergents_Paper        254.384615
Delicatessen            384.435897
total_spend           15171.435897
cluster                   5.000000
dtype: float64
6 Fresh                28634.204082
Milk                    8066.102041
Grocery                 7448.979592
Frozen                  9038.306122
Detergents_Paper        1175.489796
Delicatessen            4267.408163
total_spend            58630.489796
cluster                    6.000000
dtype: float64
```

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

Answer: From Rubric: PCA has been used to visualize the data in two dimensions. A plot has been created which clearly shows different clusters. If clusters are not clearly visible some discussion has been made of how to improve the visualization.

The central object in each cluster, is known as the centroid and is represented by a single vector, which is the arithmetic mean or average position of all the individual data points in the cluster.

If you represent each cluster as a group of customers, then the centroid of each cluster will represent the average customer within each group of customers. So each cluster of customers represents a customer type based upon the relative amounts of 'Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergent/Paper' and 'Delicatessen' they buy from the wholesale distributor.

Looking at the plot using n_clusters=2 (see figure 1 above), we see two distinct clusters, one has 395 datapoints with an average total spend of $27,935, the other has 45 datapoints with an average total spend of $79,666. So one cluster appears to represent a customer group of larger businesses, maybe large supermarkets or chains, while the other may represent smaller businesses such as local corner stores. Looking at the raw data suggests to me that just having two distinct clusters seems rather simplistic and probably not much use to the wholesale distributor in finding different customer types, ie. 'Big' and 'Small' doesn't really tell you much about them. To me each cluster is more like a super cluster in which more discrete clusters may be found, which in turn would provide more detailed information to the wholesale distributor.

Moving on to the next plot, where n_clusters=7 (see figure 2 above), here some of the clusters are tightly grouped in the top right hand corner, with a couple of centroids very close to each other. A way to better visualise the data is to apply a transformation with the aim of spreading out the data points, which should lead to better defined clusters. I did this by applying a logarithmic transformation to the raw data, then performing the scaling and data reduction processes. The resulting data was then plotted (see figure 3 above). Here we observe that the datapoints are more spread out leading to more well defined clusters and well spaced centroids. A tanh transformation on the scaled_data yielded a very similiar plot as figure 3 (not shown for clarity).

Looking at the number of datapoints in each cluster for figures 2 (173, 107, 102, 29, 21, 5 ,3) and figure 3 (107, 92, 85, 52, 49, 42, 13) we see quite a difference at either end, ie the largest and smallest clusters. In the case of the datapoints in the smaller clusters from figure 2, these apparent outliers have moved closer to the rest of the data and been classified accordingly. It would require a detailed look at the individual datapoints to determine which method (transformation or not) makes the most sense in clustering the data. Also with increasing values of n_clusters also increases the chances of overfitting the data, again domain knowledge would help alleviate that possibility.

1.2.2 Conclusions

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

Answer: From Rubric: One method has been discussed in detail and it's usefulness has been discussed.

The techniques I have used to investigate the wholesale grocery distributor customer data were PCA, ICA and K-Means.

As one of the aims of the project was to provide the wholesale grocery distributor with a sense of what sorts of different customers they have, I feel that K-Means in conjunction with the visualisation gives a good insight into the data by finding clusters of related data points that may describe distinct groups of customers.

However, it is clear that none of the used techniques can be used very meaningfully in isolation, they need to be used in conjunction with each other inorder to augment each other. PCA can be used to reduce complexity (dimensionality reduction) in the dataset by mapping the dataset into a new feature space using its principal components. K-Means is then applied to the data in the new feature space with the objective of being better able to distinguish the different clusters. (GMM can also benefit in the same way as K-Means by pre-processing the data with PCA).

One potential problem with techniques K-Means and GMM for this example is that you need to know the number of clusters in advance, and as the objective was to determine the different sorts of customers the distributor has, this suggests that neither technique is optimal for this investigation.

However, GMM provides a metric, the Bayesian Information Criterion (BIC), which can be used in an analysis to help determine the number of clusters (n_components) to specify when building the GMM model. The sklearn function silhouette_score, which gives a perspective into the density and separation of the formed clusters, can be used to find the optimal number of clusters to be specified for the K-Means n_clusters parameter. The drawback is that each requires you to run the model (K-Means, GMM) multiple times with different values for n_clusters/n_components and select a final value for n_clusters or n_components based upon the interpretation of the values output from silhouette_score or BIC.

As this project is about unsupervised learning, ideally you want an algorithm/technique that can find the number of clusters in the dataset without been given any hints or clues as to the number, as by definition, we don't know the number of clusters and are tasked with finding that number out. Other clustering techniques exist, such as DPGMM, that will attempt to find the optimal number of clusters, this model only needs to be given a loose upper bound for the number of clusters.

I used GMM's BIC function and found the optimal number of clusters to be 7 (see above). I also used the silhouette_score, but that was inconclusive, with 6-8 clusters found. DPGMM was also used, it found 6 distinct clusters. I did not include code/graphs for DPGMM and silhouette_score for the sake of clarity.

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

Answer: From Rubric: Some method of improving the ability to get good results from an A/B test has been proposed.

Once you have used K-Means to identify the number of clusters in the dataset ie. the number of different groups of customers, then for each customer (data point) identify which cluster they belong in. You could update the dataset with a "label" field containing the cluster they belong in, specifically the "customer type".

So now when the company does an A/B test, and surveys all their customers, for instance if they were to ask questions about changing their delivery times and frequency, they will be able to determine if any of the different customer types responded the same or differently. For instance, if most or all customers of a particular customer type responded negatively to a proposed delivery schedule change then the company can take some actions with that customer group to mitigate any problems this proposed change might introduce. Thus avoiding creating unhappy customers and potential loss of business.

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

Answer: Some techniques that could be used in a supervised learning analysis have been proposed.

Now that the input dataset can be labelled with a 'customer type', a supervised learning model such as SVM, Decision trees, or an ensemble model could be trained on existing data with a view to classifying brand new customers, based upon their spending, into the most likely customer type.

Having grouped customers into specific types will lead to more accurate predictions of what they buy and when, as you can make predictions of future spending using regression models within a customer type rather than just against the whole dataset. Being able to predict future sales would also make it possible to

predict when a customer would need a delivery, which opens up possibilities of the distributor making more customer friendly delivery schedules and making better use of their own transport fleet resources.

Not only being able to forecast/predict future sales to a customer also allows the distributor to have better control over stock, ie. never have too much when its not needed and have enough when it is needed. So the customer will always get stock when its required and the distributor doesn't have cash tied up in stock that doesn't sell.