P3_Creating_Customer_Segments

March 31, 2016

1 P3 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).

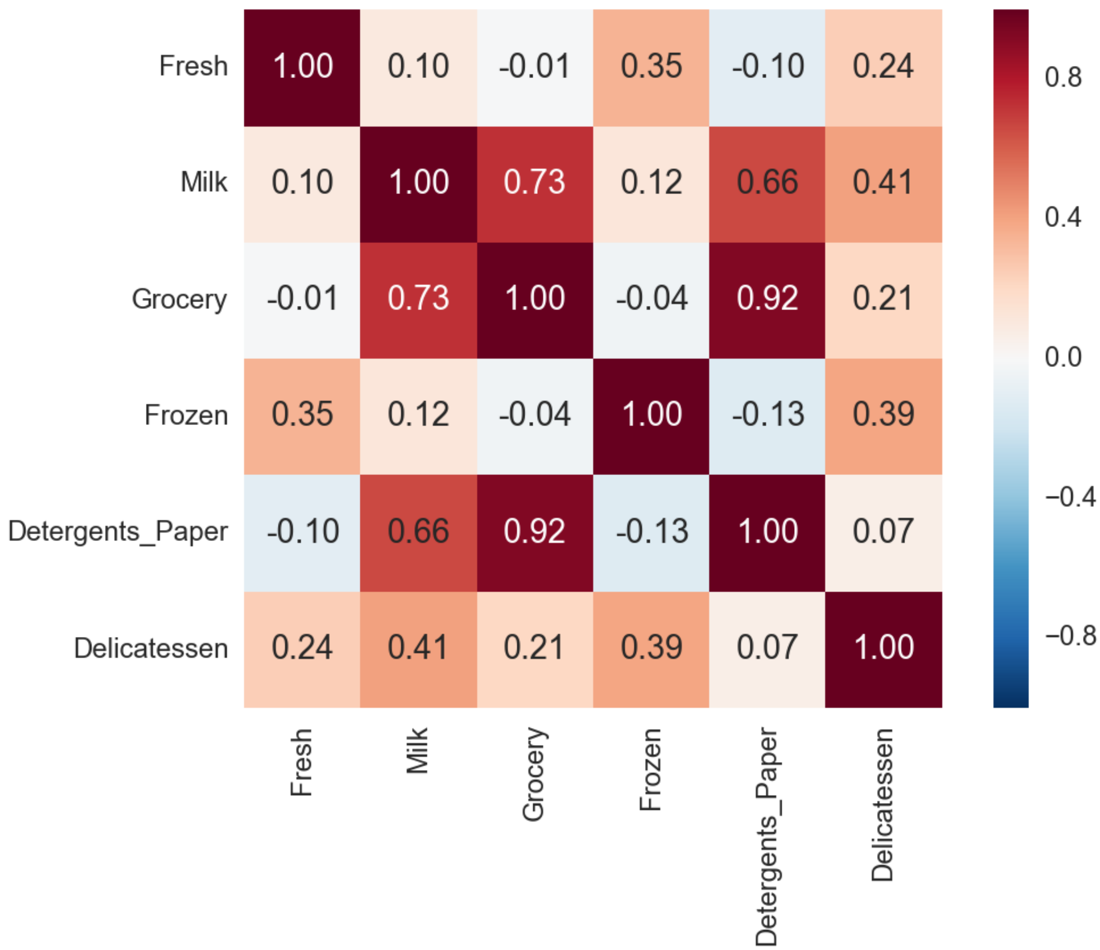
In [1]:
```
# Import libraries: NumPy, pandas, matplotlib
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from   matplotlib import rc

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

# Set styles for seaborn
%config InlineBackend.figure_formats = {'png', 'retina'}
rc_sns = {'lines.linewidth': 2,
          'axes.labelsize': 14,
          'axes.titlesize': 14,
          'axes.facecolor': 'DFDFE5'}
sns.set_context('notebook', font_scale=1.2, rc=rc_sns, )
sns.set_style  ('darkgrid', rc=rc_sns)

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

```
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

Explore the dataset with correlation matrix.

```
In [2]: from sklearn.cross_validation import train_test_split
        from sklearn.preprocessing import StandardScaler, MinMaxScaler

        sc    = StandardScaler()
        X_std = sc.fit_transform(data)

        cols = list(data.columns)
        sns.heatmap(np.corrcoef(X_std.T),
                    cbar=True, square=True, annot=True, fmt='.2f',
                    xticklabels=cols, yticklabels=cols)

Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x10e1ba190>
```



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.

Answer:

PCA seeks to find orthogonal directions (principle components) that maximize variances. If two features have high absolute correlation, a linear combination of those features will likely have large weights in the principle components. For this dataset, it's reasonable to assume "Fresh" and "Grocery" would have a high correlation, thus relatively large weights in the first PCA dimension.

For ICA, non-normality of the marginal densities are maximized to discover latent variables (independent components) underlying the dataset.[1] Thus the vectors showing up after ICA would be the "source" that can generate the observed dataset.

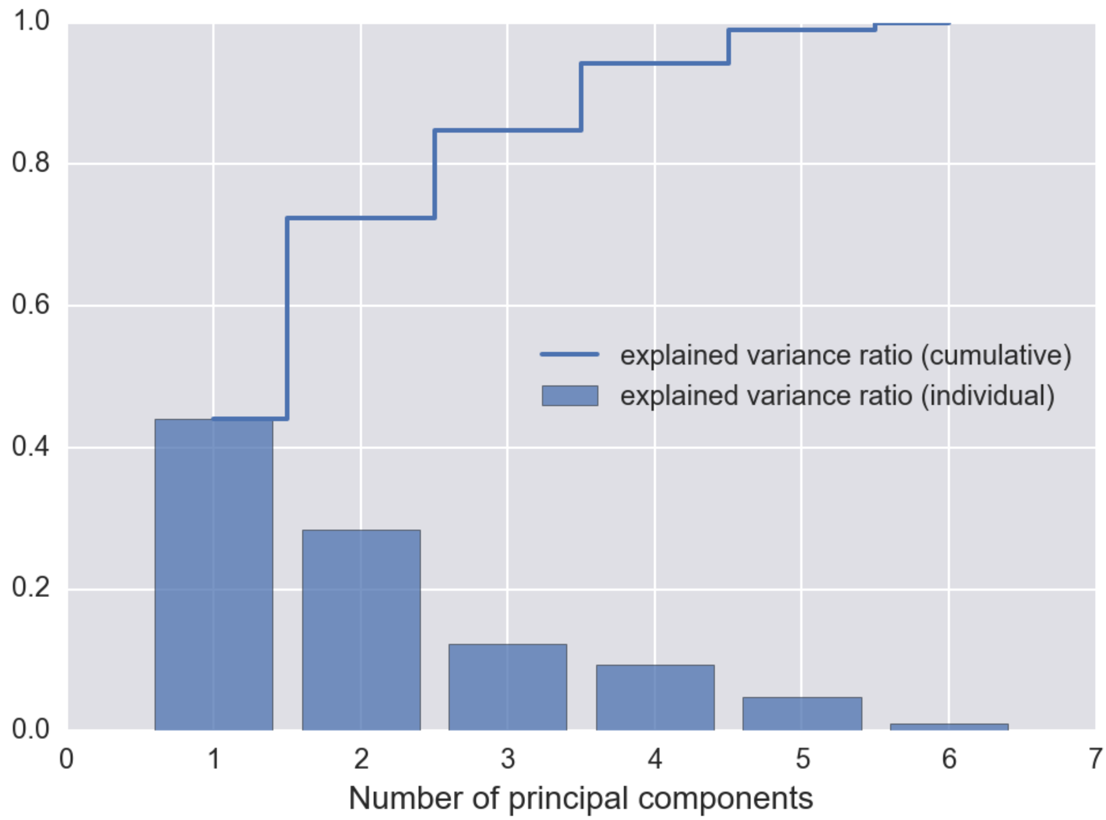
[1] http://research.ics.aalto.fi/ica/icademo/

1.2.1 PCA

In [3]:
```
# 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 input data
sc    = StandardScaler()
X     = data.values.astype(np.float64)
X_std = sc.fit_transform(X)

# Perform PCA analysis
pca       = PCA(n_components=None)
X_std_pca = pca.fit_transform(X_std)

# Print the components and the amount of variance in the data contained in each dimension
print('Principal components (pc) :')
print(pca.components_)

print('\nExplained variance ratios (EVR):')
print(pca.explained_variance_ratio_)

print('\nEVR of the 1st and 2nd pc: %.3f' % sum(pca.explained_variance_ratio_[:2]))
print('EVR of the 3rd and 4th pc: %.3f' % sum(pca.explained_variance_ratio_[2:4]))
print('EVR of the first four  pc: %.3f' % sum(pca.explained_variance_ratio_[:4]))

# visualization of indivudual and cumulative explained variance ratio
# code adapted from p132 of S. Raschka "Python Machine Learning" 2015
n_pca   = pca.n_components_
evr     = pca.explained_variance_ratio_
cum_evr = np.cumsum(evr)

plt.bar (range(1, n_pca+1), evr, alpha=0.75,
         align='center', label='explained variance ratio (individual)')

plt.step(range(1, n_pca+1), cum_evr,
         where='mid',    label='explained variance ratio (cumulative)')

plt.legend(loc='best')
plt.xlabel('Number of principal components')
rc('font', weight='bold')
```

```
Principal components (pc) :
[[-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]]

Explained variance ratios (EVR):
[ 0.44082893  0.283764    0.12334413  0.09395504  0.04761272  0.01049519]

EVR of the 1st and 2nd pc: 0.725
EVR of the 3rd and 4th pc: 0.217
EVR of the first four  pc: 0.942
```



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:

The explained variance ratio drops off pretty fast after the first two principle components, which account for **72.5%** of the total variance while the third and fourth constitute around **21.7%**.

It would be reasonable to choose a total of **four dimensions** for analysis since the first four principle components accounts for around **94.2%** of the variance.

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

```
In [4]: pd.DataFrame(pca.components_, columns=cols, index=['PC %d' %idx for idx in range(1, len(cols)+1
```

Out[4]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
PC 1	-0.042884	-0.545118	-0.579256	-0.051189	-0.548640	-0.248682
PC 2	-0.527932	-0.083168	0.146088	-0.611278	0.255233	-0.504207

```
PC 3 -0.812257  0.060388 -0.108384  0.178386         -0.136192      0.523904
PC 4 -0.236686 -0.087190  0.105987  0.768683          0.171744     -0.552065
PC 5  0.048683 -0.826579  0.314999  0.027932          0.339640      0.314701
PC 6  0.036025  0.038040 -0.721745  0.015637          0.685894      0.075134
```

Answer:

The dimensions represent linear combinations of the original features, which capture the variances between data points in decreasing order. For example, as shown in the table above, the first principle component (PC) will take the original dataset and transform it to lie along an axis that assigns more absolute weight to features such as **Milk** and **Grocery** compared to **Fresh** and **Frozen**. For the second PC, more absolute weight is assigned to **Fresh**, **Frozen** and **Delicatessen** compared to **Grocery** and **Milk**.

The transformed dataset might prove to be a better representation of the original dataset, thus potentially making clustering results more relevant. Throwing out the principal components with low variance might also help reduce the "curse of dimensionality".

1.2.2 ICA

```
In [10]: # TODO: Fit an ICA model to the data
         # Note: Adjust the data to have center at the origin first!
         from sklearn.decomposition import FastICA

         ica   = FastICA(n_components=None, whiten=True, random_state=0)
         X_ica = ica.fit_transform(X_std)


         print('The unmixing matrix:')
         pd.set_option('display.precision', 4)
         pd.DataFrame(ica.components_, columns=cols, index=['IC %d' %idx for idx in range(1, len(cols)+
```

```
The unmixing matrix:
```

```
Out[10]:         Fresh    Milk  Grocery  Frozen  Detergents_Paper  Delicatessen
         IC 1   0.0026 -0.0130   0.0642  0.0018           -0.0079       -0.0047
         IC 2   0.0037 -0.0168  -0.1130  0.0071            0.1342        0.0159
         IC 3  -0.0019 -0.0728   0.0544  0.0018           -0.0146        0.0172
         IC 4  -0.0502  0.0064   0.0065  0.0033           -0.0104        0.0029
         IC 5  -0.0049 -0.0016  -0.0055 -0.0024            0.0023        0.0509
         IC 6   0.0109  0.0010  -0.0073 -0.0541            0.0026        0.0169
```

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:

Each row vector in **ica.components_** represents a row in the un-mixing matrix. After transforming the original standardized dataset with **X_std.dot(ica.components_.T)**, each new dimension would be as statistically independent as possible. * the first independent component (IC) could be used to transform the original data to a dimension that assigns more weight to **Grocery** compared to **Fresh**, according to the table shown above. * IC2 assigns more weight to **Detergents_Paper** and negative weight to **Grocery** * IC3 assigns more weight to **Grocery** and negative weight to **Milk** * IC4 assigns negative weight to **Fresh** and more or less zero weight to other categories.

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?

Answer:

KMeans * advantage: reasonably fast performance. * disadvantages: 1) assumes spherical cluster; 2) can't handle unevenly sized clusters well; 3) need to specify number of clusters. [2]

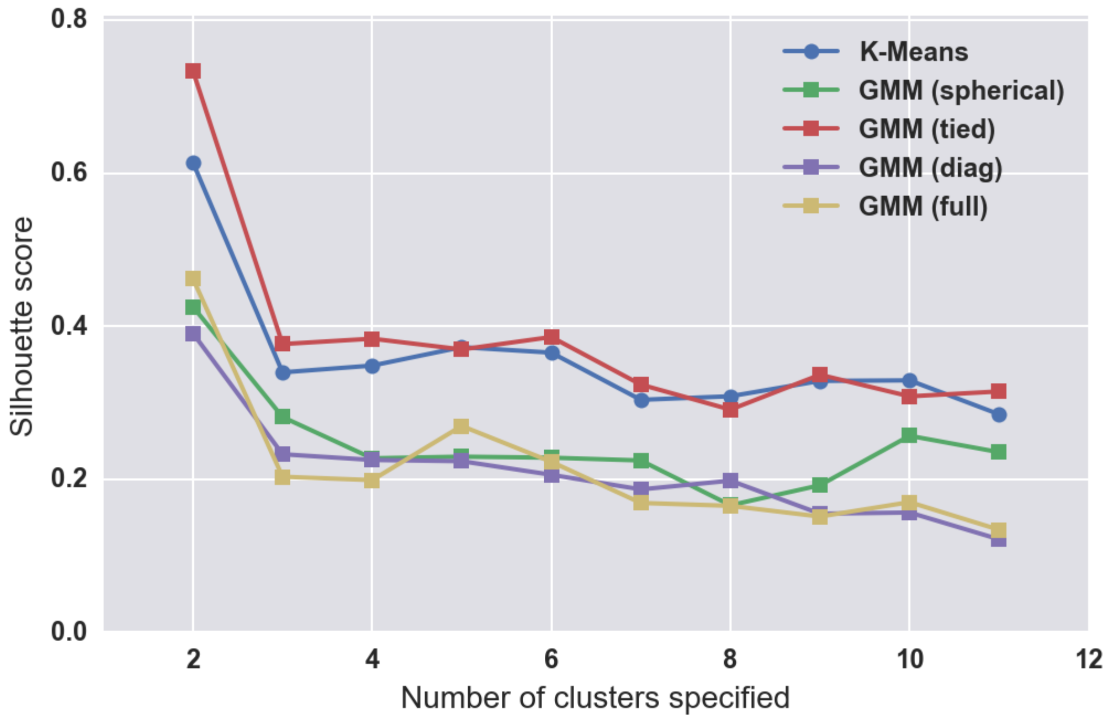
Gaussian Mixture Model * advantage: perform fuzzy membership assignment and accommodate clusters with vastly different sizes as well as correlation structures. * disadvantages: 1) need to specify number of Gaussian mixtures; 2) sometimes can be stuck in local minimums.[3]

For this case, GMM might be more appropriate since the cluster size might not be evenly distributed.

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 [6]: # Generate a plot for silhouette scores vs number of specified clusters
        # in order to aid selection of optimal cluster size

        # Import clustering modules
        import itertools
        import matplotlib.gridspec as gridspec
        from mlxtend.evaluate import plot_decision_regions
        from sklearn.cluster  import KMeans
        from sklearn.mixture  import GMM
        from sklearn.metrics  import silhouette_score

        # Use silhouette score to select the number of clusters for K-means and GMM
        n_cluster_min = 2
        n_cluster_max = 11
        cluster_range = range(n_cluster_min, n_cluster_max+1)

        # Compute silhouette scores
        km_silhouette  = []
        cov_types      = ['spherical', 'tied', 'diag', 'full']
        empty_list     = [[] for i in cov_types]
        gmm_silhouette = dict(zip(cov_types, empty_list))

        for i in cluster_range:
            y_km  = KMeans(n_clusters=i, random_state=0).fit_predict(X_std)
            km_silhouette.append (silhouette_score(X_std, y_km,  metric='euclidean'))

            for cov_type in cov_types:
                y_gmm = GMM(n_components=i, random_state=0, covariance_type= cov_type).fit_predict(X_st
                gmm_silhouette[cov_type].append(silhouette_score(X_std, y_gmm, metric='euclidean'))

        # Plot silhouette score vs number of clusters chosen
        plt.figure(figsize=(8, 5), dpi=300)

        plt.plot(cluster_range, km_silhouette,  marker='o', label = 'K-Means')
        for cov_type in cov_types:
            plt.plot(cluster_range, gmm_silhouette[cov_type], marker='s', label = 'GMM (%s)' %cov_type 

        plt.xlabel('Number of clusters specified')
        plt.ylabel('Silhouette score')
        plt.xlim(n_cluster_min-1, n_cluster_max+1)
        plt.ylim(0, 0.805)
        plt.yticks(np.arange(0, 0.805, 0.2))
```

```
plt.legend(loc='upper right')
rc('font', weight='bold')
```

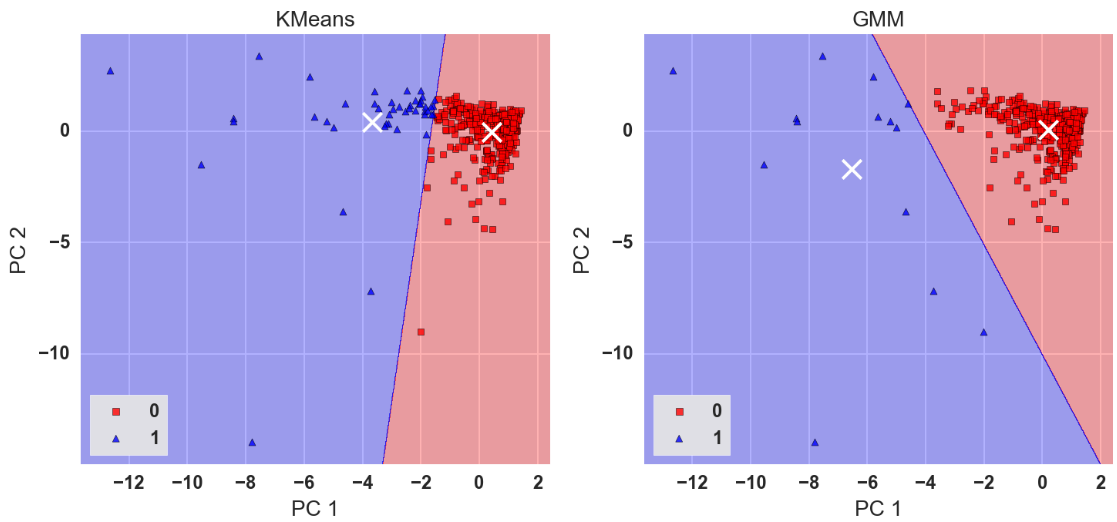


From the plot shown above, GMM clustering algorithm with a covariance type of **tied** and **n_components=2** provides the best silhouette-score of **0.733**, which is around **19.3%** better compared to K-means(n_cluster=2).

In [7]:
```
# TODO: First we reduce the data to two dimensions using PCA to capture variation
pca_reduced       = PCA(n_components=2)
X_std_pca_reduced = pca_reduced.fit_transform(X_std)
print('==============================================================================')
print('First 5 elements after transformation with the first two principle components:')
print(X_std_pca_reduced[:5])
print('==============================================================================')

# TODO: Implement your clustering algorithm here,
# and fit it to the reduced data for visualization

# initializing clustering algorithms KMeans/GMM and centroids
clus      = [KMeans(n_clusters=2), GMM(n_components=2, covariance_type='tied')]
centroids = {}

# plotting decision regions for KMeans and GMM
gs   = gridspec.GridSpec(1, 2)
grds = itertools.product([0, 1], repeat=1)

fig = plt.figure(figsize=(12, 5))
for clu, grd in zip(clus, grds):
```

```
    # fit to the reduced reduced data
    y_clu = clu.fit_predict(X_std_pca_reduced)

    clu_name = clu.__class__.__name__
    if   clu_name=='KMeans': centroids[clu_name] = clu.cluster_centers_
    elif clu_name=='GMM':    centroids[clu_name] = clu.means_

    print('%s:\n%s' % (clu_name, clu))
    print('==============================================================================')

    # plotting the decision boundaries with mlxtend.evaluate.plot_decision_regions
    # http://rasbt.github.io/mlxtend/user_guide/evaluate/plot_decision_regions/
    ax  = plt.subplot(gs[grd[0]])
    fig = plot_decision_regions(X=X_std_pca_reduced, y=y_clu, clf=clu)
    fig.legend(loc='lower left', frameon=True)

    # plotting the centroids of the clusters
    plt.scatter(centroids[clu_name][:, 0], centroids[clu_name][:, 1],
                marker='x', s=150, linewidths=2, color='w', zorder=2)
    plt.title (clu_name)
    plt.xlabel('PC 1')
    plt.ylabel('PC 2')
```

```
==============================================================================
First 5 elements after transformation with the first two principle components:
[[-0.19329055  0.30509996]
 [-0.4344199   0.32841262]
 [-0.81114323 -0.8150957 ]
 [ 0.77864783 -0.65275373]
 [-0.16628726 -1.27143372]]
==============================================================================
KMeans:
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)
==============================================================================
GMM:
GMM(covariance_type='tied', init_params='wmc', min_covar=0.001,
  n_components=2, n_init=1, n_iter=100, params='wmc', random_state=None,
  thresh=None, tol=0.001, verbose=0)
==============================================================================
```



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

```
In [8]: print('Centroids of KMeans:')
        print(centroids['KMeans'])

        print('\nMeans of GMM:')
        print(centroids['GMM'])

        pd.set_option('precision', 4)
        print('Customer types based on KMeans centroids:')
        pd.DataFrame(pca_reduced.inverse_transform(centroids['KMeans']), columns=cols, index=range(1,3)
```

```
Centroids of KMeans:
[[ 0.41858927 -0.04642563]
 [-3.67428358  0.40751388]]

Means of GMM:
[[ 0.21094239  0.05500227]
 [-6.53677134 -1.70443345]]
Customer types based on KMeans centroids:
```

Out[8]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
1	0.0066	-0.2243	-0.2493	0.007	-0.2415	-0.0807
2	-0.0576	1.9690	2.1879	-0.061	2.1199	0.7083

```
In [9]: print('Customer types based on GMM means:')
        pd.DataFrame(pca_reduced.inverse_transform(centroids['GMM']), columns=cols, index=range(1,3))
```

```
Customer types based on GMM means:
```

Out[9]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
1	-0.0381	-0.1196	-0.1142	-0.0444	-0.1017	-0.0802
2	1.1801	3.7051	3.5375	1.3765	3.1513	2.4850

Answer:

The centroids returned by both KMeans and GMM are listed above. For KMeans, the data points just to the left of the upper decision boundaries might not be suitably classified. Visually, the decision boundaries produced by GMM seems to be more plausible.

According to **KMeans**, 1. customer type 1 is **not** likely to buy Milk, Grocery, Detergents_Paper and somewhat agnostic to categories such as Fresh, Frozen and Delicatessen; 2. customer type 2 is quite likely to buy Milk, Grocery, Detergents_Paper and Delicatessen but agnostic to buy fresh and frozen.

According to **GMM**, 1. customer type 1 is more likely to spend more on all categories - high volume buyers; 2. customer type 2 is less likely to buy products from any category, especially Milk and Grocery - low volume buyers.

1.3.2 Conclusions

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

Answer:

It seems like GMM(n_components=2, covariance_type='tied') algorithm provides more insight into the data since it separates the customer base into high and low volume buyers, who might have drastically different behaviors and needs.

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

Answer:

With GMM clustering model, we can separate the existing customer base into two types and perform targeted experiments. For each type of customers, we can first randomly divide them into two groups, one of which will receive existing service and serve as the control while the other will be the experimental group, receiving a variation of the service. Responses/sales from each group of customers can then be recorded and analyzed, which can help the company to decide whether to change the service for a particular type of customers.

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

Answer:

After performing some experiments using the methodology outlined above, the responses/sales can be organized into a binary target variable (better/worse) or a continuous target variable (percent difference compared to the average of the matching control group).

For each case, we can use existing data to train and validate a supervised classification/regression model with, for example, KNN or Logistic Regression. When a new customer's data becomes available, we can use the model to predict whether the customer will respond favorably to the change in service or predict the magnitude of response in terms of sales.