Week 4. Running a k-means Cluster Analysis

July 9, 2016

In [1]:
```
#
# Created on Mon Jan 18 19:51:29 2016
#
# @author: jrose01
# Adapted by: mcolosso
#
```

In [2]:
```
%matplotlib inline

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing
from sklearn.cluster import KMeans

#pd.set_option('display.float_format', lambda x:'%.3f'%x)
#pd.set_option('display.mpl_style', 'default')   # --deprecated
#plt.style.use('ggplot')   # Make the graphs a bit prettier
plt.rcParams['figure.figsize'] = (15, 5)
```

In [3]:
```
#
# DATA MANAGEMENT
#
```

In [4]:
```
#Load the dataset

loans = pd.read_csv("./LendingClub.csv", low_memory = False)

# LendingClub.csv is a dataset taken from The LendingClub (https://www.lendingclub.com/)
# which is a peer-to-peer leading company that directly connects borrowers and potential
# lenders/investors
```

In [5]:
```
#
# Exploring the target column
#
```

In [6]:
```
# The target column (label column) of the dataset that we are interested in is called
# 'bad_loans'. In this column 1 means a risky (bad) loan and 0 means a safe loan.
#
# In order to make this more intuitive, we reassign the target to be:
# 1 as a safe loan and 0 as a risky (bad) loan.
#
```

```
# We put this in a new column called `safe_loans`.

loans['safe_loans'] = loans['bad_loans'].apply(lambda x : 1 if x == 0 else 0)
loans.drop('bad_loans', axis = 1, inplace = True)
```

In [7]:
```
# Select features to handle

# In this oportunity, we are going to ignore 'grade' and 'sub_grade' predictors
# assuming those are a way to "clustering" the loans

predictors = ['short_emp',                     # one year or less of employment
              'emp_length_num',                # number of years of employment
              'home_ownership',                # home_ownership status: own, mortgage or rent
              'dti',                           # debt to income ratio
              'purpose',                       # the purpose of the loan
              'term',                          # the term of the loan
              'last_delinq_none',              # has borrower had a delinquincy
              'last_major_derog_none',         # has borrower had 90 day or worse rating
              'revol_util',                    # percent of available credit being used
              'total_rec_late_fee',            # total late fees received to day
             ]

target     =  'safe_loans'                     # prediction target (y) (+1 means safe, 0 is risky)

ignored    = ['grade',                         # grade of the loan
              'sub_grade',                     # sub-grade of the loan
             ]

# Extract the predictors and target columns
loans = loans[predictors + [target]]

# Delete rows where any or all of the data are missing
loans = loans.dropna()
```

In [8]:
```
# Convert categorical text variables into numerical ones

categorical = ['home_ownership', 'purpose', 'term']
for attr in categorical:
    attributes_list = list(set(loans[attr]))
#    print('{}:'.format(attr), list(enumerate(attributes_list)))
    loans[attr] = [attributes_list.index(idx) for idx in loans[attr] ]
```

In [9]:
```
(loans.describe()).T
```

Out[9]:

	count	mean	std	min	25%	50%	\
short_emp	122607.0	0.123672	0.329208	0.0	0.00	0.00	
emp_length_num	122607.0	6.370256	3.736014	0.0	3.00	6.00	
home_ownership	122607.0	1.948184	0.959482	0.0	1.00	2.00	
dti	122607.0	15.496888	7.497442	0.0	9.88	15.26	
purpose	122607.0	3.556836	3.272598	0.0	1.00	1.00	
term	122607.0	0.202321	0.401732	0.0	0.00	0.00	
last_delinq_none	122607.0	0.588115	0.492177	0.0	0.00	1.00	
last_major_derog_none	122607.0	0.873906	0.331957	0.0	1.00	1.00	
revol_util	122607.0	53.716307	25.723881	0.0	34.80	55.70	
total_rec_late_fee	122607.0	0.742344	5.363268	0.0	0.00	0.00	

```
safe_loans             122607.0   0.811185   0.391363  0.0   1.00   1.00

                          75%     max
short_emp                 0.00    1.00
emp_length_num           11.00   11.00
home_ownership            3.00    3.00
dti                      20.85   39.88
purpose                   6.00   11.00
term                      0.00    1.00
last_delinq_none          1.00    1.00
last_major_derog_none     1.00    1.00
revol_util               74.30  150.70
total_rec_late_fee        0.00  208.82
safe_loans                1.00    1.00
```

In [10]:
```
#
# MODELING AND PREDICTION
#
```

In [11]:
```
# Standardize clustering variables to have mean=0 and sd=1

for attr in predictors:
    loans[attr] = preprocessing.scale(loans[attr].astype('float64'))
```

In [12]:
```
# Split data into train and test sets

clus_train, clus_test = train_test_split(loans[predictors], test_size = .3, random_state = 123

print('clus_train.shape', clus_train.shape)
print('clus_test.shape',  clus_test.shape )
```

```
clus_train.shape (85824, 10)
clus_test.shape (36783, 10)
```

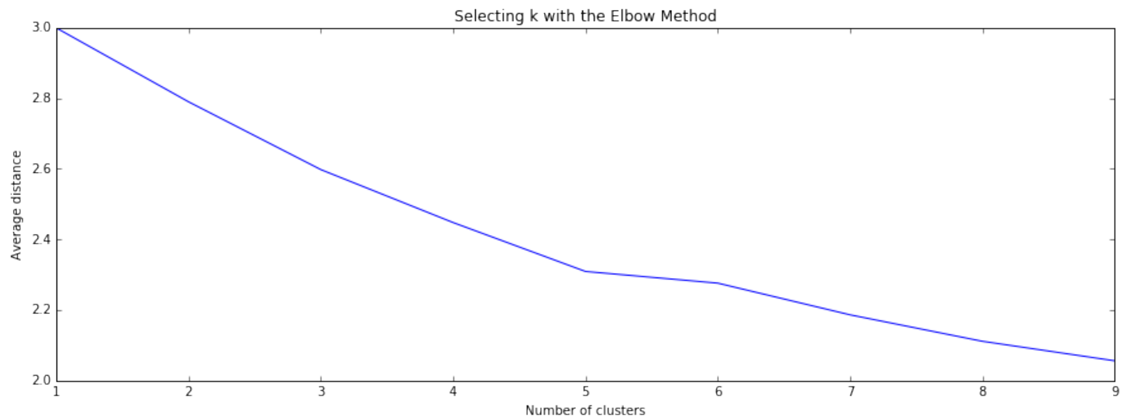
In [13]:
```
# K-means cluster analysis for 1-9 clusters

from scipy.spatial.distance import cdist
clusters = range(1,10)
meandist = list()

for k in clusters:
    model = KMeans(n_clusters = k).fit(clus_train)
    clusassign = model.predict(clus_train)
    meandist.append(sum(np.min(cdist(clus_train, model.cluster_centers_, 'euclidean'), axis=1)
                    / clus_train.shape[0])
```

In [14]:
```
"""
Plot average distance from observations from the cluster centroid
to use the Elbow Method to identify number of clusters to choose
"""

plt.plot(clusters, meandist)
plt.xlabel('Number of clusters')
plt.ylabel('Average distance')
plt.title('Selecting k with the Elbow Method')
plt.show()
```

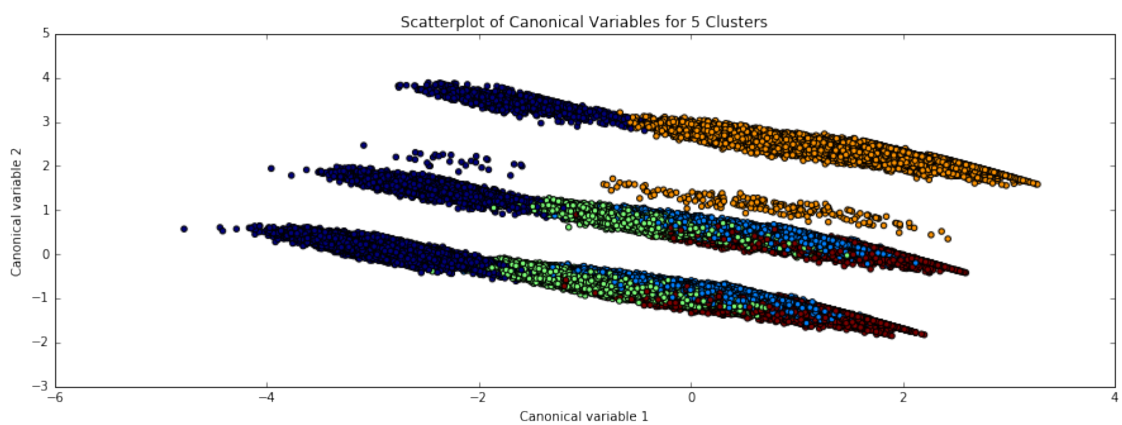



```
In [15]: # Interpret 5 cluster solution
         model = KMeans(n_clusters = 5)
         model.fit(clus_train)
         clusassign = model.predict(clus_train)

         # Plot clusters
         from sklearn.decomposition import PCA

         pca_2 = PCA(2)
         plot_columns = pca_2.fit_transform(clus_train)

         plt.scatter(x = plot_columns[:, 0], y = plot_columns[:, 1], c = model.labels_, )
         plt.xlabel('Canonical variable 1')
         plt.ylabel('Canonical variable 2')
         plt.title('Scatterplot of Canonical Variables for 5 Clusters')
         plt.show()
```




```
In [16]: #
         # BEGIN multiple steps to merge cluster assignment with clustering variables to examine
         # cluster variable means by cluster
         #
```

```
In [17]: # Create a unique identifier variable from the index for the
         # cluster training data to merge with the cluster assignment variable
         clus_train.reset_index(level = 0, inplace = True)

         # Create a list that has the new index variable
         cluslist = list(clus_train['index'])

         # Create a list of cluster assignments
         labels = list(model.labels_)

In [18]: # Combine index variable list with cluster assignment list into a dictionary
         newlist = dict(zip(cluslist, labels))
         #newlist

In [19]: # Convert newlist dictionary to a dataframe
         newclus = DataFrame.from_dict(newlist, orient = 'index')
         #newclus

In [20]: # Rename the cluster assignment column
         newclus.columns = ['cluster']

In [21]: # Now do the same for the cluster assignment variable:

         # Create a unique identifier variable from the index for the cluster assignment
         # dataframe to merge with cluster training data
         newclus.reset_index(level = 0, inplace = True)

         # Merge the cluster assignment dataframe with the cluster training variable dataframe
         # by the index variable
         merged_train = pd.merge(clus_train, newclus, on = 'index')
         merged_train.head(n = 100)
         # cluster frequencies
         merged_train.cluster.value_counts()

Out[21]: 1    26830
         2    25138
         4    13664
         0    10525
         3     9667
         Name: cluster, dtype: int64

In [22]: #
         # END multiple steps to merge cluster assignment with clustering variables to examine
         # cluster variable means by cluster
         #

In [23]: # FINALLY calculate clustering variable means by cluster
         clustergrp = merged_train.groupby('cluster').mean()
         print ("Clustering variable means by cluster")
         print(clustergrp)

Clustering variable means by cluster
                index  short_emp  emp_length_num  home_ownership       dti  \
cluster
0        58699.303753   2.661942       -1.509839        0.245320 -0.074409
1        59998.504473  -0.375666        0.400402       -0.909061 -0.029619
```

```
2        57927.052789  -0.375666       -0.111563        1.017338 -0.032616
3        78955.203476  -0.375666        0.364292       -0.080978  0.051115
4        59858.841847  -0.375666        0.337008       -0.241717  0.153958

          purpose      term  last_delinq_none  last_major_derog_none  \
cluster
0        0.062952 -0.132309          0.078780               0.067874
1        0.057060 -0.503624          0.113281               0.379852
2        0.002386 -0.503624          0.201739               0.379852
3       -0.115214  0.040211         -1.154998              -2.632602
4       -0.104852  1.985607          0.163269               0.379852

         revol_util  total_rec_late_fee
cluster
0         -0.071223            0.044134
1         -0.071310           -0.020887
2          0.018185            0.032263
3          0.043983           -0.082552
4          0.138153            0.008273
```

In [24]:
```
# Validate clusters in training data by examining cluster differences in SAFE_LOANS using ANOV
# first have to merge SAFE_LOANS with clustering variables and cluster assignment data
gpa_data = loans['safe_loans']

# split safe_loans data into train and test sets
gpa_train, gpa_test = train_test_split(gpa_data, test_size=.3, random_state=123)
gpa_train1 = pd.DataFrame(gpa_train)
gpa_train1.reset_index(level = 0, inplace = True)
merged_train_all = pd.merge(gpa_train1, merged_train, on = 'index')
sub1 = merged_train_all[['safe_loans', 'cluster']].dropna()
```

In [25]:
```
import statsmodels.formula.api as smf
import statsmodels.stats.multicomp as multi

gpamod = smf.ols(formula = 'safe_loans ~ C(cluster)', data = sub1).fit()
print (gpamod.summary())
```

```
OLS Regression Results
==============================================================================
Dep. Variable:             safe_loans   R-squared:                       0.022
Model:                            OLS   Adj. R-squared:                  0.022
Method:                 Least Squares   F-statistic:                     485.8
Date:                Sat, 09 Jul 2016   Prob (F-statistic):               0.00
Time:                        21:54:49   Log-Likelihood:                -40433.
No. Observations:               85824   AIC:                         8.088e+04
Df Residuals:                   85819   BIC:                         8.092e+04
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
Intercept           0.7885      0.004    208.708      0.000         0.781     0.796
C(cluster)[T.1]     0.0805      0.004     18.063      0.000         0.072     0.089
C(cluster)[T.2]     0.0301      0.004      6.697      0.000         0.021     0.039
C(cluster)[T.3]     0.0297      0.005      5.447      0.000         0.019     0.040
```

```
C(cluster)[T.4]    -0.0968      0.005    -19.249      0.000        -0.107    -0.087
==============================================================================
Omnibus:                    19021.697   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            34709.167
Skew:                          -1.536   Prob(JB):                         0.00
Kurtosis:                       3.513   Cond. No.                         7.48
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [26]: print ('Means for SAFE_LOANS by cluster')
         m1 = sub1.groupby('cluster').mean()
         print (m1)
```

```
Means for SAFE_LOANS by cluster
         safe_loans
cluster
0          0.788504
1          0.869027
2          0.818641
3          0.818248
4          0.691745
```

```
In [27]: print ('Standard deviations for SAFE_LOANS by cluster')
         m2 = sub1.groupby('cluster').std()
         print (m2)
```

```
Standard deviations for SAFE_LOANS by cluster
         safe_loans
cluster
0          0.408389
1          0.337377
2          0.385323
3          0.385660
4          0.461790
```

```
In [28]: mc1 = multi.MultiComparison(sub1['safe_loans'], sub1['cluster'])
         res1 = mc1.tukeyhsd()
         print(res1.summary())
```

```
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=============================================
group1 group2 meandiff  lower   upper  reject
---------------------------------------------
  0      1     0.0805   0.0684  0.0927  True
  0      2     0.0301   0.0179  0.0424  True
  0      3     0.0297   0.0148  0.0446  True
  0      4    -0.0968  -0.1105  -0.083  True
  1      2    -0.0504  -0.0597 -0.0411  True
  1      3    -0.0508  -0.0633 -0.0382  True
  1      4    -0.1773  -0.1884 -0.1662  True
  2      3    -0.0004   -0.013  0.0123 False
  2      4    -0.1269  -0.1381 -0.1157  True
  3      4    -0.1265  -0.1406 -0.1125  True
---------------------------------------------
```

```
In [ ]:
```