Fuzzy K_means & K Median

Fuzzy K Means is a soft clustering technique where data points can potentially belong to multiple clusters. Membership grades are assigned to each of the data points. These membership grades indicate the degree to which data points belong to each cluster. Thus, points on the edge of a cluster, with lower membership grades, may be in the cluster to a lesser degree than points in the center of cluster

K Medians is a variation of K-means where the centroid is calculated using median instead of mean. This technique minimizes error over all clusters with respect to 1-norm distance metric as opposed to the square of the 2-norm distance as in K-means

In our use case both fuzzy Kmeans and Kmedians struggle to form two clusters unlike Kmeans

In [1]:
```
from sklearn import datasets
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

from sklearn.cluster import MiniBatchKMeans
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn import svm, neighbors
from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score

import pandas as pd
import numpy as np
import itertools
import matplotlib.pyplot as plt
import datetime

%matplotlib inline
```

In [2]:
```
train_split = 0.80
nrows = 250_000
path = 'c:/users/ugy1/abs/'
df=pd.read_csv(path+'datasets/processed_abs_loan_'+str(nrows)+'.csv',
               #usecols=use_list,
               #sep='\t',
               #compression=bz2,
               nrows=nrows,
               low_memory=False,
              index_col=0,
               parse_dates=True
              )
df.shape
```

Out[2]: (237024, 58)

In [3]:
```
column_list=df.columns.tolist()
```

In [4]:
```
df.head()
```

Out[4]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnuml
0	66711.84	60	3.29	1
1	16258.45	60	0.90	0
2	31930.41	72	2.90	1
3	26065.02	65	0.90	0
4	42091.00	72	3.90	0

5 rows × 58 columns

In [5]:
```
# prepare label for scikit-learn
Y=df.label.values
Y.shape
```

Out[5]: (237024,)

In [6]:
```
# prepare input data for scikit-learn
input=df.values
input.shape
```

Out[6]: (237024, 58)

In [7]:
```
# calculate train/test split

len_train = int(len(input)*train_split)
print(len_train)
```

```
189619
```

In [8]:
```
# apply train/test split to labels
y_train = Y[0:len_train]
y_test = Y[len_train:]
x_train = input[0:len_train]
x_test = input[len_train:]
x_train.shape
```

Out[8]:
```
(189619, 58)
```

In [9]:
```
export_x_test = pd.DataFrame(data=x_test)
```

In [10]:
```
export_x_test.columns=column_list
export_x_test.rename(columns={'label':'True Label'}, inplace=True)
export_x_test.head()
```

Out[10]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumb
0	36863.24	72.0	1.00	1.0
1	23811.32	60.0	1.90	0.0
2	30669.00	48.0	1.00	1.0
3	54083.21	72.0	1.00	0.0
4	31557.75	72.0	3.89	1.0

5 rows × 58 columns

In [11]:
```
#from sklearn.preprocessing import MinMaxScaler
# from sklearn.preprocessing import minmax_scale
# from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
# from sklearn.preprocessing import RobustScaler
# from sklearn.preprocessing import Normalizer
# from sklearn.preprocessing import QuantileTransformer
# from sklearn.preprocessing import PowerTransformer
```

In [12]:
```
x_scaler=StandardScaler()
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.fit_transform(x_test)
```

In [13]:
```
X= x_test
labels_true=y_test
```

In [14]:
```
from sklearn.base import BaseEstimator
class KMeans(BaseEstimator):

    def __init__(self, k, max_iter=100, random_state=0, tol=1e-4):
        self.k = k
        self.max_iter = max_iter
        self.random_state = random_state
        self.tol = tol

    def _e_step(self, X):
        self.labels_ = euclidean_distances(X, self.cluster_centers_,
                                 squared=True).argmin(axis=1)

    def _average(self, X):
        return X.mean(axis=0)

    def _m_step(self, X):
        X_center = None
        for center_id in range(self.k):
            center_mask = self.labels_ == center_id
            if not np.any(center_mask):
                # The centroid of empty clusters is set to the center of
                # everything
                if X_center is None:
                    X_center = self._average(X)
                self.cluster_centers_[center_id] = X_center
            else:
                self.cluster_centers_[center_id] = \
                    self._average(X[center_mask])

    def fit(self, X, y=None):
        n_samples = X.shape[0]
        vdata = np.mean(np.var(X, 0))

        random_state = check_random_state(self.random_state)
        self.labels_ = random_state.permutation(n_samples)[:self.k]
        self.cluster_centers_ = X[self.labels_]

        for i in range(self.max_iter):
            centers_old = self.cluster_centers_.copy()

            self._e_step(X)
            self._m_step(X)

            if np.sum((centers_old - self.cluster_centers_) ** 2) < self.tol *
 vdata:
                break

        return self
```

In [15]:
```
class KMedians(KMeans):

    def _e_step(self, X):
        self.labels_ = manhattan_distances(X, self.cluster_centers_).argmin(
axis=1)

    def _average(self, X):
        return np.median(X, axis=0)
```

In [16]:
```
class FuzzyKMeans(KMeans):

    def __init__(self, k, m=2, max_iter=100, random_state=0, tol=1e-4):
        """
        m > 1: fuzzy-ness parameter
        The closer to m is to 1, the closer to hard kmeans.
        The bigger m, the fuzzier (converge to the global cluster).
        """
        self.k = k
        assert m > 1
        self.m = m
        self.max_iter = max_iter
        self.random_state = random_state
        self.tol = tol

    def _e_step(self, X):
        D = 1.0 / euclidean_distances(X, self.cluster_centers_, squared=True
)
        D **= 1.0 / (self.m - 1)
        D /= np.sum(D, axis=1)[:, np.newaxis]
        # shape: n_samples x k
        self.fuzzy_labels_ = D
        self.labels_ = self.fuzzy_labels_.argmax(axis=1)

    def _m_step(self, X):
        weights = self.fuzzy_labels_ ** self.m
        # shape: n_clusters x n_features
        self.cluster_centers_ = np.dot(X.T, weights).T
        self.cluster_centers_ /= weights.sum(axis=0)[:, np.newaxis]

    def fit(self, X, y=None):
        n_samples, n_features = X.shape
        vdata = np.mean(np.var(X, 0))

        random_state = check_random_state(self.random_state)
        self.fuzzy_labels_ = random_state.rand(n_samples, self.k)
        self.fuzzy_labels_ /= self.fuzzy_labels_.sum(axis=1)[:, np.newaxis]
        self._m_step(X)

        for i in range(self.max_iter):
            centers_old = self.cluster_centers_.copy()

            self._e_step(X)
            self._m_step(X)

            if np.sum((centers_old - self.cluster_centers_) ** 2) < self.tol
 * vdata:
                break

        return self
```

In [17]:
```
from sklearn.utils import check_random_state
from sklearn.metrics.pairwise import euclidean_distances, manhattan_distance
s
fuzzy_kmeans = FuzzyKMeans(k=10, m=4)
fuzzy_kmeans.fit(X)
kmeans = KMeans(k=10)
kmeans.fit(X)

kmedians = KMedians(k=10)
kmedians.fit(X)

# fuzzy_kmeans = FuzzyKMeans(random_state=54,k=2, m=2).fit(x_test)
```

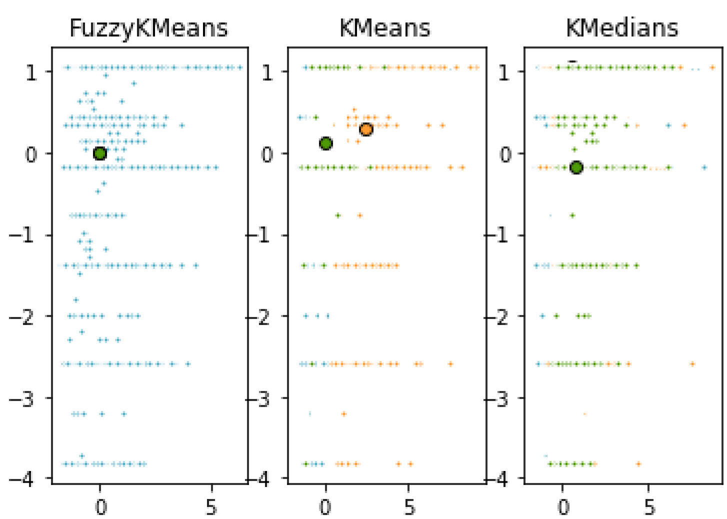
Out[17]:
```
KMedians(k=10, max_iter=100, random_state=0, tol=0.0001)
```

In [18]:
```
fig = plt.figure()
colors = ['#4EACC5', '#FF9C34', '#4E9A06']

objects = (fuzzy_kmeans, kmeans, kmedians)

for i, obj in enumerate(objects):
    ax = fig.add_subplot(1, len(objects), i + 1)
    for k, col in zip(range(obj.k), colors):
        my_members = obj.labels_ == k
        cluster_center = obj.cluster_centers_[k]
        ax.plot(X[my_members, 0], X[my_members, 1], 'w',
                markerfacecolor=col, marker='.')
        ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col
,
                                        markeredgecolor='k', markersize=6)
    ax.set_title(obj.__class__.__name__)

plt.show()
```



In [ ]: