Computational Intelligence - Project 1

An implementation and analysis of Possibilistic C Means

Holt Skinner

Description

In the world of clustering algorithms, the K Means and Fuzzy C-Means Algorithms remain popular choices to determine clusters. The basic K Means clustering algorithm goes as follows.

```
1. Initialize K cluster centers (Random or specifically chosen from data set)
2. Place all points into the cluster of the closest prototype
3. Update memberships and cluster centers
4. Repeat until Clusters Stabilize or until a certain number of iterations.
```

The Fuzzy C-Means Algorithm improves upon K Means by allowing data points to have a membership in more than one cluster, given as a number between 0-1. All of the membership values for a particular data point must add up to one. Possibilistic C Means (PCM) is an algorithm created by Dr. Jim Bezdek and Dr. Jim Keller that eliminates the probabilistic constraint that all membership values must add up to one. This allows points relatively far away from all of the clusters (outliers) to have negligible

membership in all of the clusters. This is an important advantage because noise data can be filtered out without altering the cluster centers. If the outliers were left in, it could drastically shift the cluster centers away from their true location. Despite its advantages, there is currently not an open source python library that supports the PCM algorithm. To solve this, the project will consist of an open source implementation of PCM in Python hosted on GitHub.

Goals

Short-Term

- Code Implementation
- Test with Well-Known Data Set (Iris Data)
- Display Results in a Scatter Chart

Long-Term & Final

- Initizalize Clusters with K-Means and Fuzzy C-Means output.
- Run PCM on NFL Play Data.

Benefits / Expected Outcomes

The ultimate goal for the project is to create a working implementation of the Possibilistic C-Means Algorithm that can be generalized for a multitude of use cases. This implementation will be released to Github to allow it for use by Data Scientists and Machine Learning Specialists, as well as a comparison point for my Senior Capstone Project that compares Machine Learning Models for predicting the outcomes of NFL Games.

Algorithm

Let $X = \{x_1, x_2, \ldots, x_n\}$, where $x_k \in \Re^d$ is the set of vectors to be clustered.

Initialization: Set
C, the number of clusters desired
m, the fuzzifier
ε, the convergence threshold
$V^{(0)} = \left\{v_1^{(0)}, \ldots, v_C^{(0)}\right\}$ an initial set of cluster centers

//Note: The $v_i^{(0)}$ can be chosen randomly from X or through other mechanisms//

Set t = 0
REPEAT
DO FOR each k = 1, . . . , n
IF $d(x_k, v_i) = 0$ for some subset of clusters, i.e., $I_k \neq \phi$
THEN
Set $u_{jk}^{(t)} = 0$ for $j \notin I_k$ and $u_{jk}^{(t)} > 0$ for $j \in I_k$, as in Eq. 8.3b
ELSE
Compute $u_{ik}^{(t)}$ from Eq. 8.3a.
ENDIF
END FOR
Set $t \leftarrow t + 1$
Using $U^{(t-1)}$, estimate $V^{(t)}$ from Eq. 8.4.
UNTIL $\sum_{i=1}^{C} \left\| v_i^{(t)} - v_i^{(t-1)} \right\| < \epsilon$
where $\| * \|$ is any vector norm (like Euclidean).

//Note: There are other stopping criteria, including number of iterations, but this is the most common.//

Where:

$$u_{ik} = \frac{1}{1 + \left(d^2(x_k, v_i)/\eta_i\right)^{1/(m-1)}}$$

and

$$\eta_i = \frac{\sum_{k=1}^{n} u_{ik}^m d^2(x_k, v_i)}{\sum_{k=1}^{n} u_{ik}^m} \quad \text{or} \quad \eta_i = \frac{\sum_{u_{ik} > \alpha} d^2(x_k, v_i)}{\sum_{u_{ik} > \alpha} 1}, \quad \text{for some } 0 < \alpha \leq 1$$

Calculated with the results from Fuzzy C-Means or set equal to 1

Source: Keller, Fundamentals of Computational Intelligence

Libraries Utilized

- Numpy
    - Scientific Computing Python Library
    - Contains Data Structures and Algorithms for mathematical operations
- Scipy
    - Scientific omputing Python Library (Use in conjunction with numpy)
    - Contains Distance Calulation Algorithm used in Fuzzy and Possibilistic C Means
    - Supports different measures for distance including Euclidean and Mahalanobis
- Matplotlib
    - Python Library for plotting graphs and charts
- Scikit Learn
    - Python Library with preloaded datasets
    - Use for comparison and testing.
- Scikit Fuzzy
    - Python Library containing the Fuzzy C Means Algorithm as well as other fuzzy logic operations

Code

Repository: https://github.com/holtwashere/PossibilisticCMeans

cmeans.py

```
import numpy as np
import skfuzzy as fuzz
from scipy.spatial.distance import cdist

def eta(u, d, m):
    um = u ** m
    d = d ** 2
    n = um.dot(d.T) / um.sum(axis=1)
    return n


def update_clusters(x, u, m):
    um = u ** m
    v = um.dot(x.T) / np.atleast_2d(um.sum(axis=1)).T
    return v


def _fcm_criterion(x, v, n, m, metric):

    d = cdist(x.T, v, metric=metric).T

    # Sanitize Distances (Avoid Zeroes)
    d = np.fmax(d, np.finfo(x.dtype).eps)

    exp = -2. / (m - 1)
    d = d ** exp

    u = d / np.sum(d, axis=0, keepdims=1)

    return u, d


def _pcm_criterion(x, v, n, m, metric):

    d = cdist(x.T, v, metric=metric).T
```

```
    d = np.fmax(d, np.finfo(x.dtype).eps)

    d2 = d ** 2

    d2 = d2 / n

    exp = 1. / (m - 1)
    d2 = d2 ** exp
    u = 1. / (1. + d2)

    return u, d


def _cmeans(x, c, m, e, max_iterations,
criterion_function, metric="euclidean", v0=None,
u0=None, d=None):
    """

    # Parameters

    `x` 2D array, size (S, N)
        Data to be clustered. N is the number of data
sets;
        S is the number of features within each sample
vector.

    `c` int
        Number of clusters

    `m` float, optional
        Fuzzifier

    `e` float, optional
        Convergence threshold
```

```
    `max_iterations` int, optional
        Maximum number of iterations

    `v0` array-like, optional
        Initial cluster centers

    # Returns

    `v` 2D Array, size (S, c)
        Cluster centers

    `u` 2D Array (S, N)
        Final partitioned matrix

    `u0` 2D Array (S, N)
        Initial partition matrix

    `d` 2D Array (S, N)
        Distance Matrix

    `t` int
        Number of iterations run

    """

    if not x.any() or len(x) < 1 or len(x[0]) < 1:
        print("Error: Data is in incorrect format")
        return

    # Num Features, Datapoints
    S, N = x.shape

    if not c or c <= 0:
        print("Error: Number of clusters must be at
least 1")
```

```
    if not m or m <= 1:
        print("Error: Fuzzifier must be greater than
1")
        return

    n = 1
    # Initialize the cluster centers
    # If the user doesn't provide their own starting
points,
    if v0 is None:
        # Pick random values from dataset
        xt = x.T
        v0 = xt[np.random.choice(xt.shape[0], c,
replace=False), :]
    else:
        n = eta(u0, d, m)

    # List of all cluster centers (Bookkeeping)
    v = np.zeros((max_iterations, c, S))
    v[0] = np.array(v0)

    # Membership Matrix Each Data Point in eah cluster
    u = np.zeros((max_iterations, c, N))

    # Number of Iterations
    t = 0

    while t < max_iterations - 1:

        u[t], d = criterion_function(x, v[t], n, m,
metric)
        v[t + 1] = update_clusters(x, u[t], m)
        n = eta(u[t], d, m)

        # Stopping Criteria
        if np.linalg.norm(v[t + 1] - v[t]) < e:
```

```
            break

        t += 1

    return v[t], u[t - 1], u[0], d, t


# Public Facing Functions


def fcm(x, c, m, e, max_iterations, metric="euclidean",
v0=None):
    return _cmeans(x, c, m, e, max_iterations,
_fcm_criterion, metric, v0=v0)


def pcm(x, c, m, e, max_iterations, metric="euclidean",
v0=None, u0=None, d=None):
    return _cmeans(x, c, m, e, max_iterations,
_pcm_criterion, metric, v0=v0, u0=u0, d=d)
```

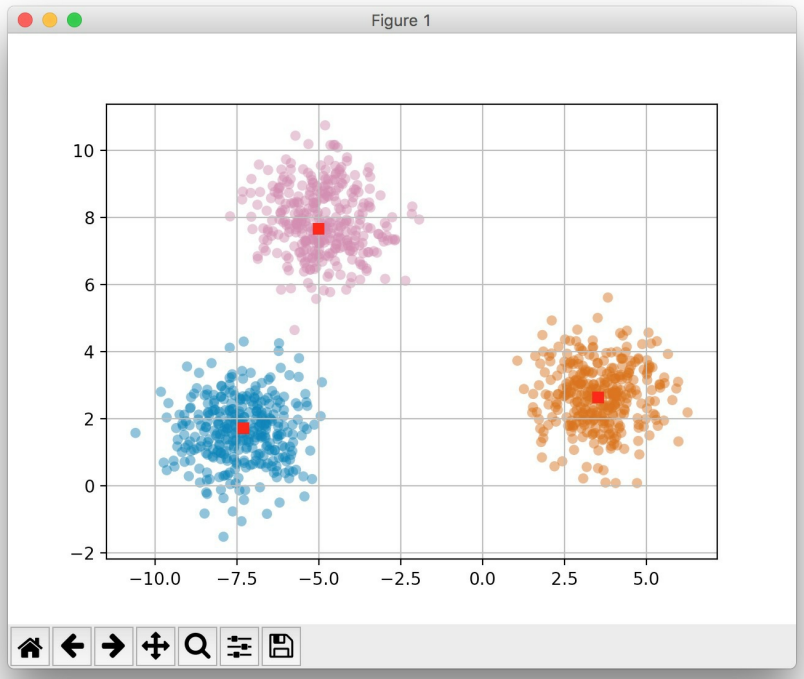
main.py

```
import numpy as np
import sklearn as sk
import sklearn.datasets as ds
import skfuzzy as fuzz
from plot import plot
import cmeans

num_samples = 100000
num_features = 4
```

```
c = 3
fuzzifier = 2
error = 0.001
maxiter = 100

x = ds.make_blobs(num_samples, num_features, c)[0].T

np.random.shuffle(x)

v, u, u0, d, t = cmeans.fcm(x, c, fuzzifier, error,
maxiter)

v, u, u0, d, t = cmeans.pcm(
    x, c, fuzzifier, error, maxiter, v0=v, u0=u, d=d)

plot(x, v, u, c)

iris = ds.load_iris()

labels = iris.target_names
iris = np.array(iris.data)

iris = iris.T

v, u, u0, d, t = cmeans.fcm(iris, c, fuzzifier, error,
maxiter)

v, u, u0, d, t = cmeans.pcm(
    iris, c, fuzzifier, error, maxiter, v0=v, u0=u,
d=d)

plot(iris, v, u, c, labels)
```

Validation and Testing

For each run of the algorithm, the data was first run through the Fuzzy C Means algorithm to obtain reasonable initial cluster centers, and them the data is run through the Possibilistic C Means Algorithm with the cluster centers obtained from FCM. During development of the algorithms, random multi-dimensional gaussian "blobs" were generated with the scikit learn function make_blobs. After using matplotlib functions to graph the clusters and the centers (using different colors to differentiate), the following graphs were produced with random initializations of cluster numbers, features, sample sizes.



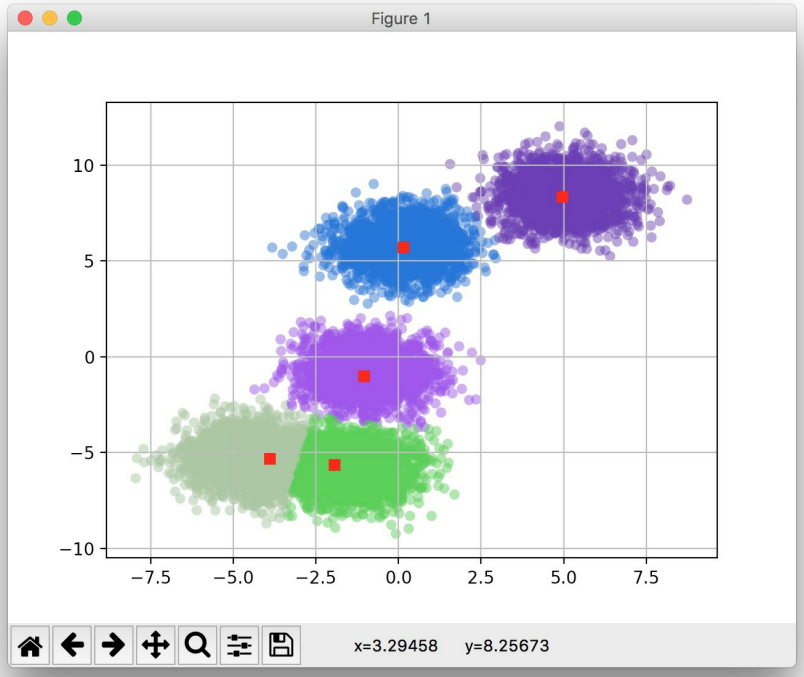


Figure 1
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
x=3.29458
y=8.25673

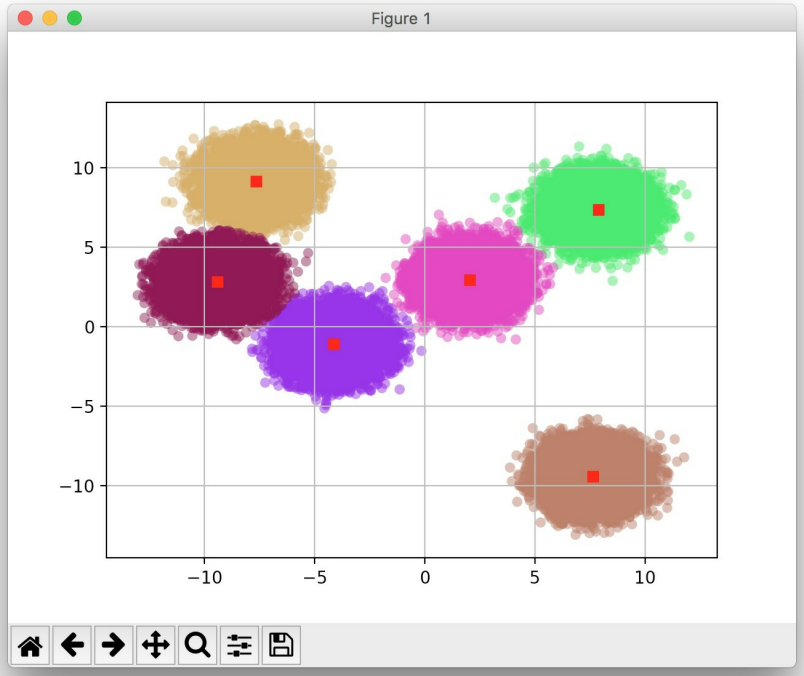
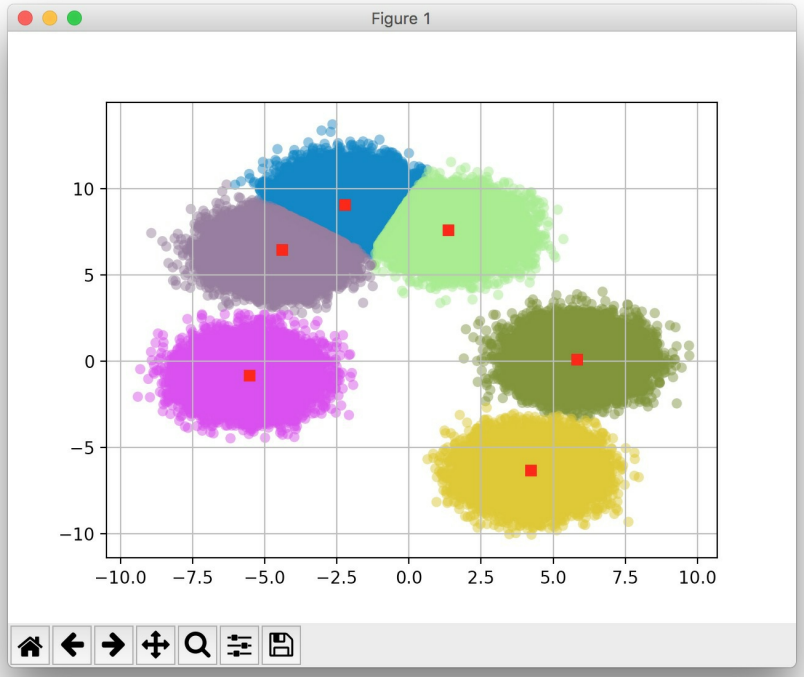
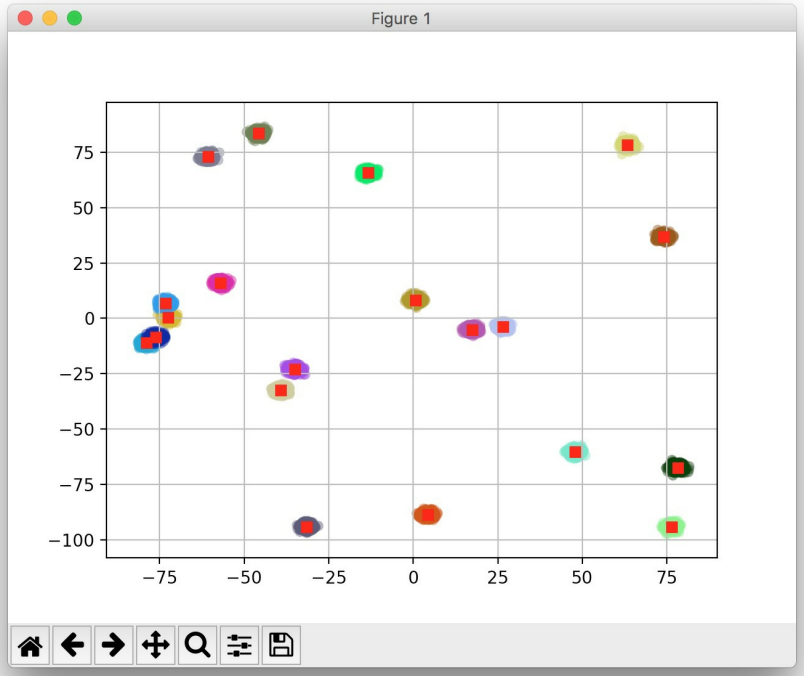


Figure 1
10
5
0
−5
−10
−10
−5
0
5
10

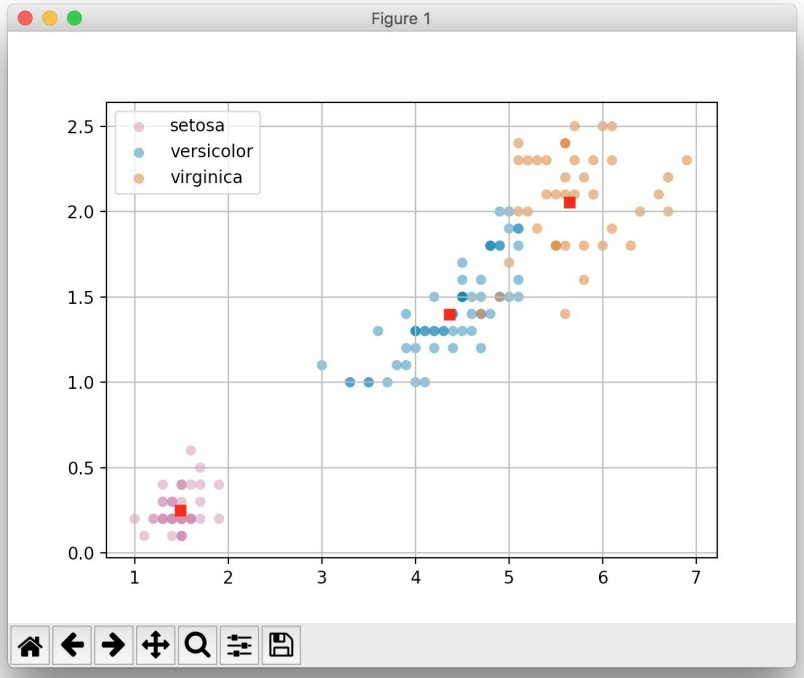




https://en.wikipedia.org/wiki/Iris_flower_data_set

For additional validation and testing, the Iris flower data set as collected by

Ronald Fisher was used, as it is easily available in the scikit learn library and the proper classiciation is already known. The Data was run through both the Project Built PCM and FCM as well as the Fuzzy C-Means as available in the Scikit-fuzzy Open Source Python Library. Since the data has been filtered for noise points, both algorithms provided similar results. The labels in the data itself were used as a validation measure by calculating the percentage correctly identified based on the target labels. The FCM and PCM together averaged 92% accuracy. The especially challenging parts to differentiate were between the versicolor and virignica species because thedata is very close in certain dimensionality. This plot is based on two of the four features in the data set that are easiest to dicern visually.



Analysis

Efficiency

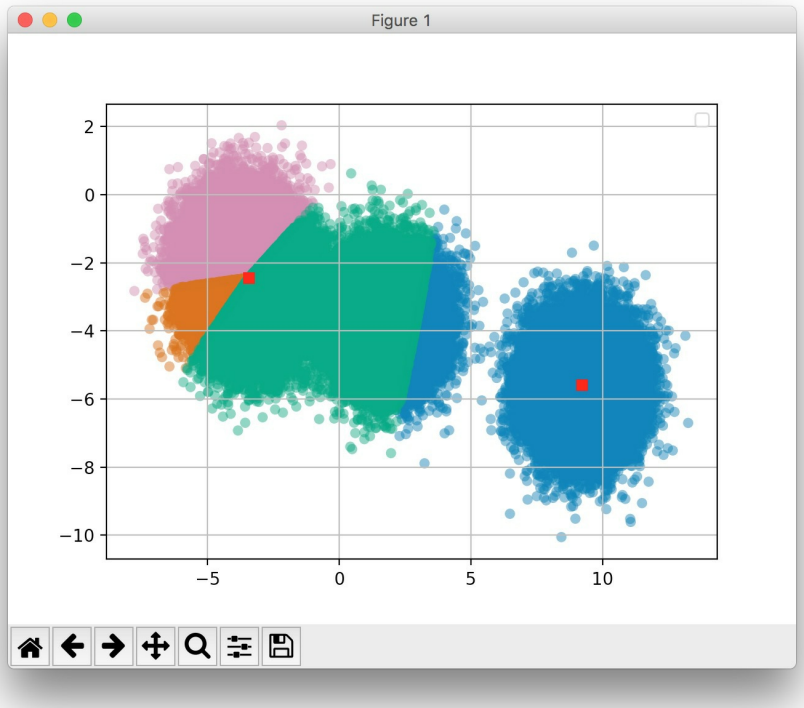
This implementation has allowed a great expansion of knowledge about

not only the logic behind clustering, but also the intricacies of the python language and specifically the numpy library. The implementation lent itself not only to correctness, but efficiency for large datasets. Many of the matrix computations proved challenging to compute. The first attempt consisted of a lot of nested for loops. While easier to visualize, they proved to be very slow once the dataset got above 100 elements and 2 dimensions. The calculations took almost a minute for each dataset. After consulting with experts (StackOverflow), a much more efficient solution was found in the form of the numpy library. Numpy creates python stub calls that link to compiled code in C or C++, which allows much more high performance programming. The most useful element is the numpy array, which is a wrapper for a C array allowing constant lookup time. In Python, Lists are the primary data structure which are made using hashmaps behind the scenes. The extra hash step in the structure greatly slowed down the performance. In its final form, the program takes 1 second to generate the random data and run the algorithm on 100,000 samples in 4 feature space.

Poking and Prying

The primary con of Posssibilistic C Means Algorithm as compared to Fuzzy C Means is its stability and consistency. PCM is much more sensitive to random initialization of cluster centers and to the value of the fuzzifier constant. For all of the images above, the data was run through FCM before hitting the PCM and the fuzzifier was kept at 2, as it provided the most accurate results for these datasets. One issue that arose when running PCM on its own was collision of the cluster centers. If any of the initial cluster centers happened to have values very close to each other, the PCM would have issues moving one of the centers to another part of the data space. This resulted in charts similar to the one below. The cluster center on the left is actually 3 cluster centers with neglibible distance between them. Runnign FCM before PCM resolves this by providing better starting points

rather than random initialization. This bug was a major block in production for a part of the project because the results would be wildly inconsistent between tests.



Further Work

To continue working on the implementation, an ultimate goal is to add a PCM implementation to the scikit fuzzy open source library. The library is currently on GitHub and is accepting pull requests. A next step is to make a clone of the scikit fuzzy library and make modifications to add PCM to the FCM module. In my implementation, I have set up the parameters similar to the existing library to allow for an easier transition. The implementation also uses functional programming concepts to allow the criterion function to be sent as a parameter to the clustering algorithm, since it is the primary differentiator.

Another possible use of this implementation is for my undergraduate capstone project team. We are using various Machine Learning and Pattern

Recodnition Tools to predict the outcome of NFL Games, and it would be interesting to compare the results of the PCM algorithm versus other methods such as a Multi Layer Perceptron or Support Vector Machines.