# Assignment 4 Report: Data Clustering

Annelise Wittenberg & Cody Stammer

([wittenberg.18@osu.edu](mailto:wittenberg.18@osu.edu); [stammer.3@osu.edu](mailto:stammer.3@osu.edu))

## Approach

For this assignment, we chose to import the Orange Library in python to write a program that clusters the dataset. We chose k-means and hierarchical clustering as the algorithms, and as our distance metrics, Euclidean and Manhattan. Initially, we attempted to use the Orange GUI but were limited by it as it couldn’t give us the definite values that we needed.

Using the functions in the Orange library made it possible to run the entire data set using less RAM than in the classification assignment. While observing the CPU and RAM usage graphs, the line stayed very consistently around 24% CPU usage and roughly 3.59GB of RAM. These levels stayed the same even when running the entire data set versus the smaller data set.

## Results

## Issues

## Assumptions

## Work Distribution

Clustering Algorithms

K Means Plot

Dendrogram

Report

Testing

# Data Generation (rand(150,2) + array([.5,.5]),rand(150,2))

dat = vstack(data\_input)

# Computing k means with K = 2

centroids,\_ = kmeans(dat, 2)

# Assign each sample to a cluster

idx,\_ = vq(dat, centroids)

# Some plotting using numpy's logical indexing

plot(dat[idx==0,0],dat[idx==0,1], 'ob',

dat[idx==1,0],dat[idx==1,1], 'or')

plot(centroids[:,0],centroids[:,1], 'sg', markersize=8)

show()

def plot\_scatter(table, km, attx, atty, filename="kmeans-scatter", title=None):

# Plot a data scatter plot with the position of centeroids

plt.rcParams.update({'font.size': 8, 'figure.figsize': [4,3]})

x = [float(d[attx]) for d in table]

y = [float(d[atty]) for d in table]

colors = ["c", "w", "b"]

cs = "".join([colors[c] for c in km.clusters])

plt.scatter(x, y, c=cs, s=10)

xc = [float(d[attx]) for d in km.centroids]

yc = [float(d[atty]) for d in km.centroids]

plt.scatter(xc, yc, marker="x", c="k", s=200)

plt.xlabel(attx)

plt.ylabel(atty)

if title:

plt.title(title)

plt.savefig("%s-%03d.png" % (filename, km.iteration))

plt.close()

def callback(km):

print "Iteration: %d, changes: %d, score: %8.6f" % (km.iteration, km.nchanges, km.score)

plot\_scatter(data\_input, km, "petal width", "petal length", title="Iteration %d" % km.iteration)