BIO IQ Tutorial

Introduction

An **algorithm** is similar to a recipe. More formally, an **algorithm** is a process or set of rules with which a task is completed or with which an input is manipulated to produce an output. For example, we use algorithms for tasks like sorting.

What if there is a task where the algorithm isn’t obvious? What can we do? Well, we often have a large amount of data that we can use to help inform how we produce appropriate outputs from these inputs. As beautifully defined in [1], "Machine Learning provides automated methods of data analysis that can then be used further. Machine Learning is a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty."

Where did the idea for Machine Learning come from?

Traditionally, humans learn how to do certain tasks from experience in the real-world while computers must be instructed to do things. Machine learning is a field which seeks to enable computers and machines to learn from experience. The experience that computers have access to is known as data.

Thus far, there are three different ways that a machine can learn: **supervised learning**, **unsupervised learning**, and **reinforcement learning**. We will focus on the first two methods.

Supervised learning uses **labeled** data as it’s feature set, or the data we initially have to learn patterns from. The labeled data is a set of input-output pairs. In other words, you have input variables (X) and their corresponding output variables (Y) and you want to learn the mapping between these variables.

Unsupervised learning uses **unlabeled** data as it’s feature set, or the data we initially have to learn patterns from. In **unsupervised learning**, we only have a set of inputs, and the goal is to discover interesting patterns in the data.

Some examples of achievements which are direct results of the benefits of machine learning are AlphaGo, a system which can outperform a champion Go player, face detection on the iPhone, and various object detection paradigms. In the past three examples, we see that these tasks are complex, like a game with many rules. Before Machine Learning approaches, progress towards achieving these tasks was minimal.

How do we learn from data?

To get an undestanding of how patterns and relationships may be uncovered in data to then be used for future predictions or decision making, let's dive into some interactive examples.

Each case where we want to use machine learning is unique, but can be boiled down into a few main components:

1. Identify the input/output relationship you want to understand
2. Identify an approximate model from patterns in the data that explains this relationship
3. Verify the quality of your model
4. Use the model to predict future data or influence future decisions

We will learn that these five steps are useful to guide the development of many machine learning models. We can expand these steps to more specifically reference certain problem spaces, however, these five underlying steps will still be present.

First we will look at an example of **supervised learning**, where we have a set of input/ouput pairs and we would like to find a relationship between these inputs and outputs. We will use a very simple synthetic data set.

Tutorial 1: Least Squares Linear Regression (LSLR)

In this tutorial, we will go through the steps necessary to fit a LSLR model to data. This will help orient you with the basic steps machine learning tasks require with a simple, easy to understand example.

This tutorial is adapted from [2], and may be referred to to further explore this example.

The package that is used in this tutorial is referenced in this github repository [3], and may be referred to for further exploration of the structure of the class, LinearRegression, which is implementing Ordinary Least Squares Linear Regression.

As introduces in [2], we will complete five basic steps to implement linear regression in python:

1. Import the packages and classes necessary for implementation.
2. Identify the data you would like to explore, and ensure it is in the proper format
3. Create the model and fit it with the existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

These steps are reminiscent of those introduced early with a few more specifics which guide real-world development in coding platforms.

Step 1: Import Packages and Classes

First, we make sure to import the packages and classes necessary to run the tutorial!

In [33]:
```
import numpy as np # a package for manipulating numbers and using arrays
from sklearn.linear_model import LinearRegression # a package with an optimized LinearRegression class
import matplotlib.pyplot as plt # a package with plotting capabilities
```

Step 2: Provide Data

Next we load and prepare our data. In this case, we are using synthetic data, so there is not much we need to do!

In [34]:
```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1)) # synthetic x-
data
y = np.array([5, 20, 14, 32, 22, 38]) # synthetic y-data

# Use these commands to look at the arrays after the .reshape functio
n
# print(x)
# print(y)
```

Step 3: Create and Fit the Model

Next, we take advantage of the `LinearRegression()` class that we downloaded from the `sklearn` package. First, we create an instance of the `LinearRegression()` class. Next, we use the `.fit(x,y` method from this class.

In [35]:
```
model = LinearRegression() # this line is creating an instance of the
class LinearRegression
model.fit(x, y) # this line is using a method within the LinearRegres
sion class which fits the model
```

Out[35]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

We can actually condense these first two lines to one line, and get an equivalent output!

In [36]:
```
model = LinearRegression().fit(x, y) # syntactically, this line is eq
uivalent to the two lines above
```

Step 4: Get Results

After fitting the model, we would like to explore the results and performance of the model!

In [37]:
```
r_sq = model.score(x, y) # this line helps show us how good our model
is
print('coefficient of determination:', r_sq) # print the model score
```

```
coefficient of determination: 0.715875613747954
```

From using this `LinearRegression()` class and its method `.fit(x, y)` we have output model which has different methods. Above, we looked at the method `.score(x, y)` which output the R^2 value for this set of input output pairs with this trained model.

Below, we extract the learned y-intercept, and slope. We can use these values to then predict unknown input/output pairs in the future!

In [38]:
```
print('intercept:', model.intercept_) # print the y-intercept
print('slope:', model.coef_) # print the slope
```

```
intercept: 5.633333333333329
slope: [0.54]
```

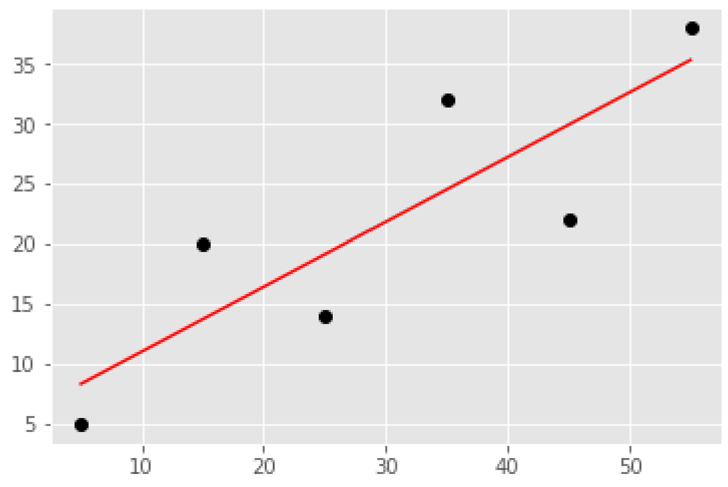
Step 5: Make Predictions

After fitting the model, we would now like to visualize the results and then use the results to predict future outputs of new inputs!

In [39]:
```
plt.plot(x, y, 'ko') # plot the original data in black ('k') circles
 ('o')
y_pred_package = model.predict(x) # use a predefined method to predic
t outputs from a set of inputs
print('predicted response from package:', y_pred_package, sep='\n') #
print prediction results
plt.plot(x, y_pred_package, 'r') # plot the learned relationship in a
red ('r') line (default)
```

```
predicted response from package:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333
333]
```

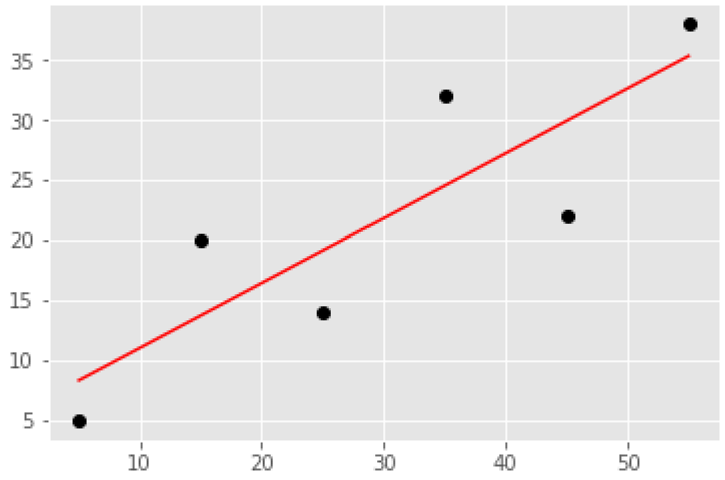
Out[39]:
```
[<matplotlib.lines.Line2D at 0x7f1adf401630>]
```



In [40]:
```
plt.plot(x, y, 'ko')# plot the original data in black ('k') circles
 ('o')
y_pred_user = (model.coef_ * x) + model.intercept_ # generate the sam
e predicted relationship using paramteres
print('predicted response from user:', y_pred_package, sep='\n') # pr
int prediction
plt.plot(x, y_pred_user, 'r') # plot the learned relationship in a re
d ('r') line (default)
```

```
predicted response from user:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333
333]
```

Out[40]:
```
[<matplotlib.lines.Line2D at 0x7f1adf3802b0>]
```



We have successfully learned an approximate relationship between synthetic inputs, x, and outputs, y. We did this through use of a **supervised learning** method, Least Squares Linear Regression. Next, let's explore a method of **unsupervised learning**

Tutorial 2: K-means

In this second tutorial, we will explore a method of **unsupervised learning** known as K-means clustering. We will use an MEG data set for application of this method.

To get some extra help with the underlying training procedure used in K-means, please watch [4], as it is a great explanation.

Similar to the procedure in LSLR, we will follow a few main steps:

1. Import and download all necessary packages and classes
2. Identify and prepare the data you desire to learn about
3. Create the model and fit it with the existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

Import Packages and Classes

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import scipy.io as sio
import time
```

Step 2: Provide Data

In [42]:
```
sio.whosmat('MEG_decoding_data_final.mat')
```

Out[42]:
```
[('stim_ID', (1, 125), 'double'),
 ('cat_ID', (1, 125), 'double'),
 ('MEG_data', (125, 306), 'double'),
 ('train_data', (40, 306), 'double'),
 ('train_cat_labels', (1, 40), 'double'),
 ('train_stim_labels', (1, 40), 'double'),
 ('test_data', (10, 306), 'double'),
 ('test_cat_labels', (1, 10), 'double')]
```

In [43]:
```
sio.whosmat('kmeans_results.mat')
```

Out[43]:
```
[('IDX', (125, 1), 'double')]
```

In [44]:
```
mat_1 = sio.loadmat('MEG_decoding_data_final.mat')
mat_2 = sio.loadmat('kmeans_results.mat')
```

In [45]:
```
MEG = mat_1['MEG_data']
print(MEG)
```

```
[[ 2.81111470e-14  8.36154238e-13 -4.85679006e-13 ...  4.95203710e-15
  -2.60111669e-12 -2.60539527e-14]
 [ 1.70528469e-14  2.64439255e-13  3.29650207e-13 ... -2.71882223e-14
  -5.56628425e-13 -5.21515479e-13]
 [ 4.73273365e-14  8.61060867e-13  8.29028754e-13 ... -1.81926512e-14
   2.22443615e-13 -7.69609477e-13]
 ...
 [-5.00628992e-14  8.99357787e-13 -1.91885072e-12 ... -4.38943615e-14
  -1.52066217e-12 -1.50715828e-12]
 [-6.99054154e-14  4.31405166e-13  2.93901717e-14 ... -5.18895469e-17
  -8.79982248e-13 -8.20168690e-13]
 [ 1.10738544e-14 -4.72107713e-13  1.65955268e-12 ...  2.05967210e-14
  -6.91858051e-13 -5.68636666e-13]]
```

In [46]:
```
current_data = MEG[:,[199, 232]]
print(current_data[:,0])
```

```
[-1.47673282e-12 -5.84458894e-13 -1.19851196e-12 -5.28628939e-13
 -2.27152044e-12  5.41376348e-14 -1.16936764e-12  8.92152591e-13
  2.28172401e-12 -2.38448467e-13 -5.89475389e-13  1.29159010e-12
 -1.40499632e-12 -9.07388253e-13 -1.28462084e-12 -2.53050955e-12
 -1.35498643e-12 -1.58556310e-12 -3.99028611e-14 -2.52023621e-12
 -2.67484035e-12 -4.91906117e-12 -2.93001296e-12 -3.98469965e-13
 -3.59499752e-12 -2.23143081e-12 -3.11551470e-12  3.55305588e-12
  8.20346690e-14 -2.84337331e-13 -2.18372734e-12 -3.78489798e-12
 -2.10929842e-12 -1.59836249e-12 -3.35795226e-12 -2.05992469e-12
 -1.88808850e-12 -2.25113475e-12 -1.37166183e-12 -1.56521572e-12
 -1.94248132e-12 -5.99979725e-13 -2.70451990e-12 -1.89581499e-12
 -4.29852101e-12 -1.52667897e-12 -1.32465235e-12 -5.24227045e-13
 -1.45172172e-12  1.50160499e-12 -1.53200922e-12 -2.12557436e-12
 -2.69963358e-12 -4.23409572e-12 -4.01035495e-13 -1.46265664e-12
 -8.59366598e-13 -4.88426192e-12 -6.30934905e-12 -5.00368136e-12
 -4.08242723e-13 -5.37755650e-13  9.74067713e-15 -1.74451850e-12
 -1.08306275e-12 -2.13604146e-12 -2.80120374e-12 -2.65876332e-12
 -2.73891666e-12 -6.64968324e-13  1.56561634e-12 -2.28766016e-12
 -2.26361454e-12 -1.18873308e-12 -4.53405860e-12 -2.82308845e-12
 -2.18826679e-13 -1.41936717e-12 -6.63012105e-13  2.72902113e-13
 -2.88659119e-12  5.16578353e-13 -2.57429866e-12 -1.45076376e-12
 -2.53258989e-12 -7.69792662e-13 -1.37383673e-12 -5.46474607e-13
  1.90769467e-12  3.62929299e-13 -2.99757456e-12 -3.37863752e-12
 -2.43260996e-12 -1.41326377e-12 -2.97404984e-12 -1.80724051e-12
  1.37147326e-12  5.01425747e-13 -3.31620289e-12 -3.84685884e-12
 -3.40508040e-12 -3.10607628e-13 -3.81275197e-12 -4.04577385e-12
 -1.30938474e-12 -2.02078338e-12 -4.08412288e-12 -1.05688194e-12
 -4.42009149e-13 -3.78634120e-12 -1.75562616e-12 -2.52094091e-12
 -3.91822672e-12 -2.55282036e-12 -3.76367404e-12 -3.25095843e-12
 -8.67891023e-13 -1.79491474e-12 -2.37171581e-12 -1.09894919e-14
 -2.34118219e-12 -2.83474017e-12 -5.29486416e-12 -2.04659416e-12
 -2.00555764e-12]
```

Step 3: Create and Fit the Model

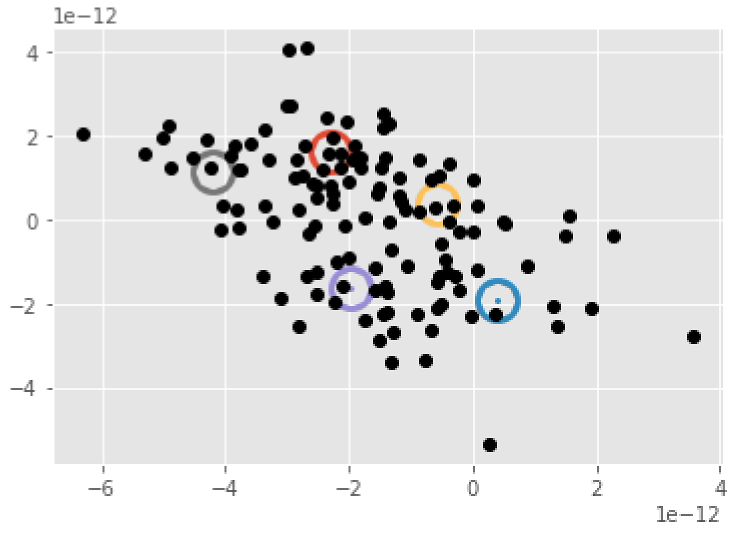
In [47]:
```
Kmean_MEG = KMeans(n_clusters = 5).fit(current_data)
Kmean_MEG.fit(current_data)

# What could you re-write the above two lines to be to generate the s
ame results?
Kmean_MEG = KMeans(n_clusters = 5).fit(current_data)
```

In [48]:
```
Kmean_MEG.cluster_centers_
```

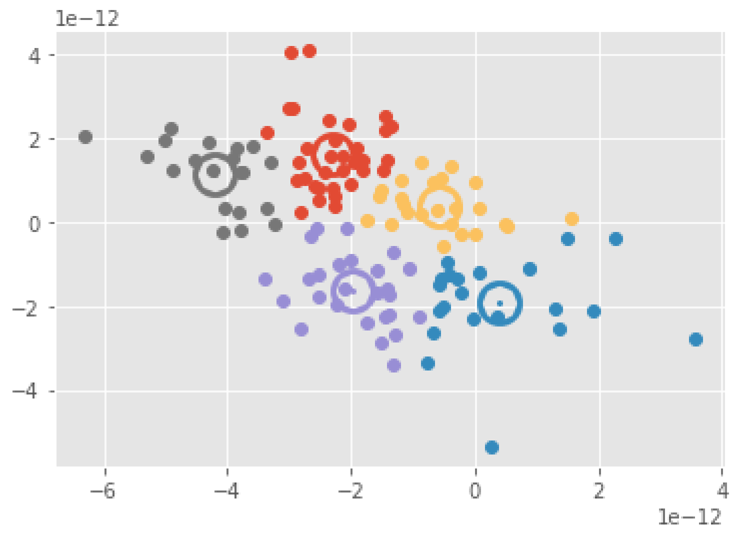
Out[48]:
```
array([[-2.28236064e-12,  1.63098347e-12],
       [ 4.01275255e-13, -1.93948916e-12],
       [-1.95708705e-12, -1.62706465e-12],
       [-4.20203258e-12,  1.14788738e-12],
       [-5.66605571e-13,  3.84670341e-13]])
```

Step 4: Get Results

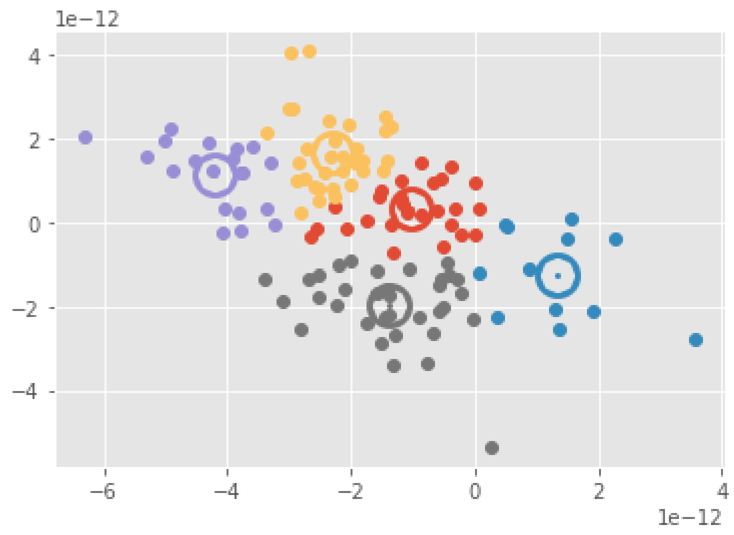
In [49]:
```
plt.style.use('ggplot') # we use the style 'ggplot' because it is pre
tty

plt.plot(current_data[:,0], current_data[:,1], 'ok') # plot the isola
ted data from out MEG file

# this for loop iterates through some nice colors
for i, color in enumerate(plt.rcParams['axes.prop_cycle']):
    if i == 5: break
    plt.scatter(Kmean_MEG.cluster_centers_[i,0], Kmean_MEG.cluster_ce
nters_[i,1], linewidth = 20, marker = '.', color = color['color'])
```

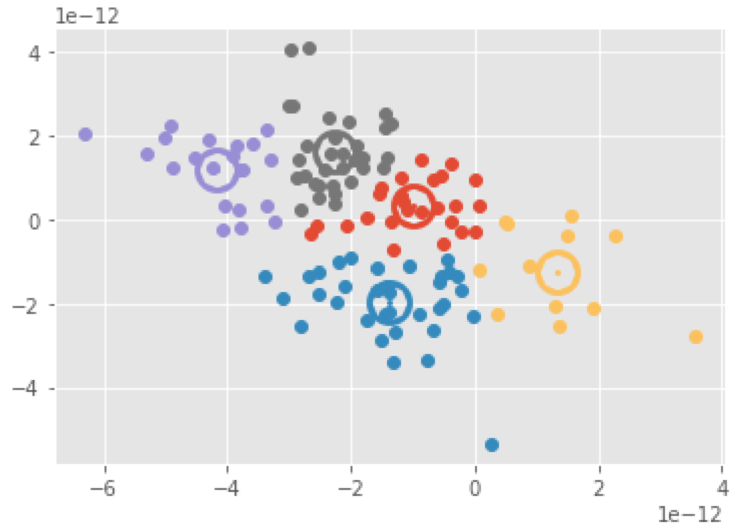


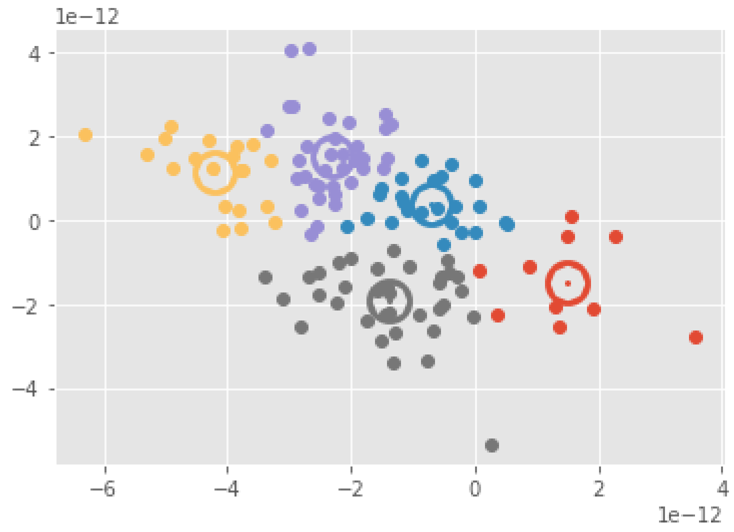
In [50]:
```
lab = Kmean_MEG.labels_

for i, color in enumerate(plt.rcParams['axes.prop_cycle']):
    if i == 5: break
    idx_tmp = np.nonzero(lab == i)
    pt_tmp = current_data[idx_tmp]
    plt.plot(pt_tmp[:,0], pt_tmp[:,1], linewidth = 0, marker = 'o', c
olor = color['color'])
    plt.scatter(Kmean_MEG.cluster_centers_[i,0], Kmean_MEG.cluster_ce
nters_[i,1], linewidth = 20, marker = '.', color = color['color'])
```

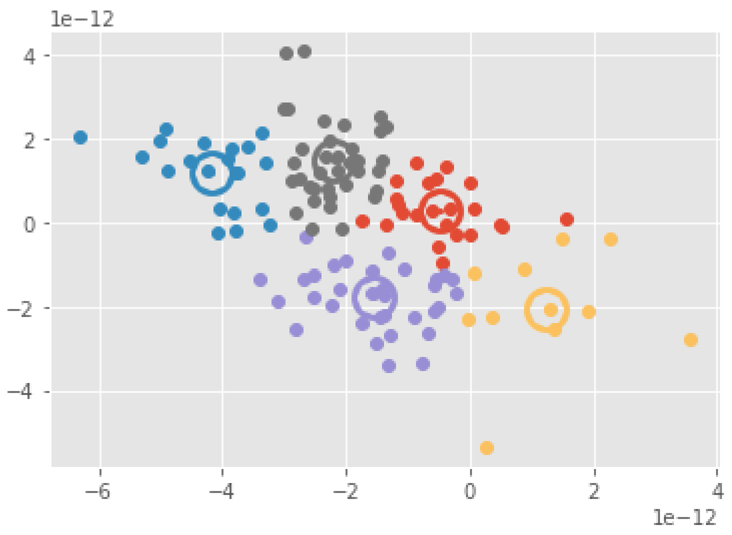


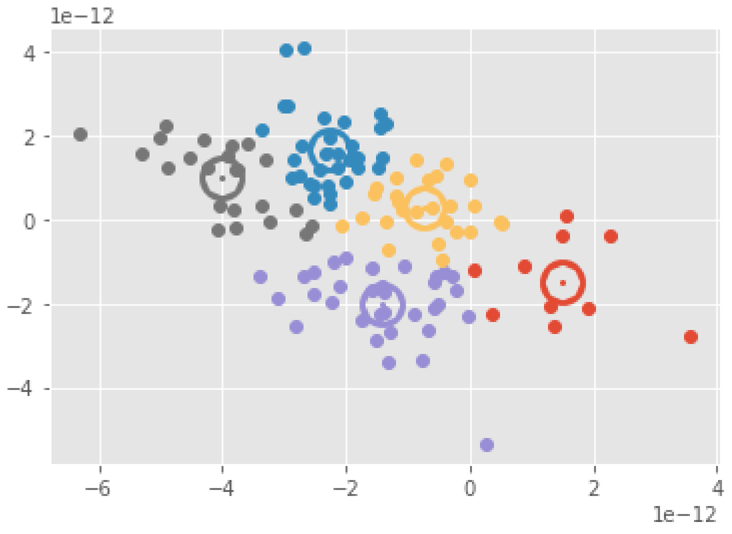
In [51]:
```
for i in range(0,5):

    Kmean_MEG = KMeans(n_clusters = 5)
    Kmean_MEG.fit(current_data)
    Kmean_MEG.cluster_centers_

    lab = Kmean_MEG.labels_

    for i, color in enumerate(plt.rcParams['axes.prop_cycle']):
        if i == 5: break
        idx_tmp = np.nonzero(lab == i)
        pt_tmp = current_data[idx_tmp]
        plt.plot(pt_tmp[:,0], pt_tmp[:,1], linewidth = 0, marker = 
'o', color = color['color'])
        plt.scatter(Kmean_MEG.cluster_centers_[i,0], Kmean_MEG.cluste
r_centers_[i,1], linewidth = 20, marker = '.', color = color['color'
])

    plt.show()
```


1e−12
4
2
0
−2
−4
−6
−4
−2
0
2
4
1e−12


1e−12
4
2
0
−2
−4
−6
−4
−2
0
2
4
1e−12


1e−12
4
2
0
−2
−4
−6
−4
−2
0
2
4
1e−12





Step 5: Make Predictions

We do not explicitly explore step 5 in this tutorial, but if we did, how might we use the K-means Clustering algorithm to generate predictions about new data?

Sources

1. Murphy, Kevin P.. Machine Learning: A Probabilistic Perspective, MIT Press, 2012. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/pensu/detail.action?docID=3339490 (http://ebookcentral.proquest.com/lib/pensu/detail.action?docID=3339490).
2. https://realpython.com/linear-regression-in-python/#linear-regression (https://realpython.com/linear-regression-in-python/#linear-regression)
3. https://github.com/scikit-learn/scikit-learn/blob/7813f7efb/sklearn/linear_model/base.py#L367 (https://github.com/scikit-learn/scikit-learn/blob/7813f7efb/sklearn/linear_model/base.py#L367)
4. A great video on the training procedure for K-means clustering can be found at https://www.youtube.com/watch?v=_aWzGGNrcic&feature=youtu.be (https://www.youtube.com/watch?v=_aWzGGNrcic&feature=youtu.be).