A2

February 3, 2024

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("A2.ipynb")
```

1 Assignment 2 (10 Points in total, will be 10% of your grade)

1.1 How to complete assignments

There are both code and math/text components to these assignments.

1.1.1 Math/text responses

Whenever you see:

`_Type your answer here, replacing this text._`

You should editing this Markdown cell and insert your answers.

If the question is asking for math use LaTeX. We understand not everyone is fluent in LaTeX but we think this is a good skill to learn. Here is a fantastic tutorial from CalTech about using LaTeX in Jupyter Notebook.. You could also find various LaTeX tutorials and cheat sheets online.

Some things to keep in mind: Jupyter's Markdown-LaTeX will display things that don't work when you turn in the assignment. Here are some generic ideas that we find prevent trouble down the road. 1. Test your output to see if it works well when you submit. So execute the otter grader export and check the PDF to make sure it looks right!!! 1. Avoid `$x=1$` use `$$x=1$$` instead 1. No trailing space after the first dollar sign and before the second dollar sign. i.e. don't do `$ $ x = 5 $ $` use `$$x=5$$` instead 1. Markdown lists in the same cell after a math environment don't seem to work, and we don't know why 1. Avoid `\align` use `\aligned` instead

An example:

Question: Given a triangle with sides x, y, z where $x = 3$, $y = 4$ calculate z Your answer in the markdown should be

`$$z = \sqrt{x^2 + y^2} = \sqrt{3^2+4^2} = \sqrt{25} = 5$$`

which will render as

$$z = \sqrt{x^2 + y^2} = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$$

These responses will mostly be manually graded. There is leeway for individual style in notation, so don't worry too much as long as you don't write anything that is false.

1.1.2 Code responses

Whenever you see:

`answers = ...`

You need to replace this section with some code that answers the questions and meets the specified criteria. Make sure you remove the 'raise' line when you do this (or your notebook will raise an error, regardless of any other code, and thus fail the grading tests).

You should write the answer to the questions in those cells (the ones with `...`), but you can also add extra cells to explore / investigate things if you need / want to.

Any cell with `grader.check("Question_id")` statements in it is a test cell. You should not try to change or delete these cells. Note that there might be more than one assert that tests a particular question. Once you run the cell, you will see the public tests we provided for you.

If a test does fail, reading the error that is printed out should let you know which test failed, which may be useful for fixing it.

Note that some cells, including the test cells, may be read only, which means they won't let you edit them. If you cannot edit a cell - that is normal, and you shouldn't need to edit that cell.

2 Q1: Mixture of Bernoullis (1 Point)

2.1 Q1.1 Mixture of Distributions

Consider a mixture distribution of the form

$$p(x) = \sum_{k=1}^{K} \pi_k p(x|k)$$

where you may assume the elements of x are discrete, and $\vec{\pi}$ is a k dimensional vector that satisfy the following condition $\sum_k \pi_k = 1$. Denote the mean and covariance of $p(x|k)$ by μ_k and Σ_k, respectively. Show that the mean of the mixture distribution is given by the following equation:

$$\mathbb{E}[\mathrm{x}] = \sum_{k=1}^{K} \pi_k \mu_k$$

To get there you may want to start with the definition of the mixture $p(x)$ above. Reminder, the generic definition of expectation:

$$\mathbb{E}[\mathrm{x}] = \sum_{x} x p(x)$$

Points: 0.3

$$\mathbb{E}[\mathrm{x}] = \sum_{x} x p(x)$$

Illustration of Bernoulli Mixture Model on MNIST dataset We illustrate the Bernoulli mixture model to model handwritten digits from the MNIST dataset. Here the digit images have been turned into binary vectors by setting all elements whose values exceed 0.5 to 1 and setting the remaining elements to 0. A sample of this binary-valued MNIST is shown below for just the digits 2, 3, and 4. In other words, the only two possible pixel intensities of the MNIST is now 0 (fully black) and 1 (fully white). In this setting, we can consider each individual pixel as a bernoulli random variable.

This approach would fit each digit with a Bernoulli parameter vector as long as the number of pixels in an image. And we'd converge to the correct parameter vectors by running iterations of the EM algorithm.

2.2 Q1.2 Mixture of Bernoulli Distribution

Consider the joint distribution of latent and observed variables for the Bernoulli distribution obtained by forming the product of $p(x|z,\mu)$ (given by Bishop equation (9.52)):

$$p(x|z,\mu) = \prod_{k=1}^{K} p(x|\mu_k)^{z_k}$$

and $p(z|\pi)$ (given by Bishop equation (9.53)):

$$p(z|\pi) = \prod_{k=1}^{K} \pi_k^{z_k}$$

Show that if we marginalize this joint distribution with respect to z, then we obtain the following equation:

$$p(x|\mu,\pi) = \sum_{k=1}^{K} \pi_k p(x|\mu_k)$$

Hint: you might want to read the pages of Bishop noted above. Also when we say "marginalize with respect to z", you should review what that means and start with the definition of marginalizing over a latent variable.

Points: 0.3

Type your answer here, replacing this text.

2.3 Q1.3 Mixture of Bernoulli Distribution M-step

Show that if we maximize the expected complete-data log likelihood function (Bishop equation (9.55))

$$\mathbb{E}_z[\ln p(\mathrm{Z},\mathrm{X}|\mu,\pi)] = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\Big\{\ln \pi_k + \sum_{i=1}^{D}[x_{ni}\ln \mu_{ki} + (1-x_{ni})\ln(1-\mu_{ki})]\Big\}$$

for a mixture of Bernoulli distributions with respect to μ_k, we obtain the M-step equation below:

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})x_n,$$

that is, the k-th mean's MLE is a $\gamma(z_{nk})$ weighted mean of the entire dataset.

Hint: you will want to start with the idea that we always go to when we want to maximize a convex function (as log likelihood is indeed convex). Make sure you are doing this process with respect to the variable you are trying to maximize.

Points: 0.4

Type your answer here, replacing this text.

3 Q2: K-Means (3.5 Points)

Algorithm Description Suppose we have a dataset $\mathcal{D} = \{x_1, \ldots, x_N\}$ consisting of N observations of a D-dimensional variable x. In clustering, the distortion measure of the cluster is given by

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n - \mu_k||^2$$

where μ_k is a prototype associated with the k^{th} cluster, and $r_{nk} \in \{0,1\}$ is a binary indicator variables where $k = 1, \ldots, K$ describing which of the k cluster the data point x_n is assigned to, so that if data point x_n is assigned to cluster k then $r_{nk} = 1$ and $r_{nj} = 0$ for $j \neq k$.

The objective of the K-Means algorithm is to find values for the r_{nk} and the μ_k so as to minimize the distortion measure, i.e.

$$\arg\min_{r,\mu} \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n - \mu_k||^2$$

Note: Compared with the objective function you saw in the lecture slides $\arg\min_C \sum_{i=1}^{k}\sum_{x_j \in C_i}||x_j - \mu_{C_i}||^2$ this is a more nuanced way to express the K-means objective. In this question, we will explore the underlying update rules of the K-Means algorithms.

Algorithm Pseudocode

1. Choose some initial values for the μ_k (Initialize the Centroids)
2. Repeat until converged:
    1. Keeping μ_k fixed, minimizing J w.r.t. r_{nk} **(E-Step)**
    2. Keeping r_{nk} fixed, minimizing J w.r.t. μ_k **(M-Step)**

Optimizing for J For the **(E-Step)**, when the μ_k is fixed, it is rather simple to minimize the function value of J since J is a linear function of r_{nk}. The terms involving different n are independent and so we can optimize for each n separately by choosing r_{nk} to be 1 for whichever value of k gives the minimum value of $||x_n - \mu_j||^2$. In other words, we simply assign the nth data point to the closest cluster center.

3.1 Question 2.1 Update Rule for μ_k

Once we minimizing J w.r.t. r_{nk} while fixing μ_k, we will continue to minimize J w.r.t. μ_k fixing r_{nk} fixed. Prove that

$$\hat{\mu}_k = \arg\min_{\mu_k} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

Hint: Take the derivative of J w.r.t. μ_k, and find the critical point.

Points: 0.4

Type your answer here, replacing this text.

The denominator in this expression is equal to the number of points assigned to cluster k, and so this result has a simple interpretation, namely set μk equal to the mean of all of the data points x_n assigned to cluster k. For this reason, the procedure is known as the K-means algorithm.

For the rest of the question, you will implement your own version of the K-Means clustering algorithm. To begin with, let's create some toy data. This is borrowed from the lecture notebook.

Data Generation

[ ]:
```
# things we will need to do stuff in this notebook
import doctest
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.datasets import make_blobs
from scipy.stats import multivariate_normal
from tqdm import tqdm
from matplotlib.patches import Ellipse

# two useful data viz libraries
import matplotlib.pyplot as plt
import seaborn as sns
# setup plotting in a notebook in a reasonable way
%matplotlib inline
```

```
%config InlineBackend.figure_format='retina'

# default figure aesthetics I'll be using,
# there are other choices, see seaborn docs
sns.set_style("white")
sns.set_context("notebook")
```

[ ]:
```
# Generating Dataset

points, labels = make_blobs(
    cluster_std=7.0, n_samples=1000, n_features=2, random_state=42, centers=[␣
 ↪[20,20], [-20,0], [20,-20]]
)

data_KMeans = pd.DataFrame(points,columns=['x','y'])

# Define a helper function
def plot_cluster(data:pd.DataFrame, km=None, plot_init=False, ax=None):
    """
    helper function to plot out the clusters. When km is unspecified, it will
    plot out the dataset. When km is specified, it will plot out the
    results of the km's cluster.
    """
    # Take a look at the cluster
    fig = plt.figure(figsize=(8, 8))
    ax = plt.axes()
    if km is None:
        sns.scatterplot(data=data,x='x',y='y',ax=ax);
    else:
        sns.scatterplot(data=data,x='x',y='y',ax=ax, hue=km.membership,␣
 ↪palette='Paired');
        plt.scatter(
                km.centroids[:, 0],
                km.centroids[:, 1],
                s=100, marker=",", c="r",
                label="Centroids"
            )
        plt.legend()
    if plot_init:
        plt.scatter(
            km.initial_centroids[:, 0],
            km.initial_centroids[:, 1],
            s=100, facecolors='none', edgecolors='r',
            label="Initial Centroid"
        )
        plt.legend()
    plt.axis('equal');
```

```
    plt.axis('off');

plot_cluster(data_KMeans)
```

3.2 Question 2.2 K-Means Implementation

Implement each methods of the class `MyKMeans` below. We've provided doctests (public tests) for each of the methods. But be careful… just because you pass the doctests doesn't mean you're good. These tests provide basic sanity checks, but do not extensively test the correctness of your implementation. If you want to check the validity of your code, you should write your own tests – either high-level tests, such as examining the end results of the K-Means clusters, or low-level tests, such as asserting each data is in the correct shape and datatype.

Points: 2.1

[ ]:
```
class MyKMeans():
    """
    Your own KMeans implementation
    """

    def __init__(self, k) -> None:
        # this hints you about the type of each arguments
        self.k = k
        self.data = np.array([[]])
        self.initial_centroids = np.array([[]])
        self.centroids = np.array([[]])
        self.membership = np.array([])

    def euc_distances(self, target_pt:np.ndarray, pts:np.ndarray) -> np.ndarray:
        """
        Calculating the euclidean distances between a single target point and
        a numpy array of datapoints. Return a numpy array of distances.

        Doctest/Example:
        >>> km1 = MyKMeans(k=1)
        >>> target_pt = np.array([0,0])
        >>> pts = np.array([[0,0], [1,0], [0,1]])
        >>> km1.euc_distances(target_pt, pts)
        array([0., 1., 1.])
        """
        # WRITE YOUR CODE HERE
        ...

    def _load_data(self, data) -> None:
        """
        internal methods. assign data to self.data

        Doctest/Example:
```

```
        >>> km1 = MyKMeans(k=1)
        >>> data = np.array([[0,0],[1,1]])
        >>> km1._load_data(data)
        >>> np.isclose(data, km1.data).all()
        True
        >>>
        """
        self.data = data

    def _init_centroids(self):
        """
        internal methods. randomly initialize k centroids by sampling k points␣
↪without replacement
        from the dataset. Assign the centroids to the self.centroids

        Doctest/Example:
        >>> km2 = MyKMeans(k=2)
        >>> target_pt = np.array([0,0])
        >>> data = np.array([[0,0], [0,0], [0,0]])
        >>> km2._load_data(data)
        >>> km2._init_centroids()
        >>> km2.centroids.shape
        (2, 2)
        """
        # we provided this for you
        idx = np.random.choice(self.data.shape[0], self.k, replace=False)
        self.centroids = self.data[idx]
        self.initial_centroids = self.data[idx]

    def E_step(self) -> None:
        """
        Compute the E-step of the K-Means. This is calculating membership array
        that indicates the membership of the data at that corresponding index.

        For example, if self.membership is [0, 0, 1], this means that the first␣
↪and
        the second data points are closer to centroid 0 (self.centroid[0]), and␣
↪the
        third data point is closer to the center 1 (self.centroid[1])

        Doctest/Example:
        >>> km2 = MyKMeans(k=2)
        >>> data = np.array([[0,0], [0,0], [0,0]])
        >>> km2._load_data(data)
        >>> km2._init_centroids()
        >>> km2.E_step()
        >>> km2.membership
```

```
        array([0, 0, 0])
        """
        # WRITE YOUR CODE HERE
        ...

    def M_step(self) -> None:
        """
        Compute the M-step of the K-means. Update the self.centroids according
        to the new membership array.

        >>> km2 = MyKMeans(k=2)
        >>> np.random.seed(42)
        >>> data = np.array([[0], [0], [10], [10]])
        >>> km2._load_data(data)
        >>> km2._init_centroids()
        >>> km2.E_step()
        >>> km2.M_step()
        >>> km2.centroids.shape
        (2, 1)
        >>> np.isclose(km2.centroids, np.array([[0], [10]])).all()
        True
        """
        # WRITE YOUR CODE HERE
        ...


    def fit(self, data, iteration=100) -> np.ndarray:
        """
        Run the K-Means algorithm based on the dataset
        """
        # we provided this for you
        self._load_data(data)
        self._init_centroids()
        for _ in range(iteration):
            self.E_step()
            self.M_step()
        return self.membership
```

[ ]:
```
# Run the doctest
doctest.testmod()
```

Clusters Visualizations Run the following cell to take a look at the K-Means cluster.

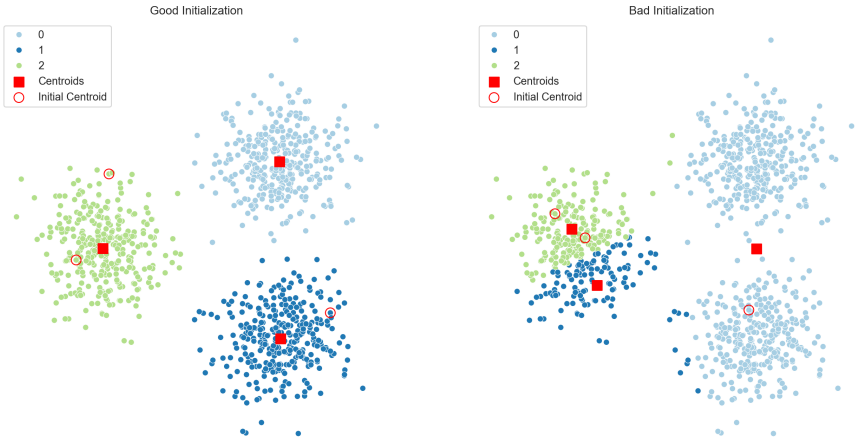
(Not Graded, just FYI:) Run the following cell several times. What did you see? Are the clusters' shape consistent between each run? Are the color of the clusters consistent between each run? Are the labels consistent between each run?

```
[ ]: # instantiate MyKMeans Object
km = MyKMeans(k=3)
# Get out the membership array
labels = km.fit(points)

plot_cluster(data_KMeans, km)
```

3.3 Question 2.3 Effects of Centroid Initialization

Take a look at the K-Mean cluster.



What is the differences between the clusters in the left and the clusters in the right? In what ways does the initialization of the centroids affect the clusters we got?

Points: 0.4

Type your answer here, replacing this text.

3.3.1 Question 2.4.1 Distortion Measures

Given the `points` and `centroids` of the dataset, calculate the distortion measure given by this formula, and return that value. Remember, the distortion measure is given by:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

Points: 0.4

```
[ ]: def distortion_measure(
        points: np.ndarray, centroids: np.ndarray
    ) -> float:
    """
```

```
    Given the points and the centroids, calculate the
    distortion measure this set of clusters.

    points: 2d numpy array containing the points
    centroids: 2d numpy array containing the centroids
    """
    # WRITE YOUR CODE HERE
    ...
```

[2]:
```
grader.check("241_distortion_measures")
```

[2]:
```
241_distortion_measures results:
    241_distortion_measures - 1 result:
         Test case failed
        Trying:
            assert callable(distortion_measure)
        Expecting nothing
        **********************************************************************
        Line 1, in 241_distortion_measures 0
        Failed example:
            assert callable(distortion_measure)
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.9/doctest.py", line 1336, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest 241_distortion_measures 0[0]>", line 1, in <module>
                assert callable(distortion_measure)
            NameError: name 'distortion_measure' is not defined
        Trying:
            (pts, centroids) = (np.array([[0], [0], [0]]), np.array([[0]]))
        Expecting nothing
        **********************************************************************
        Line 2, in 241_distortion_measures 0
        Failed example:
            (pts, centroids) = (np.array([[0], [0], [0]]), np.array([[0]]))
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.9/doctest.py", line 1336, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest 241_distortion_measures 0[1]>", line 1, in <module>
                (pts, centroids) = (np.array([[0], [0], [0]]), np.array([[0]]))
            NameError: name 'np' is not defined
        Trying:
            assert np.isclose(0, distortion_measure(pts, centroids))
        Expecting nothing
        **********************************************************************
        Line 3, in 241_distortion_measures 0
```

```
Failed example:
    assert np.isclose(0, distortion_measure(pts, centroids))
Exception raised:
    Traceback (most recent call last):
      File "/opt/conda/lib/python3.9/doctest.py", line 1336, in __run
        exec(compile(example.source, filename, "single",
      File "<doctest 241_distortion_measures 0[2]>", line 1, in <module>
        assert np.isclose(0, distortion_measure(pts, centroids))
    NameError: name 'np' is not defined
```

3.3.2 Question 2.4.2 Choose the Right K

In real world scenarios, we as data scientists oftentimes do not have access to the knowledge of the true k in the data generating process. Therefore, in this section, we explore how to choose the right value of the k to achieve the desired clusters.

[ ]:
```
# Fit K-Means on different values of k
costs = []
for k in range(1, 9):
    # instantiate KMeans Object
    km = MyKMeans(k=k)
    # Get out the membership array
    labels = km.fit(points)
    costs.append(distortion_measure(points, km.centroids))
```

[ ]:
```
# Visualize the Distortion Measure.
plt.plot(range(1, 9), costs)
plt.xlabel("k")
plt.ylabel("distortion measure")
plt.title("Distortion Measures of each $k$ values")
```

If your implementations is sound, you should have a plot looks like this.

Which is the right K to choose? And why?

Points: 0.2

Type your answer here, replacing this text.

3.4 Q3 K-Means with scikit-learn (1 Point)

3.4.1 Q3.1 K-means on MNIST

Let's explore doing a more realistic clustering task with K-means. MNIST is a dataset of digits scanned from zip codes of snail mail letters. This particular version is reduced size… fewer samples (<1800 vs 70k), fewer grayscale values (16 vs 255) and downscaled (8x8 pixels vs 64x64 pixels).

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
from sklearn.datasets import load_digits
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.cluster import KMeans

data_MNIST, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data_MNIST.shape, np.unique(labels).size

print(f"# digits: {n_digits}; # samples: {n_samples}; # features {n_features}")

fig = plt.figure(figsize=(6, 6))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

# heres a sample of 20 images to help you visualize just how ugly this level of␣
 ↪downscaling/sampling is
for i in range(20):
    ax = fig.add_subplot(5, 5, i + 1, xticks=[], yticks=[])
    # data is 28 x 28 pixels, grayscale
    ax.imshow(data_MNIST[i].reshape(8,8), cmap=plt.cm.binary,␣
 ↪interpolation='nearest')
    # label the image with the true target value in the lower left corner
    ax.text(0, 7, str(labels[i]))
```

In the cell below please do the following, in the following order: 1. Create a KMeans instance with 10 clusters, plus the following arguments `init="k-means++", n_init=4, random_state=777` 1. Fit the KMeans with `data_MNIST` 1. Create a pandas DataFrame called `kmm_pred` with the following two columns: - 'true': which contains the true categories of each datapoint, the variable `labels` we already loaded - 'pred': the predicted clustering of `data` from the fitted Kmeans - NOTE: you MUST use exactly the column names in single quotes above 1. Calculate the Rand score for the clustering, and store the value in a variable called `kmeans_rand`. NOTE: you need to pay attention to the order of arguments to this function, if you reverse them you may get a wrong answer. 1. Calculate the Adjusted Rand score for the clustering, and store the value in a variable called `kmeans_adj_rand`.

Use the scikit-learn docs to help you!

Points: 0.4

[ ]:
```
kmeans = ...
kmeans_pred = ...
kmeans_pred['true'] = ...
kmeans_rand = ...
kmeans_adjusted_rand = ...

print(kmeans_rand, kmeans_adjusted_rand)
```

[ ]:
```
grader.check("3_1_k_means_sklearn")
```

3.4.2 Q3.2 Interpret the result

The next two cells have visualizations that you can do to see what is going on with the predicted clusters and how they relate to the true labels and vice versa. Take a look.

[ ]:
```
axs=kmeans_pred.hist(column='pred',by='true',sharey=True, figsize=(10,10),␣
↪bins=range(11));
for ax in axs.ravel():
    ax.set_xlim((0,10))
    ax.set_xticks(range(10))
    old=ax.get_title()
    ax.set_title('true='+old)
    ax.set_xlabel('pred')
```

[ ]:
```
axs=kmeans_pred.hist(column='true',by='pred',sharey=True, figsize=(10,10),␣
↪bins=range(11));
for ax in axs.ravel():
    ax.set_xlim((0,10))
    ax.set_xticks(range(10))
    old=ax.get_title()
    ax.set_title('pred='+old)
    ax.set_xlabel('true')
```

OK, now in the next cell discuss your results above.

Please give us some insights… think deeply about the plots and empirical results in light of what you know about the clustering method, the metrics, and the visualizations. If you just copy/paste some definitions and say "yes/no" you will not be getting full points.

1. Describe what Rand score and Adjusted Rand score are. I suggest you look at the Wikipedia page as well as the scikit-learn docs. Describe the difference between the metrics. Interpret the numbers you got for Rand vs Adjusted Rand in light of what the difference is in the metrics.
2. Do you think that the clustering produced by KMeans is any good for predicting the true label? Why or why not… please include evidence from the viz and the scores above.
3. Regardless of what you said about the prediction above, do you think that the clustering can tell you anything about the data… like are there particular true labels that are MORE likely to be confused with others? What else might we learn from the clustering?

Points: 0.6

Type your answer here, replacing this text.

4 Q4 Gaussian Mixture Model (GMM) (3.5 Points)

[ ]:
```
# Helper function
def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()
```

```
    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle=angle, **kwargs))

def plot_gmm(gmm, X, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='Set3')
    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()

    if gmm.covariance_type=='diag':
        fullc = np.array([ np.diag(x) for x in gmm.covariances_])
    elif gmm.covariance_type=='spherical':
        fullc = np.array([ np.diag([x, x]) for x in gmm.covariances_])
    elif gmm.covariance_type=='tied':
        fullc = np.array( gmm.n_components*[gmm.covariances_])
    elif gmm.covariance_type=='full':
        fullc = gmm.covariances_
    else:
        raise NotImplemented

    for pos, covar, w in zip(gmm.means_, fullc, gmm.weights_):
        alph = np.max([w*w_factor, 0.05])
        draw_ellipse(pos, covar, ax=ax, alpha=alph)
```

New Dataset In the following problems, you will implement you own version of Gaussian Mixture Model. But first, you will use the K-means you implemented in Q1 on a new dataset generated in the following cell. The new data are stored in the variable `X_gmm`.

[ ]:
```
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X, y_gmm = make_blobs(
    n_samples=[150, 100, 200], cluster_std=[1, 0.6, 1.5],␣
 ↪centers=[[-1,-1],[6,4],[12,12]], random_state=42
)
X_gmm = np.dot(X, transformation)  # Anisotropic blobs
data = pd.DataFrame(X_gmm,columns=['x','y'])
```

```
plot_cluster(data)
```

4.1 Q4.1 Apply the K-means you implemented

Use the K-means class you implemented in the Q1 to model the new dataset stored in `X_gmm`.

Points: 0.3

[ ]:
```
data = pd.DataFrame(X_gmm,columns=['x','y'])

# instantiate KMeans Object
km = ...
# Get out the membership array
labels = ...

plot_cluster(data, km)
```

[ ]:
```
grader.check("41_k_means_new_data")
```

4.2 Q4.2 Interpret the results of K-means from Q3.1

According to the result of Q3.1, does K-means fit well to this new dataset? Why or why not? Explain.

Points: 0.6

Type your answer here, replacing this text.

4.2.1 Gaussian Mixture Model

Gaussian Mixture Models (GMMs) are probabilistic models that assume data is generated from a mixture of K Gaussian distributions, each with its own parameters: mean(μ_k) and covariance(Σ_k). They are commonly used in clustering and density estimation, as they can flexibly capture the presence of multiple subpopulations within a dataset.

$$p(\mathrm{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathrm{x}|\mu_k, \Sigma_k)$$

where π_k(mixing coefficients) can be interpreted as proportion of data in clutser k and

$$\sum_{k=1}^{K} \pi_k = 1$$

Another quantity that will play an important role is the *responsibility* conditoinal probability of z_k given x_n, denoted as:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathrm{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{\mathrm{K}} \pi_j \mathcal{N}(\mathrm{x}_n|\mu_j, \Sigma_j)}$$

And γz_{nk} can be viewed as the responsibility that cluster k takes for explaning the obsevation x_n

Algorithm Pseudocode Given a Gaussian Mixture Model, the goal is to maximize the log likelihood function with respect to the parameters(comprising the means(μ_k), covariances(Σ_k) of the clusters, and the mixing coefficients(π_k))

$$\arg\max_{\pi,\mu,\Sigma} \ln p(\mathrm{X}|\pi,\mu,\Sigma) = \sum_{n=1}^{N} \ln\left\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathrm{x}|\mu_k,\Sigma_k)\right\}$$

1. Initialize parameters μ, Σ and π
2. Repeat for some number of iterations or until converged:
    1. Keep the parameters fixed, evaluate the responsibilities γ. **(E-Step)**
    2. Keep the responsibilities(γ) fixed, update the parameters π, μ, Σ **(M-Step)**

You are provided with the skeleton codes for Gaussian Mixture Model and you will implement the E-step and M-step.

[ ]:
```
class GMModel():

    def __init__(self, X, k, max_iters):
        """
        This function initializes our parameters (mu, pi and sigma) and plots our data points.
        """
        self.X = X # Data
        self.k = k # Number of cluster/latents
        self.max_iters = max_iters
        self.dim = self.X.shape[1] # Equals 2, as we are considering 2D points
        self.N = self.X.shape[0] # Equals the number of points in the dataset
        """
        Here we initialize mu, pi, sigma and gamma.
        """
        self.mu = np.copy(X[np.random.choice(self.N, self.k, False), :]) # Shape: k x dim
        self.pi =  np.ones(self.k)/self.k # Shape: k
        self.sigma = np.array([5.0 * np.identity(self.dim) for i in range(self.k)]) # Shape: k x dim x dim
        self.gamma = np.zeros((self.N, self.k)) # Shape: N x k (used in E-step and M-step)
        """
        The following part is used for plotting
        """
        x,y = np.meshgrid(np.sort(self.X[:,0]),np.sort(self.X[:,1]))
        self.XY = np.array([x.flatten(),y.flatten()]).T

    def plot_gmm_data(self, title, colors=None):
        """
        This function creates a scatter plot of all the data points. It also creates a contour plot of the probability
        distributions of each of the clusters (specified by mu, pi and sigma)
        """
```

```
        fig = plt.figure(figsize=(6,6))
        ax0 = fig.add_subplot(111)
        # ax0.scatter(self.X[:,0], self.X[:,1], c=colors)
        data = pd.DataFrame(self.X,columns=['x','y'])
        sns.scatterplot(data=data,x='x',y='y',ax=ax0, hue=colors)
        ax0.set_title(title)
        ax0.axis('equal')
        ax0.axis(xmin=-4,xmax=8)
        ax0.axis(ymin=-4,ymax=8)
        for m,c in zip(self.mu,self.sigma):
            # c += self.sigma_correction
            multi_normal = multivariate_normal(mean=m,cov=c)
            # ax0.contour(np.sort(self.X[:,0]),np.sort(self.X[:
↪,1]),multi_normal.pdf(self.XY).reshape(len(self.X),len(self.
↪X)),colors='black',alpha=0.3)
            draw_ellipse(m, c, ax=None, alpha=0.08)
            ax0.scatter(m[0],m[1],c='r',zorder=10,s=30, marker=",")
        # ax0.legend()
        # return fig

    def fit(self, graph=True, disable=False):
        if graph:
            self.plot_gmm_data('Initial State')
        for i in tqdm(range(self.max_iters), disable=disable):
            self.expectation()
            self.maximization()
        if graph:
            color_labels = np.argmax(self.gamma, axis=1)
            self.plot_gmm_data('Final State', color_labels)
            plt.legend()

        return self.mu, self.sigma
```

4.3 Q4.3 E-step of GMM?

What is the formula of the E-step of GMM?

$\gamma(z_{nk}) = ...$

Points: 0.4

4.4 Q4.4 Implement the E-step of GMM

Complete the `expectation` function of GMM based on the formula from Q3.3.

Points: 0.6

[ ]:
```
def expectation(self):
    """
```

```
    Perform the E-Step of GMM. Update the responsibility based on the␣
 ↪parameters.
    """

    ...

    return self.gamma

# assign this method to GMModel module
GMModel.expectation = expectation
```

[ ]:
```
grader.check("44_gmm_e_step_implementation")
```

4.5 Q4.5 What is the formula of M-step of GMM?

Write down the formula for the M-step of GMM that updates the parameters of GMM.

Points: 0.5

4.6 Q4.6 Implement M-step of GMM

Complete the `maximization` function based on the formula from Q3.5.

Points: 0.6

[ ]:
```
def maximization(self):

    ...

    # Update the sigma parameter
    self.sigma = np.zeros((self.k, self.dim, self.dim))
    for j in range(self.k):
        for i in range(self.N):
            c = self.X[i] - self.mu[j]
            self.sigma[j] += self.gamma[i,j] * np.outer(c,c)
        self.sigma[j] /= N_k[j]

GMModel.maximization = maximization
```

[ ]:
```
grader.check("46_gmm_m_step_implementation")
```

4.7 Q4.7 Interpret the result

Comment on the result of GMM. How is it different from that of K-means you get from Q3.1

Points: 0.5

[ ]:
```
k = 3
max_iterations = 1000
```

```
em = GMModel(X_gmm, k, max_iterations)
em.mu = np.array([X_gmm[10], X_gmm[100], X_gmm[300]]) # set a fixed starting␣
↪point
mu, sigma = em.fit()
```

5 Q5: Gaussian Mixture Model with Scikit-Learn (1 Point)

Let's try to use the same MNIST dataset from before but use a Gaussian Mixture Model.

You are responsible for importing and using the correct class from scikit-learn.

5.1 Q5.1 GMM on MNIST

In the cell below please do the following, in the following order: - Create a Mixture of Gaussians with 10 components plus the following arguments: `random_state=777, covariance_type='tied'` 1. Fit the Mixture Model with `data` 1. Create a pandas DataFrame called `gmm_pred` with the following two columns: - 'true': which contains the true categories of each datapoint, the variable `labels` we already loaded - 'pred': the predicted clustering of `data` from the fitted mixture model - NOTE: you MUST use exactly the column names in single quotes above 1. Calculate the Rand score for the clustering, and store the value in a variable called `gmm_rand`. NOTE: you need to pay attention to the order of arguments to this function, if you reverse them you may get a wrong answer. 1. Calculate the Adjusted Rand score for the clustering, and store the value in a variable called `gmm_adj_rand`. NOTE: you need to pay attention to the order of arguments to this function, if you reverse them you may get a wrong answer.

Points: 0.4

[ ]:
```
data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape, np.unique(labels).size
```

[ ]:
```
from sklearn.mixture import GaussianMixture

gmm = ...
gmm_pred = ...
gmm_pred['true'] = ...
gmm_rand = ...
gmm_adjusted_rand = ...
print(gmm_rand, gmm_adjusted_rand)
```

[ ]:
```
grader.check("51_gmm_mnist")
```

5.2 Q5.2 Interpret the GMM results

The next two cells have visualizations that you can do to see what is going on with the predicted clusters and how they relate to the true labels and vice versa. Take a look.

[ ]:
```

```

```
axs=gmm_pred.hist(column='pred',by='true',sharey=True, figsize=(10,10),␣
 ↪bins=range(11));
for ax in axs.ravel():
    ax.set_xlim((0,10))
    ax.set_xticks(range(10))
    old=ax.get_title()
    ax.set_title('true='+old)
    ax.set_xlabel('pred')
```

[ ]:
```
axs=gmm_pred.hist(column='true',by='pred',sharey=True, figsize=(10,10),␣
 ↪bins=range(11));
for ax in axs.ravel():
    ax.set_xlim((0,10))
    ax.set_xticks(range(10))
    old=ax.get_title()
    ax.set_title('pred='+old)
    ax.set_xlabel('true')
```

OK, now in the next cell discuss your results above.

Please give us some insights… think deeply about the plots and empirical results in light of what you know about the clustering method, the metrics, and the visualizations. If you just copy/paste some definitions and say "yes/no" you will not be getting full points.

1. Do you think that the clustering produced by the Mixture Model is any good for predicting the true label? Why or why not… please include evidence from the viz and the scores above. Compare the clustering produced by the Mixture Model with the one produced by the KMeans.
2. Given the covariance argument passed to the Mixture Model, and the fact that the Mixture Model produced better results, what does that imply about the strucuture of the problem?

Points: 0.6

Type your answer here, replacing this text.

5.3 The End of A2

Have a look back over your answers, and also make sure to `Restart & Run All` from the kernel menu to double check that everything is working properly. This restarts everything and runs your code from top to bottom.

Once you're happy with your work, click the disk icon to save, and submit the zip file onto gradescope. **You MUST submit all the required component to receive credit.**

Note that you can submit at any time, but **we grade your most recent submission**. This means that **if you submit an updated notebook after the submission deadline, it will be marked as late**.

Note: If you encountered `LatexFailed` message during exporting, or has the `AttributeError: module 'nbconvert' has no attribute 'pdf'` error, it indicates that your LaTeX code is not

correct. Try to the LaTeX syntax error by scrolling up to see the LaTeX error message. If you need any additional helps, please make a private post on campuswire and we are happy to help.

5.4 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit.

Please make sure to see the output of the gradescope autograder. You are responsible for waiting and ensuring that the autograder is executing normally for your submission. Please create a campuswire post if you see errors in autograder execution.

[ ]:
```
grader.export(force_save=True, run_tests=True, files=['imgs'])
```