DSC 190 - Homework 03

Due: Wednesday, April 20

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 PM.

Problem 1.

This problem uses the data set hosted at:

`https://f000.backblazeb2.com/file/dsc-data/unknown_number_of_clusters.npy`

This data set is in standard numpy format – you can use `np.load` to load it into a numpy array. Each row in the array is a data point in 10-dimensional space.

It turns out that this data set does have cluster structure, but we do not know how many clusters it contains.

a) Create a function named `kmeans_cost(k)` which takes in a number of clusters, k, runs Lloyd's algorithm to cluster the data stored in the variable data, and returns the value of the k-means objective (cost) function resulting from the clustering.

Run `kmeans_cost(5)` and report the output. Include your code.

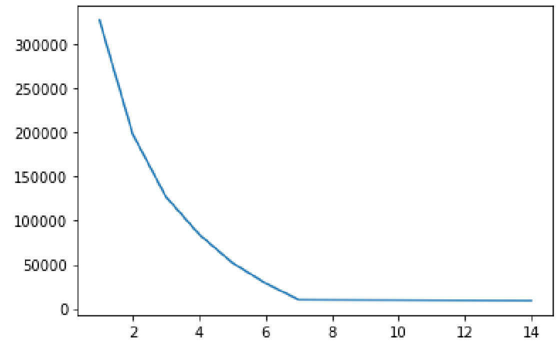
Hint: You can use sklearn and its submodules (which you'll need to import). The documentation will be helpful! Feel free to use whatever functions, methods, attributes, etc. that are provided by sklearn. It is possible to solve this in one line of code.

Solution:

```
def kmeans_cost(k):
    import sklearn.cluster
    return sklearn.cluster.KMeans(n_clusters=k).fit(data).inertia_
```

b) Using your function, plot the k-means cost for $k = \{1, 2, \ldots, 15\}$.

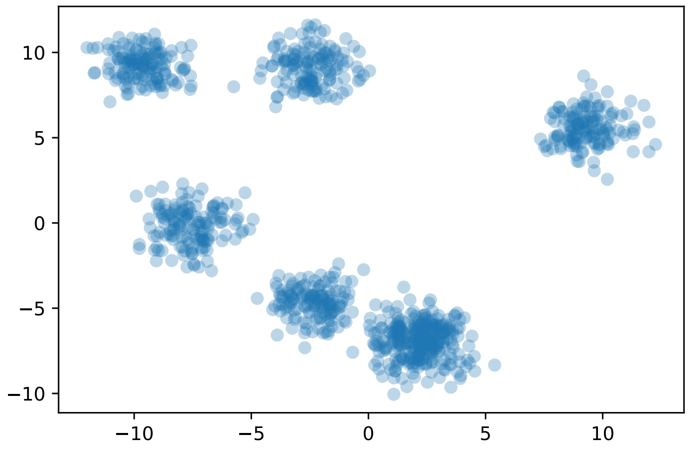
Solution:



c) Using your plot, how many clusters would you guess are in the data set?

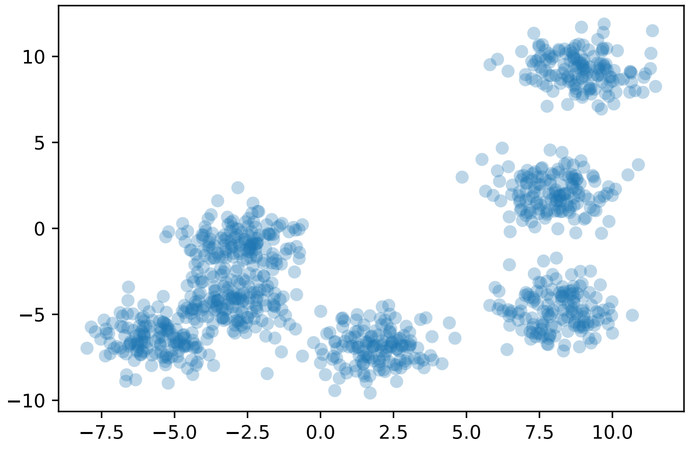
Solution: It looks like there are seven clusters.

You may have plotted the data in order to check your answer. The data is high-dimensional, but we can plot only the first two coordinates for example. If we do, we see this:



It looks like there are only 6 clusters – what gives?

This is an example of how projecting to only two dimensions can be misleading. If you choose a different pair of columns to visualize, you'll see something different. Here is with the third and fourth columns:



It looks like there might be seven clusters here, though that's certainly up to your interpretation.

Problem 2.

The goal of training any predictive model is to make good predictions on future, *unseen* data; that is, to *generalize*. The generalization performance of a Gaussian RBF network is influenced by several factors: the number of basis functions, their locations, and their widths. These factors must be decided by the data scientist who trains the model (a.k.a., you).

As discussed in DSC 80, a good choice of these "hyperparameters" can be determined through some sort of validation scheme: reserve some training data for a validation set, and pick the "hyperparameters" to maximize the accuracy on the validation set. There are several ways of doing this in practice, including k-fold cross-validation.

In this problem, you will train a Gaussian RBF Network with centers chosen via k-means as your model. You will need to choose all of the parameters of your model, including the number of Gaussian basis functions, as well as their locations and widths. We'll judge your model and assign credit based on its generalization performance. You do not necessarily need to run any particular cross-validation algorithm (e.g., k-fold cross-validation), but you should perform some type of validation to ensure that your model isn't over-fitting. You may use any language and package that you'd like (for example, `sklearn`).

This problem uses the data set at `https://f000.backblazeb2.com/file/dsc-data/kmeans-centers.csv`

This file is a CSV with 1400 rows and 5 columns. The first four columns are features, and the last column is the label (either +1 or -1).

Using your trained model, predict the label of each of these five points:

- $\vec{x}^{(1)} = (7, -94, 19, -78)^T$
- $\vec{x}^{(2)} = (71, 73, 16, 101)^T$
- $\vec{x}^{(3)} = (-52, 6, -82, 78)^T$
- $\vec{x}^{(4)} = (70, -86, -36, -86)^T$
- $\vec{x}^{(5)} = (-64, -67, -96, 78)^T$

We know the right label for these five points, and will test your model by comparing its predictions to the right labels. A good model will be able to get all five predictions exactly right.

In addition to these predictions, report:

- The number of clusters, k, used.
- The Gaussian width parameter, σ, used by your model.

Be sure to attach your code.

Hint: you can assume that the data points are nicely separated, just not by a linear decision boundary. In other words, you should be able to train an RBF network that achieves close to perfect predictions on a validation set without needing to do anything too sophisticated to pick the hyperparameters. For example, we were able to get 100% accuracy on a validation set of a few hundred points by picking the hyperparameters manually via "guess and check".

Solution:

The correct predictions are 1, -1, -1, -1, and 1.

We achieved this using $k = 100$ and a $\sigma = 10$.

```python
def augment(X):
    return np.column_stack((
        np.ones(len(X)),
        X
    ))

w = np.linalg.lstsq(augment(X_train), y_train)[0]
(np.sign(augment(X_train) @ w) == y_train).mean()

np.sign(augment(X_test) @ w) == y_test

kmeans = sklearn.cluster.KMeans(100)
kmeans.fit(X_train)

def make_phi(mu, sigma):
    def phi(X):
        return np.exp(-np.linalg.norm(X - mu, axis=1) / sigma**2)
    return phi

phis = [make_phi(mu, 10) for mu in kmeans.cluster_centers_]

X_train_phi = np.column_stack([phi(X_train) for phi in phis])
X_test_phi = np.column_stack([phi(X_test) for phi in phis])

w = np.linalg.lstsq(augment(X_train_phi), y_train)[0]

(y_train == np.sign(augment(X_train_phi) @ w)).mean()
(y_test == np.sign(augment(X_test_phi) @ w)).mean()\


def predict(X):
    """
    Accept an n x 4 data matrix and apply your trained RBF network to
    return an array of predictions (either 1 or -1), one for each row of X
    """
    X_phi = np.column_stack([phi(X) for phi in phis])
    return np.sign(augment(X_phi) @ w)
```