3. (60 points, with 20 bonus points) This problem will combine two different ideas: Bayesian networks and unsupervised clustering. We will build a classifier that first converts continuous features into discrete classes by a process called vector quantization, using an algorithm known as k-means clustering. The entire exercise is meant to be done programmatically. We have provided you the code for the k-means algorithm and Step 1. If you choose to use a programming language other than Python, you are allowed to use an off-the-shelf implementation of k-means.

   Given: Consider a two dimensional real valued data point $R^{(i)} = (x_1^{(i)}, x_2^{(i)})$. We have a training set of labeled data as Train= $\left\{\left(R^{(i)}, C^{(i)}\right)\right\}_{i=1}^{n}$ where $C^{(i)} \in \{1, \ldots, L\}$ is a discrete class label for point $R^{(i)}$. You are also given a set of testing data with similar labeling.

   Procedure:

   Step 1: (already done; 0 points) First, we want to find k vectors that represent the set Train well. To do so we use a clustering algorithm to find k clusters, and then we pick the mean of each cluster as the

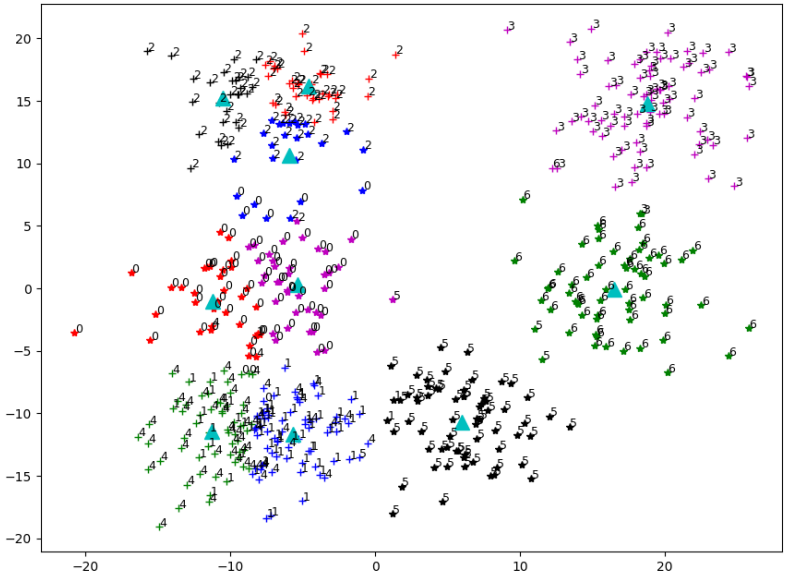

Figure 2: Example run of k-means on training data, where: the k-means vectors are shown as large cyan triangles; the number next to each point is the training label; and the color and shape of the point indicate the closest of the k-means vectors.

representative of members of that cluster. The code of the k-means algorithm is given to you in `kmeans.py`. You should run it on Train and make sure you understand it. A plot showing an example run is shown in Figure 2.

The k-means algorithm is relatively simple:

- Initialization: choose n vectors $\mathcal{V} = \{V_1, \ldots, V_n\}$ randomly.
- While not converged:
  - For each datapoint $R_i = (x_1^{(i)}, x_2^{(i)})$ find the closest vector V_j and assign R_i to V_j.
  - Let $V_j \leftarrow$ average of all points assigned to V_j
  - If no points are assigned to V_j (or fewer than some threshold) randomly reinitialize V_j

Take the training points, ignoring the current labels, and perform k-means clustering with $k = 10$ means.

Step 2: (10 points) Once you have a set of vectors $\mathcal{V}$, you can determine the closest vector V_j for each training data point R_i. From vectors representing the training set, compute a table $P(V_j|C = c)$ and $P(C = c)$ by counting.

Step 3: (20 points) On the test set, determine the class of each point by using $P(C|V) = \alpha P(V|C)P(C)$. (Note that I want you to do it this way rather than directly computing $P(C|V)$ because we will be using this idea later on in a future exercise.) Report the average and standard deviation of the classification error rate over ten different runs of the k-means algorithm. Submit the working code.

Step 4: (20 points) Sample the average/standard deviation of the classification error rate for $k = 2, 5, 6, 8, 12, 15, 20, 50$ (and other values if you wish). Qualitatively discuss the results of this experiment (1 paragraph).

Step 5: (10 points) The data were generated by sampling a set of Gaussians with random centers (see `generate_data.py`). Modify the data generator script (or write your own) to increase the variance of the Gaussian distribution that generates the data (so that they overlap more). Discuss how the results change as you increase variance. Use k=10 and average results over 10 runs of the data generator script for each selected variance.

For bonus questions:

(a) Bonus 1 (10 points): Modify the data generator script (or write your own) so that the Gaussians are asymmetric — i.e., the Gaussian's variances are different in different dimensions. Perform the experiment again. Thinking about the Euclidean distance metric, describe how the asymmetry affects the k-means process.

(b) Bonus 2 (10 points): Modify the data generator script (or write your own) to generate higher dimensional data (3d, 4d, 5d). How do the results change as a function of dimensionality?