Imperial College London

Department of Mathematics
MATH60026 - MATH70026
Methods for Data Science
Years 3/4/5

Coursework 2 - Part 2

Task 3: k-means clustering (20 marks)

3.1 (8 marks) - Using the k-means code given in the Week 7 Notebook, cluster the embeddings given in `star_embeddings_train` using k-means with: k = 4; convergence threshold = 0; random seed = 0; maximum number of iterations = 100. Take the best clustering among 10 different initialisations. Plot the embeddings in the space of their first two coordinates colouring each data point according to their cluster.

[cell for code]

(Continue 3.1)

Now you have two partitions of the star embeddings:

- Partition 1 - Partition into k-means clusters: the partition of the star embeddings into four clusters you have obtained with k-means (k=4);
- Partition 2 - Partition into true classes: the partition of the star embeddings into the four ground truth star classes given in `star_classes_train`.

Plot the embeddings in the space of their first two coordinates colouring each data point according to their class.

Compute the Normalised Variation of Information (NVI) between Partition 1 and Partition 2 (see Week 9 Notebook).

[cell for code]

3.2 (7 marks) - Consider the `compute_within_distance` function in the Week 7 Notebook on k-means, which we include here for your reference:

```
def compute_within_distance(centroids, X, labels):
    """
    Compute the within-cluster distance.
    Args:
        centroids (np.ndarray): the centroids array, with shape (k, p).
        X (np.ndarray): the samples array, with shape (N, p).
        labels (np.ndarray): the cluster index of each sample, with shape (N,).
    Returns:
        (float): the within-cluster distance.
    """
    within_distance = 0.0
```

```
    k, p = centroids.shape
    for l in range(len(centroids)):
        centroid = centroids[l]
        # Applying aggregate computations on `NaN` values
        # can propagate the `NaN` to the results.
        # In this case we skip the `NaN` centroid,
        # which is effectively an empty cluster.
        if np.isnan(centroid).any():
            continue
        # Select samples belonging to label=l.
        X_cluster = X[labels == l]
        # You need to add the `X_cluster` contribution to `within_distance`
        # 1. Compute the cluster contribution.
        cluster_se = (X_cluster - centroid)**2
        assert cluster_se.shape == (len(X_cluster), p)
        # 2. Accumulate
        within_distance += np.sum(cluster_se)
    return within_distance
```

(Continue 3.2)

Modify this function appropriately to produce a new function called `compute_within_distance_per_group` that takes as input the data points (`X`), their group labels (`labels`) and the centroid of each group (`centroids`) and outputs the within-distance for *each* of the groups of data points.

Use your function `compute_within_distance_per_group` on the Partition 1 and Partition 2 to obtain the within-distances for the clusters (Partition 1) and the within-distances for the classes (Partition 2). Produce a bar plot comparing: the four within-distances for the clusters in Partition 1, and the four within-distances for the classes in Partition 2.

[cell for code]

3.3 (5 marks) - Using all the computations and plots you have produced in Task 3, draw conclusions on the comparison between the two partitions, on the structure of the embeddings, and on the performance of k-means. Discuss your results.

[cell for text]

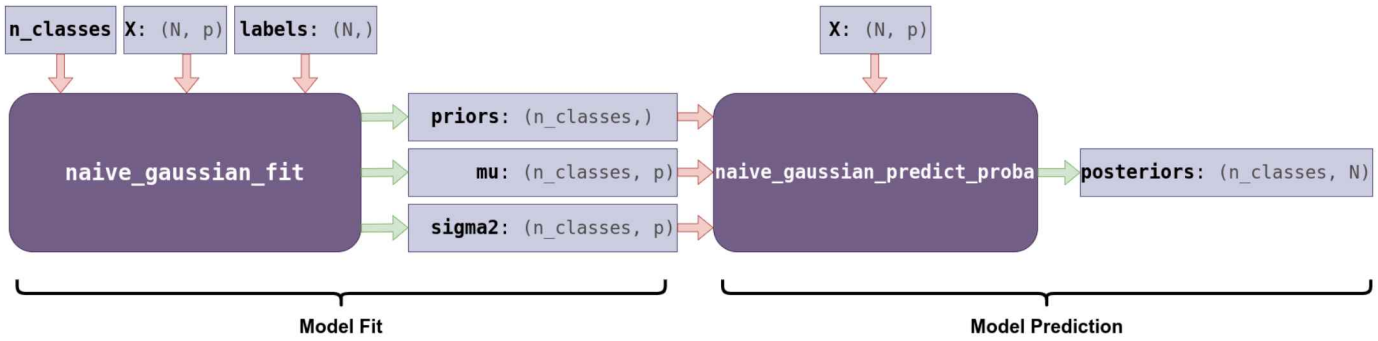
Task 4: Gaussian Naive Bayes classifier (15 marks)

4.1 Gaussian Naive Bayes classifier for the star image embeddings (10 marks) - In this task, you will fit a Gaussian Naive Bayes (GNB) classifier to classify the embeddings given in `star_embeddings_train` into the four star classes. In the GNB classifier, the features are assumed to be independent and the likelihood is given by a Gaussian distribution. Hence, for a data feature j, its likelihood of belonging to class q is given by:

$$P(X_j = x_j | Y = y_q) = \frac{1}{\sqrt{2\pi\sigma_{j,q}^2}} e^{\frac{-(x_j - \mu_{j,q})^2}{2\sigma_{j,q}^2}}, \quad j = 1, \ldots, p$$

where $\mu_{j,q}$ and $\sigma_{j,q}^2$ are, respectively, the mean and the variance of feature j within class q (see Chapter 5 in the Lecture Notes).

The following scheme gives an overview of the implementation:



In this task, you have skeleton code for different functions. In case you do not know how to correctly fill in the lines, note that you can still run the cells, obtaining a random classifier.

4.1.1 - Model fit (3 marks) - You first need to estimate means $\mu_{j,q}$ and variances $\sigma_{j,q}^2$ for each class q, which you will store in the variables `mu` and `sigma2`. Fill the missing lines of the following skeleton code to obtain the function `compute_mean_variance` that will do this:

```
def compute_mean_variance(n_classes, X, labels):
    """
    Compute the mean and the variance of samples with respect to each class.
    Args:
        n_classes (int): total number of classes.
        X (np.ndarray): data points, with shape (N, p)
        labels (np.ndarray): class labels for each sample in X, with shape (N,),
            while each label value belongs to {0, 1, ..., n_classes - 1}.
    Returns:
        mu: the mean of the classes, with shape (n_classes, p).
        sigma2: the variance of the classes, with shape (n_classes, p).
    """
    N, p = X.shape
    mu = np.zeros((n_classes, p))
    sigma2 = np.ones((n_classes, p))

    for label in range(n_classes):
        X_c = 0 # <-- EDIT THIS LINE
        mu[label] = 0 # <-- EDIT THIS LINE
        sigma2[label] = 1 # <-- EDIT THIS LINE

    return mu, sigma2
```

(Continue 4.1.1)

Use the function `compute_mean_variance` to obtain means and variances for each class, and use them to fit a Gaussian Naive Bayes classifier with a uniform prior over the classes from the training data (`star_embeddings_train, star_classes_train`) by completing the following skeleton code:

```
def naive_gaussian_fit(n_classes, X, y):
    """Use training data to fit the Gaussian Naive Bayes classifier parameters.
    Args:
        n_classes (int): total number of classes.
        X (np.array): The samples array, shape: (N, p).
        y (np.array): Categorical target array, shape: (N, ). Each category value
            belongs to {0, 1, ..., n_classes - 1}.
    Returns:
        prior (np.array): Prior distribution of classes, shape: (n_classes, ).
        mu (np.ndarray): the mean for each class, shape: (n_classes, p)
        sigma2 (np.ndarray): the variance for each class, shape: (n_classes, p)
    """
    p = X.shape[1]
    # define prior
    prior = np.zeros(n_classes) + 1e-2 # <-- EDIT THIS LINE
    mu, sigma2 = compute_mean_variance(n_classes, X, y)
    return prior, mu, sigma2
```

(Continue 4.1.1)

Use the function matplotlib.pyplot.errorbar(*x, y, yerr=None, xerr=None,...*) to plot the first two coordinates of the mean $\mu_{j,q}$ (to pass as the arguments *x, y*) and of the standard deviation $\sigma_{j,q}$ (to pass as the arguments *xerr, yerr*) for each class q. Plot means and standard deviations of all four classes in the same figure.

[cell for code]

4.1.2 Model Prediction & Evaluation (7 marks) - Next, build a function `naive_gaussian_predict_proba` to evaluate the prediction of the GNB classifier by filling in the missing lines of the following skeleton code. For numerical convenience, the function uses the Gaussian log-likelihood: to compute it, you need to write another function, `naive_gaussian_log_likelihood`, for which we also provide a template to fill in.

```
def naive_gaussian_log_likelihood(X, mu_i, sigma2_i):
    """
    Compute the Gaussian log-likelihood for each multidimensional sample in X,
    with the naive assumption of independence across features (dimensions).
    Args:
        X (np.ndarray): the data array, shape: (N, p).
        mu_i (np.ndarray): the class mean, shape: (p, )
        sigma2_i (np.ndarray): the class variance, shape: (p, )
    Returns:
        (np.ndarray): the log-likelihood for each sample, shape: (N, ).
    """
    return np.zeros(len(X)) # <-- EDIT THIS LINE
```

```
def naive_gaussian_predict_proba(prior, mu, sigma2, X):
    """Predict the posterior probabilities of the classes with the Gaussian Naive Bayes.
    Args:
        prior (np.array): Prior distribution of classes, shape: (n_classes, ).
        mu (np.ndarray): the mean for each class, shape: (n_classes, p)
        sigma2 (np.ndarray): the variance for each class, shape: (n_classes, p)
```

```
    X (np.array): The samples array, shape: (N, p).
Returns:
    posteriors (np.array): Posterior distribution of samples, shape: (n_classes, N).
"""
n_classes, N = len(prior), len(X)
# define likelihood P(x|y) shape: (n_classes, N)
log_lk = np.vstack([naive_gaussian_log_likelihood(X, mu_i, sigma2_i) for (mu_i, sigma2_i) in zip(mu, sigma2)])
# compute log-posterior
log_posterior = np.zeros((n_classes, N)) # <-- EDIT THIS LINE

# to ensure numerical stability when computing the normalisation factor, we can subtract the maximum log posterior
# this change will be re-absorbed into the normalisation factor without loss of generality
log_posterior = log_posterior - log_posterior.max(axis=0)

# normalise to get full posterior distribution
normalize_term = np.ones((1, N)) # <-- EDIT THIS LINE
posteriors = np.ones((n_classes, N)) / n_classes # <-- EDIT THIS LINE
return posteriors
```

(Continue 4.1.2)

Evaluate the function `naive_gaussian_log_likelihood` on the data points of Class 3 and produce a histogram of these values to visualise the Gaussian log-likelihood distribution.

Next, evaluate the function `naive_gaussian_predict_proba` on the test set of star embeddings (`star_embeddings_test`, `star_classes_test`), and use these predictions for a hard-classification assignment of classes. Compute and report the resulting classification accuracy.

[cell for code]

4.2 GNB classifier on a subset of coordinates (5 marks, 3rd-year students only) - Repeat the fit and evaluation of the GNB classifier using only two coordinates of the star embeddings given by the Python indices 1 and 141. Compute the accuracy on the test set and discuss its performance relative to that of the GNB classifier on the full embeddings (tasks **4.1.1 - 4.1.2**).

[space for code]

4.2 GNB classifier of the raw images (5 marks, MSc/4th-year students only) - Repeat the fit and evaluation of the GNB classifier on the raw images (`star_images_train` for fitting the classifier, `star_images_test` for evaluating its predictions). Compute the accuracy on the test set and discuss its performance relative to that of the GNB classifier on the full embeddings (tasks **4.1.1 - 4.1.2**).

[space for code]

Task 5: Soft-margin linear Support Vector Machine (SVM) classifier (15 marks)

5.1 (10 marks) - Use the SVM code in the Week 4 Notebook to perform two *binary* classifications, where a class is classified against the rest. Use the training set to train two soft-margin linear SVMs to classify the embeddings into class α vs. the rest (not α), where α=1,3. Train the SVMs using only the coordinates of the embeddings with Python index 86 and 136, setting the regularisation strength to 100, the maximum number of iterations to 2000, the learning rate to `1e-6`, and the stopping criterion to `1e-3`. In both cases, print the loss for several training iterations to show convergence and compute the classification accuracy on the embeddings of the test set (once again, for coordinates 86 and 136 only). Discuss your results and any differences between the two cases (α=1,3).

[space for code]

5.2 SVM for the raw images (5 marks, 3rd-year students only) - Repeat the two binary classification tasks in **5.1**, but now on the raw images (`star_images_train` to train the classifier, and `star_images_test` for evaluation). Compute the accuracy on the test set, and compare it to the results obtained in **5.1**. Discuss your findings using the fact that the embeddings were obtained from the output of one of the layers of a Neural Network.

[space for code]

5.2 Drawing the SVM decision boundary and margin (5 marks, MSc/4th-year students only) - For each of the two binary classification tasks in **5.1**, plot the coordinates 86 and 136 of the training embeddings colouring each data point according to the class (α vs the rest). For each plot, draw the decision boundary and the margin of the soft-margin linear SVM that you have trained for the corresponding classification task in **5.1**.

[space for code]