Xi'an Jiaotong-Liverpool University
西交利物浦大學

Module Code	Examiner	Department	Tel
INT104	Shengchen Li	INT	3077

2nd SEMESTER 23-24 RESIT EXAMINATION

Undergraduate

Artificial Intelligence

TIME ALLOWED: *3.5 hours*

INSTRUCTIONS TO CANDIDATES

1. This is a blended open-book exam and the duration is 3.5 hours.
2. Total marks available are 100. This accounts for 100% of the final mark.
3. Answer all questions. Relevant and clear steps should be included in the answers.
4. Please use MCQ card delivered to answer MCQ questions. Please use answer booklet for answer other questions.
5. Only English solutions are accepted.
6. The use of calculator is allowed.
7. Besides lecture notes and hand writing notes, only books (with an ISBN) are allowed. NO DICTIONARIES.

Section 1 Multiple Choice Questions

This section of the exam contains multiple-choice questions. Each question will be followed by four options A, B, C, and D. You are required to choose ONE answer that you deem to be the most appropriate.

Section 2 Computation Questions

Outlook	Humidity	Wind Speed	Preference
Rainy	80%	0.5m/s	Yes
Rainy	40%	0.2m/s	Yes
Rainy	50%	5.0m/s	No
Rainy	50%	0.2m/s	Yes
Rainy	75%	4.0m/s	No
Sunny	70%	5.0m/s	No
Sunny	75%	0.4m/s	No
Sunny	80%	0.1m/s	No
Sunny	50%	0.2m/s	Yes
Sunny	40%	4.0m/s	Yes

1. The given table shows the preference of golf players for playing golf or not. Based on the information presented in the table, what is the golf player's preference under the condition of rainy weather and humidity higher than 65%? Please use Naïve Bayes to make the decision.

(10 Marks)

Section 3 Programming Questions

2. Assume a dataset is stored in a variable `features` where each column of `features` represents a feature and each row of `features` represents a data sample. The samples belong to **4 classes**. A variable `labels` stores the class information of each sample as a column vector where each row of `labels` represents a data sample.

The Python script on the next page attempts to cluster the features with k-means in an unsupervised manner. The resulting clusters then correspond to a class according to the ground truth provided by `labels`. As a result, each resulting cluster will be assigned to a class.

Both `features` and `labels` are an `ndarray` in Numpy.

Please fill in the blank marked as `[#001]` to `[#010]` as appropriate in the script and then answer the following questions:

(a) When `n_clusters=4`, is that guaranteed that each class in the dataset can be assigned to at least one resulting cluster in the given Python script? If yes, explain the reason. If no, provide a way that assures each class being assigned to at least one resulting cluster.

(b) There are about 600 samples in the dataset. When `n_clusters=500`, the accuracy of clustering is `91.438%`. Will the result be considered as a case of overfitting? Why?

Each blank in the Python script is worth 1 mark. Each question you are asked to answer is worth 4 marks.

A set of API of Python has been provided in the section of Appendix for your reference.

```
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from scipy import stats
import numpy as np

# Range of number of clusters to try
num_clusters_range = [4, 40, 120, 200, 240, 400, 500]

for n_clusters in [#001]:
    # Initialize and fit KMeans
    kmeans = KMeans(n_clusters=[#002], random_state=0)
    kmeans.[#003](features)

    # Get the cluster labels for each data point
    cluster_labels = [#004]

    new_labels = np.empty_like([#005]) #Initialize

    # For each unique cluster label
    for cluster_label in np.unique([#006]):
        # Find the most common class label in 'labels'
        # corresponding to this cluster label
        most_common_label = \
            stats.mode(labels[[#007]])[0][0]

        # Assign this label to all data in this cluster
        new_labels[cluster_labels == cluster_label] = [#008]

        # Calculate accuracy
        accuracy = accuracy_score(labels, new_labels)
        print(f'Number of clusters: {[#009]}, \
            Accuracy: {[#010]*100:.3f}%')
```

(18 Marks)

3. Write a Python script that detects the best value of `C` in a binary SVM classification. Assume the input data is `X` whose shape is `(1000, 2)`. Please remember to perform cross validation. The candidate `C` value is `[0.1, 0.5, 0.7]`.

The appendix of the exam provides a set of API that may be used in this question.

(18 Marks)

4. Write a Python script that compares the clustering results of K-means with the following initialisation methods: 1) totally random; 2) pick random samples. The data should be randomly generated and contains 1000 samples for 2 clusters, which is named as a variable `X`. `k` hence should be set to 2. Use silhouette index to evaluate the performance of clustering.

The appendix of the exam provides a set of API that may be used in this question.

(18 Marks)

Section 4 Appendix: Edited Python API being used in this exam

The following API information may be used in this exam.

numpy.empty_like

`numpy.empty_like(prototype)`: Return a new array with the same shape and type as a given array.

Parameters

`prototype: array_like`

The shape and data-type of prototype define these same attributes of the returned array.

Returns

`out: ndarray`

Array of uninitialized (arbitrary) data with the same shape and type as *prototype*.

sklearn.cluster.KMeans

`class sklearn.cluster.KMeans(n_clusters=8, random_state=None)`: K-Means clustering.

Parameters

`n_clusters: int, default=8`

The number of clusters to form as well as the number of centroids to generate.

`random_state: int, RandomState instance or None, default=None`

Determines random number generation for centroid initialization. Use an int to make the randomness deterministic.

Attributes

`labels_ : ndarray of shape (n_samples,)`

Labels of each point.

Methods

`fit(X)`

Compute k-means clustering.

Parameters

`X: array-like, sparse matrix of shape (n_samples, n_features)`
Training instances to cluster.
<u>Returns</u>
`self: object`
Fitted estimator.

numpy.unique

`numpy.unique(ar)`: Find the unique elements of an array. Returns the sorted unique elements of an array.

Parameters
`ar: array_like`
Input array. The array will be flattened if it is not already 1-D.
Returns
`unique: ndarray`
The sorted unique values.

scipy.stats.mode

`scipy.stats.mode(a)`: Return an array of the modal (most common) value in the passed array. If there is more than one such value, only one is returned. The bin-count for the modal bins is also returned.

Parameters
`a: array_like`
Numeric, n-dimensional array of which to find mode(s).
Returns
`mode: ndarray`
Array of modal values.
`count: ndarray`
Array of counts for each mode.

sklearn.model_selection.train_test_split

`sklearn.model_selection.train_test_split(*arrays, test_size=None)`: Split arrays or matrices into random train and test subsets.

Parameters

`*arrays: sequence of indexables with same length / shape[0]`

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

`test_size: float or int, default=None`

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples.

Returns

`splitting: list, length=2 * len(arrays)`

List containing train-test split of inputs.

sklearn.datasets.make_blobs

`sklearn.datasets.make_blobs(n_samples=100, n_features=2, *, centers=None)`: Generate isotropic Gaussian blobs for clustering.

Parameters

`n_samples: int or array-like, default=100`

If int, it is the total number of points equally divided among clusters. If array-like, each element of the sequence indicates the number of samples per cluster.

`n_features: int, default=2`

The number of features for each sample.

`centers: int or array-like of shape (n_centers, n_features), default=None`

The number of centers to generate, or the fixed center locations. If n_samples is an int and centers is None, 3 centers are generated. If n_samples is array-like, centers must be either None or an array of length equal to the length of n_samples.

Returns

`X: ndarray of shape (n_samples, n_features)`

The generated samples.

`y: ndarray of shape (n_samples,)`

The integer labels for cluster membership of each sample.

sklearn.metrics.silhouette_score

`sklearn.metrics.silhouette_score(X, labels)`: Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is $(b - a)/max(a, b)$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is $2 <= n_{labels} <= n_{samples} - 1$.

This function returns the mean Silhouette Coefficient over all samples.

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

Parameters

`X: array-like of shape (n_samples_a, n_features)`

An array of pairwise distances between samples, or a feature array.

`labels: array-like of shape (n_samples,)`

Predicted labels for each sample.

Returns

`silhouette: float`

Mean Silhouette Coefficient for all samples.

sklearn.metrics.accuracy_score

`sklearn.metrics.accuracy_score(y_true, y_pred)`: Accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.