Summary Report of Assignment 1

Name: Meenakshi Vaidhiyanathan
Banner ID: 811318774

The given python code with respect to Iris dataset was replicated on a simulated dataset using the function `make_blobs` from the library `sklearn` called as scikit-learn, imported from the dataset module. Other libraries that were used for the simulated dataset include `matplotlib`, `pandas` and `numpy`.

Problem definition: The given question is represented as a 2D matrix. It can be defined as a 2D tensor of rank 2. The shape was found to be (3,2). The dimensionality of each centre in the 2D space is 2 while the number of centers are 3. It is important to note that the labels are a vector of the 1D tensor.
`n_classes` determines the class as 0,1 or 2 with respect to the centers defined as [[2, 4], [6, 6], [1,9]]. `random_state=1` ensures that the random process in the algorithm produces the same results every time the code is run, thereby ensuring reproducibility.

Generation of simulated dataset: About 150 such data points around the defined centers were generated. The dataset was drawn from a multivariate normal distribution centered about the above mentioned centers.

In tensor terms:
Shape of Centers is a (3, 2) matrix.
Shape of data is (150, 2) matrix.
Shape of labels is (150,) vector.

The simulated dataset is often used for classification or clustering algorithms. This was used to implement the k-Nearest Neighbour(kNN) algorithm in python.
This is instance-based learning also defined as a non-parametric algorithm. The classifier is from the `scikit-learn` library. It is a supervised machine learning algorithm that is used to build a classification model. The working of the algorithm involves the calculation of the distance of the new data point to k nearest data points and classifying by majority vote. It does not train the model but simply memorises the training data. Given that it is supervised learning, there are labels associated with the target attribute. If no parameters are specified, it would be called an “out-of-the-box” classifier. The said classifier is given the name `knn` which is then made to fit on the training dataset.

Splitting of the dataset: The simulated dataset was then split into a training set and testing set with each containing about 80% and 20% of the total data respectively. Most machine learning models would be built after splitting the data into a training set, testing set and a validation set, if necessary.
The training set is used to train the model while the test is made unseen or unexposed while

building the model. The validation set is used to check the various performance metrics of the model built using the training set. It is also used in hyperparameter tuning. The model is then made to predict on the test set.

Model building:
The first and second kNN model that was built, was named `knn` and ‘knn2`. While `knn` had no parameters specified explicitly as arguments in the classifier, the latter had them mentioned and passed as arguments to the kNeighbourClassifier(). Both the models were built on the training set. The need to split the data into a validation set allows the possibility for hyperparameter tuning. In this case, both the models were built on the training set containing 80% of the data and were made to predict on the test set.

Performance metric: The models built are evaluated using various metrics such as accuracy, precision, recall and specificity. With respect to this assignment, the metric used to evaluate the model is accuracy. It is the correctly predicted instances to the total number of instances.
In many machine learning algorithms, it is to be noted that accuracy alone is insufficient to evaluate model performance and other metrics are to be considered as well.

Faster calculations of distance can be carried out by passing an argument to the classifier KNeighborClassifier() and setting the parameters to the following conditions, depending on the need.

- algorithm = `auto` # would choose the most appropriate method for distance calculation.
- algorithm = `ball_tree` # This would use a ball-tree algorithm to calculate distances and is useful for high-dimensional data.

  algorithm = `kd_tree` #This would use a kd_tree algorithm to calculate distances and is useful for low-dimensional data.

- algorithm = `brute` # This would use a brute-force search.

- Metric = ‘minkowski’ is used with respect to either of:
  p= 1 # To calculate using manhattan distance.
  p= 2 # To calculate using euclidean distance.
- n_jobs = 1 # this is a default setting and no parallel processing
- n_jobs = -1 # sets parallel processing for faster computation

Results and interpretation:

- The accuracy of the `knn` model on the training set was found to be 1.0 as the predicted values of the training set were stored in `train_data_predicted`.

Accuracy is calculated as follows:

Accuracy = TP + TN / (TP + TN + FP + FN)
Where TP is true positive, TN is true negative, FP is false positive and FN is False negative.

This is calculated with respect to `train_labels` and `train_data_predicted`.

Target values of the `train_data` depicting true labels:

```
Target values of training data:
[0 0 1 0 1 1 1 1 1 0 2 0 1 1 1 1 2 1 1 1 2 2 2 2 1 0 0 0 0 2 1 1 0 1 2 2 0
 1 1 2 2 1 2 0 0 1 0 0 0 2 0 2 0 1 2 1 2 2 2 0 2 0 2 0 2 2 2 2 2 1 1 1 2 2
 0 2 1 0 1 0 0 0 2 1 0 1 0 0 1 0 0 2 1 2 2 2 0 2 2 1 2 1 1 1 0 2 2 0 0 2 2
 1 1 1 1 2 0 0 2 1]
```

Predictions of the `knn` model on the training data:

```
Predictions from the classifier:
[0 0 1 0 1 1 1 1 1 0 2 0 1 1 1 1 2 1 1 1 2 2 2 2 1 0 0 0 0 2 1 1 0 1 2 2 0
 1 1 2 2 1 2 0 0 1 0 0 0 2 0 2 0 1 2 1 2 2 2 0 2 0 2 0 2 2 2 2 2 1 1 1 2 2
 0 2 1 0 1 0 0 0 2 1 0 1 0 0 1 0 0 2 1 2 2 2 0 2 2 1 2 1 1 1 0 2 2 0 0 2 2
 1 1 1 1 2 0 0 2 1]
```

Predictions of the `knn` model on the `test_data`:

```
Target values of testing data:
[2 2 2 0 0 1 1 2 2 1 0 1 0 0 2 0 0 0 1 0 0 1 1 2 0 0 0 1 2 1]
```

Predictions of the `knn2` model on the `test_data`::

```
Target values of testing data:
[2 2 2 0 0 1 1 2 2 1 0 1 0 0 2 0 0 0 1 0 0 1 1 2 0 0 0 1 2 1]
```

Similarly, the `knn` and `knn2` models were made to predict on the testing set called `test_data` containing 20% or 30 data points.

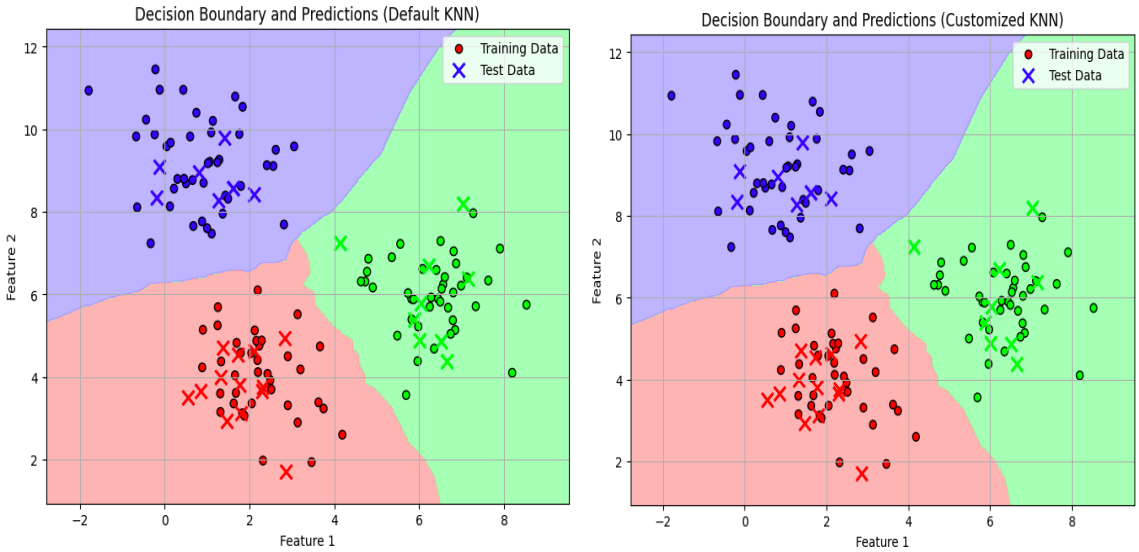
- It can be seen that the accuracy of both `knn` and `knn2` models on the test set was found to be 1.0, thereby suggesting that the predictions on the test set were correct with an accuracy of 100%.
- Both the models had similar predictions, as the `knn2` model, although it had parameters specified, the designed model was similar to the default settings of the `knn` model. This can be seen as visualised below.

Visualization of training set and testing set as seen below:

The visualizations below are depicted using the `matplotlib` library.
As seen below, the data points are represented as a scatter plot with Feature 1 on the x-axis with respect to Feature 2 on the y-axis. The circles represent the training data while the cross

marks represent the testing data. These data points are generated about the mentioned `centers`.

Visualization of model's predictions:



The above scatter plot depicts Feature 1 on the x-axis with respect to Feature 2 on the y-axis. While circles depict the training data, the cross mark is depicted as the predictions of the model on the testing data containing 20% of the dataset. The three classes are represented with three colours depicting their respective decision boundary as seen above. It is interesting to note that a similar plot is generated for the second model labeled `Customized KNN`, as the parameters are set and passed as arguments explicitly into the classifier with values such as algorithm= 'auto', leaf_size=30, weights=`uniform`, metric_params=None, n-neighbours=5, n_jobs=1, p=2, metric=`minkowski`. It is evident that the model has made predictions on the test set with an accuracy of 1.0 for both `knn` and `knn2` model.

References:

1. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). **Scikit-learn: Machine learning in Python**. *Journal of Machine Learning Research, 12*, 2825–2830. https://jmlr.org/papers/v12/pedregosa11a.html

2. Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

3. Cover, T., & Hart, P. (1967). **Nearest neighbor pattern classification**. *IEEE Transactions on Information Theory, 13*(1), 21–27. https://doi.org/10.1109/TIT.1967.1053964

4. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7

5. Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.