APML Assignment 2

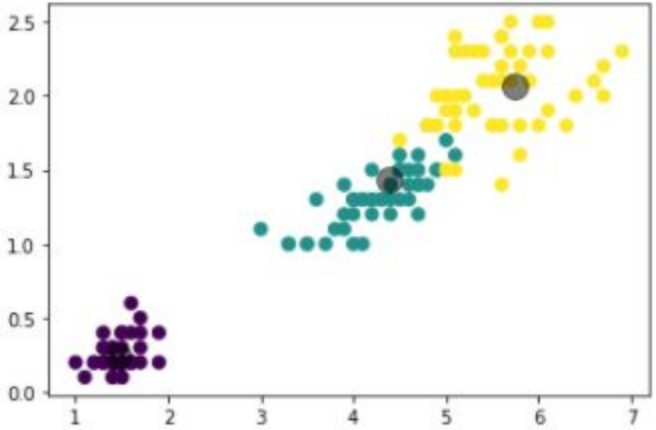
Tan Siew Ling

Part A - Unsupervised Learning (Iris dataset)

Part A - Data Exploration

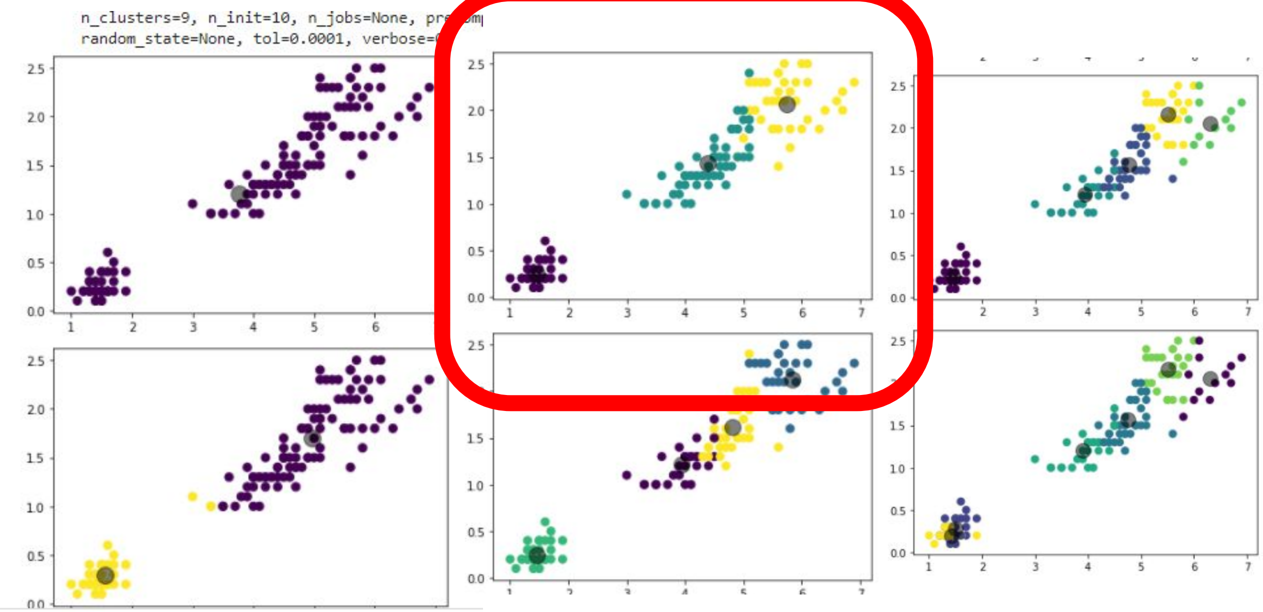
Visualize the target variable (number of clusters)

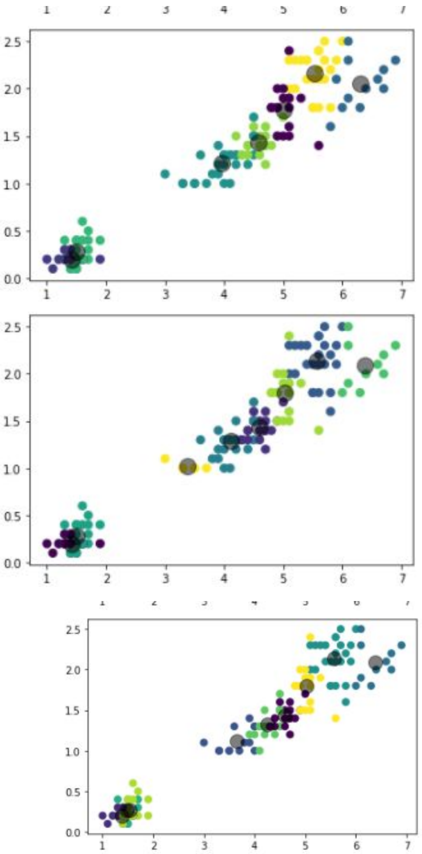
```
<matplotlib.collections.PathCollection at 0x7f3ac0c56d10>
```



Part A - Clustering Learning Algorithm - K-MEANS

Test clustering with different possible values of k (k =1 to 9)





k=3 seems like the best clustering
(similiar to the actual target)

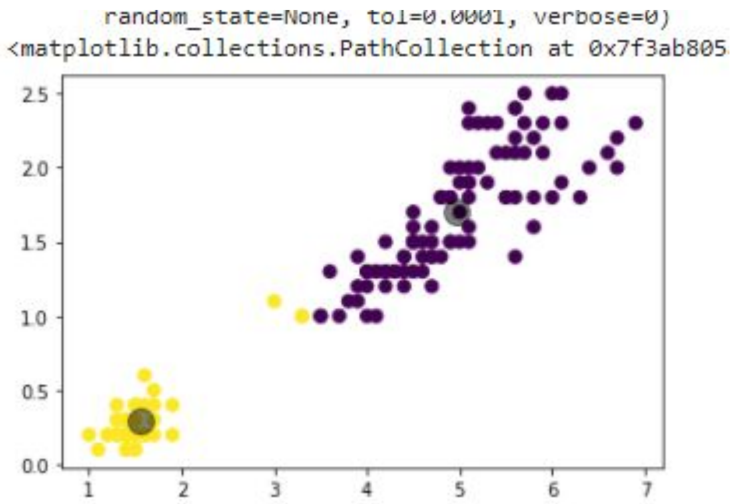
Part A - Clustering Learning Algorithm - K-MEANS

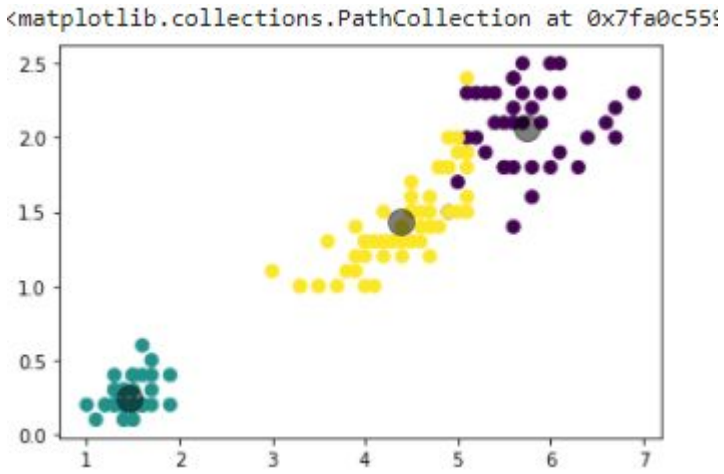
- Determine the best possible value of k using Silhouette Coefficient score:
- Higher Silhouette Coefficient score => a model with better defined clusters.

```
For n_clusters=2, The Silhouette Coefficient is 0.681046169211746
For n_clusters=3, The Silhouette Coefficient is 0.5528190123564091
For n_clusters=4, The Silhouette Coefficient is 0.49745518901737446
For n_clusters=5, The Silhouette Coefficient is 0.4887488870931048
For n_clusters=6, The Silhouette Coefficient is 0.3664804028900824
For n_clusters=7, The Silhouette Coefficient is 0.3566882476581684
For n_clusters=8, The Silhouette Coefficient is 0.34901133143367136
For n_clusters=9, The Silhouette Coefficient is 0.33046129197328006
For n_clusters=10, The Silhouette Coefficient is 0.3110878549031692
```

Part A - Clustering Learning Algorithm - K-MEANS

Although 2 cluster has the highest Silhouette Coefficient score, 2 clusters is not similar to the visualisation of target, 3 clusters is closer to the target



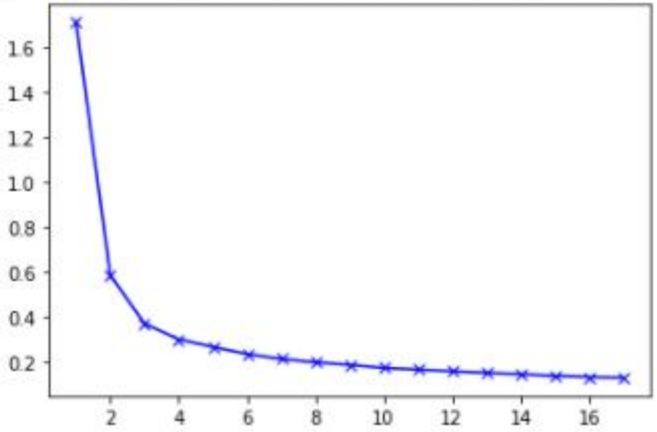


Part A - Clustering Learning Algorithm - K-MEANS

Use the Elbow technique for finding the number of clusters:

From the graph, the value changes at around 3, hence number of clusters = 3

[<matplotlib.lines.Line2D at 0x7ff66fb90710>]



Part A - Clustering Learning Algorithm - K-MEANS

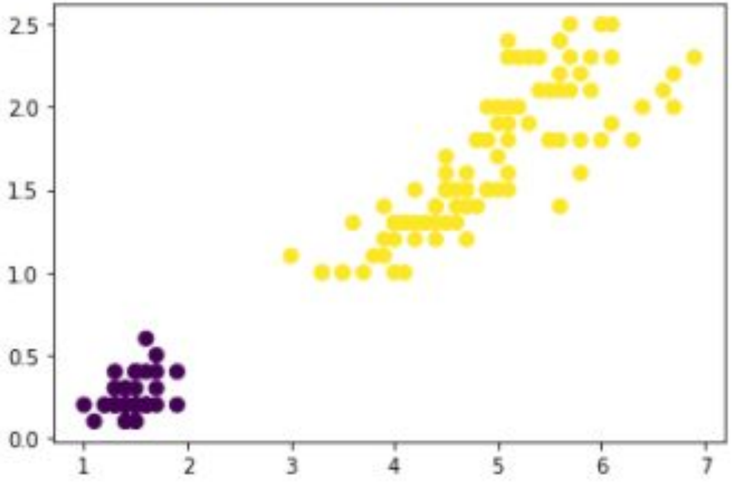
Conclusion :

Best value of k is 3

.

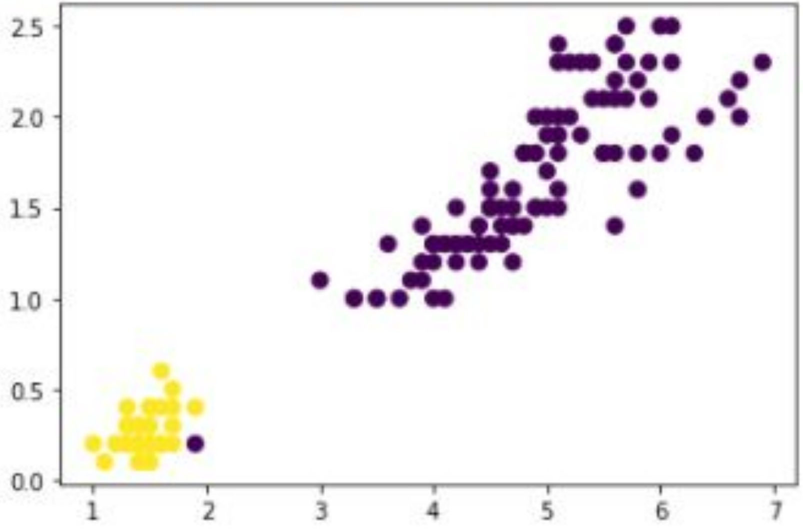
Part A - Clustering Learning Algorithm - DBSCAN

```
DBSCAN(algorithm='auto', eps=0.9, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=5, n_jobs=None, p=None)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
Estimated number of clusters: 2
Estimated number of noise points: 0
Silhouette Coefficient: 0.687
```



Part A - Clustering Learning Algorithm - Gaussian Mixture

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=7, n_init=1, precisions_init=None,
                random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
<matplotlib.collections.PathCollection at 0x7f3ee958ce10>
```



Part B - Deep Learning
(MNIST dataset)

Part B - Deep Learning (MNIST dataset)

Data Pre-processing

(a) Flatten the (28 x 28 pixels) image (3-D array) to a 784 vector (single array) for each image (28*28=784 pixels)

(b) convert from integer values to float (0 (black) -255(white)) as matric multiplication in neural network works better with float

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

Part B - Deep Learning (MNIST dataset)

Data Pre-processing

(c) Scaling

```
X_train = X_train / 255
X_test = X_test / 255
```

- Normalize inputs from 0-255 to 0-1

(b) One-Hot encoding

- 1 integer can represent 10 output neurons using softmax activation

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Part B - Deep Learning Model - Complex CNN

(A) Input Layer :

- **Convolutional** Layer: **Conv2D** - filter size (3,3), 30 filters
- **Pooling** Layer : **MaxPooling2D** - pool size (2,2) to downsize the image and hence reduce the trainable parameters

```
model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu', kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2, 2))) #pooling layer is a compression layer - downsize image (insert i
```

Best Practices - All layers use the **ReLU activation function** and the **he_uniform** kernel initializer (weight initialization scheme)

Part B - Deep Learning Model - Complex CNN

(B) **Hidden** Layers:

```
model.add(Conv2D(15, (3, 3), activation='relu', kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5)) #tune from 0.2 to 0.5
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(50, activation='relu', kernel_initializer='he_uniform'))
```

-

Part B - Deep Learning Model - Complex CNN

(B) **Hidden** Layers:

- **Convolutional** Layer: **Conv2D** - filter size (3,3), 15 filters
- **Pooling** Layer : **MaxPooling2D** - pool size (2,2)
- **Dropout** Layer : For regularization - randomly reducing the number of interconnecting neurons within a neural network based on the given probability (0.5) to avoid overfitting.
- **Flatten** Layer : flattens input image data into a one-dimensional array.
- **Dense** Layer: to interpret the features with 100 nodes
- **Dense** Layer: to interpret the features with 50 nodes

Part B - Deep Learning Model - Complex CNN

(C) Output Layer:

- **Dense** Layer : - With 10 neurons/nodes to predict the image belonging to one of the 10 digits (0-9) using softmax activation function

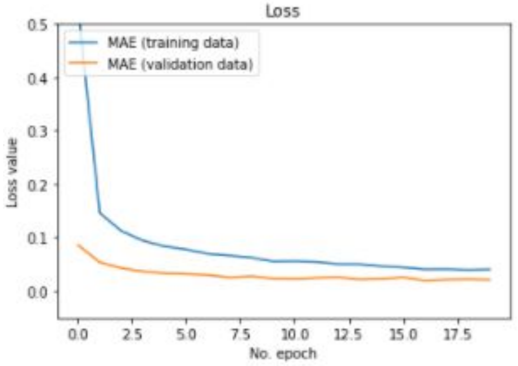
```
#output layer - 10 neurons to predict the images of 10 digits (0-9)
model.add(Dense(num_classes, activation='softmax'))
```

Part B - Deep Learning Model - Complex CNN

Compile the model with categorical cross-entropy loss function optimized with adam optimizer and the classification accuracy metric

```
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'])
```

Part B - Deep Learning (MNIST dataset)



Hyperparameter Tuning:

Number of epochs

- tune from 10 to 20 until the gap between the test error and the training error is small
- As the epoch increased from 10 to 99, the error gradually drop to below 6 but this increases the training time as the training set will have to pass the network 99 times!

Batch size

- tested from 16 to 100 to 150 to 200
- as batch size increases, the error gradually decrease

Part B - Deep Learning (MNIST dataset)

Hyperparameter Tuning:

Number of hidden layers and units

- Added 1 pair of Convolution layer and Pooling layer besides the input layer pair
- Added 2 dense layers

until test error no longer improves.

Dropout for regularization

- tune from 0.2 to 0.5 to increase the probability of dropping out the nodes in the neural network to regularize to avoid overfitting

Part B - Deep Learning (MNIST dataset)

Best Results :

Error: 0.65%

```
784
10
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 24, 24, 30)        780
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 12, 12, 30)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 15)        4065
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 15)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 5, 5, 15)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 375)               0
_________________________________________________________________
flatten_6 (Flatten)          (None, 375)               0
_________________________________________________________________
dense_15 (Dense)             (None, 100)               37600
_________________________________________________________________
dense_16 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_17 (Dense)             (None, 10)                510
=================================================================
Total params: 48,005
Trainable params: 48,005
Non-trainable params: 0
_________________________________________________________________
```

Part B - Deep Learning (MNIST dataset)

Best Results :

Error: 0.65%

```
300/300 - 1s - loss: 0.0663 - accuracy: 0.9792 - val_loss: 0.0253 - val_accuracy: 0.9914
Epoch 9/20
300/300 - 1s - loss: 0.0624 - accuracy: 0.9806 - val_loss: 0.0275 - val_accuracy: 0.9903
Epoch 10/20
300/300 - 1s - loss: 0.0553 - accuracy: 0.9821 - val_loss: 0.0234 - val_accuracy: 0.9914
Epoch 11/20
300/300 - 1s - loss: 0.0557 - accuracy: 0.9821 - val_loss: 0.0225 - val_accuracy: 0.9919
Epoch 12/20
300/300 - 1s - loss: 0.0544 - accuracy: 0.9825 - val_loss: 0.0247 - val_accuracy: 0.9916
Epoch 13/20
300/300 - 1s - loss: 0.0504 - accuracy: 0.9837 - val_loss: 0.0260 - val_accuracy: 0.9911
Epoch 14/20
300/300 - 1s - loss: 0.0502 - accuracy: 0.9838 - val_loss: 0.0214 - val_accuracy: 0.9927
Epoch 15/20
300/300 - 1s - loss: 0.0460 - accuracy: 0.9849 - val_loss: 0.0227 - val_accuracy: 0.9916
Epoch 16/20
300/300 - 1s - loss: 0.0443 - accuracy: 0.9861 - val_loss: 0.0258 - val_accuracy: 0.9917
Epoch 17/20
300/300 - 1s - loss: 0.0405 - accuracy: 0.9868 - val_loss: 0.0191 - val_accuracy: 0.9936
Epoch 18/20
300/300 - 1s - loss: 0.0408 - accuracy: 0.9865 - val_loss: 0.0210 - val_accuracy: 0.9926
Epoch 19/20
300/300 - 1s - loss: 0.0394 - accuracy: 0.9869 - val_loss: 0.0218 - val_accuracy: 0.9924
Epoch 20/20
300/300 - 1s - loss: 0.0404 - accuracy: 0.9865 - val_loss: 0.0207 - val_accuracy: 0.9935
CNN Error: 0.65%
```

References

1. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
2. https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d
3. https://www.analyticsvidhya.com/blog/2021/06/mnist-dataset-prediction-using-keras/
4. https://www.kaggle.com/prashant111/mnist-deep-neural-network-with-keras
5. https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
6. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
7. https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98
8. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
9. https://www.datacamp.com/community/tutorials/cnn-tensorflow-python
10. https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd

Used these references to help to understand and determine which parameters to tune and how to tune