UNIVERSITY OF
ABERDEEN

CS551G - Data Mining and Visualisation

Assessment 1

Name: Avinash Bagul

Student Id: 51987820

Note: Extension was given for this coursework, Deadline for the course work 03 May, 2020.

Contents

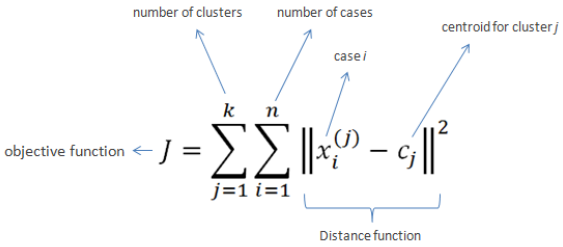
1. Task 1: Unsupervised Learning with K-means and EM for Dog Breed Data Clustering and EDA 3
1.1 K-Means clustering 3
1.2 Description of the Dataset 4
1.3 Exploratory data analysis (EDA) 5
1.4 K-means clustering algorithm and EM algorithm 7
1.5 Internal Cluster Validation 9
2. Task 2: Time Series Classification of User Movement Data 11
1.6 Description of Dataset: 11
1.7 Data Analysis 11
1.8 Padding and Truncation 12
1.9 Building Base Model 12
1.10 10-fold cross validation 13
1.11 Optimizing models: 13
1.12 Conclusion 15
Works Cited 16

1. Task 1: Unsupervised Learning with K-means and EM for Dog Breed Data Clustering and EDA

1.1 K-Means clustering

K-Means Clustering algorithm is an unsupervised machine learning algorithm. In K-Means algorithm the given data is split into the k number of clusters by initially choosing k number of **centroids** at random, where centroid is a data point aimed to be in the center of each cluster. Randomly selected centroids are used to train kNN classifier which results into set of clusters. The main motive is to reduce the sum of distances between data points and centroids, The process of centroid adjustment is repeated until the values for centroid are stabilized.

Objective function can be represented as following:



Here, J is objective function, n represents total number of cases, k represents number of clusters. Distance function is calculated for every case, between the data point x_i and the centroid c_j. So, the **centroids** are the selected points which represent center of each cluster.

K-Means algorithm is simply based on the pairwise **Euclidian distances** between the data points, as the sum of squared deviation of each data point from centroid is equal with the sum of squared Euclidian distances divided by number of data points. After choosing centroid at random, in **assignment step** each data point is picked up and checked for the nearest cluster center. Now, the data point is assigned to the centroid with minimum distance from the data point using Euclidian distance.

In **update step** the, centroids are recalculated by taking the mean of data points in the cluster and then again repeating the assignment step to allot data points to cluster using new obtained cluster centers. This is repeated until the optimum values for centroids are obtained.

Both, **K-Mean** and **Expectation Maximization** are clustering algorithms. **Expected Maximization algorithm (EM)** is a method to find the maximum likelihood of parameters in statistical model.

The difference between this algorithm is the method used for calculating the distance between data points. K-Means algorithm use Euclidian distance to calculate the distance between two data

items while Expectation maximization uses statistical method to calculate the distance. It needs more computational power and it is difficult to optimized compared to K-Means.

According to the comparison used in (Yong Gyu Juyng, 2014) it is seen that K-Means was slower compared with EM but it gave higher accuracy (over 94%), where, Expectation Maximization algorithm gave less accuracy (87%) but consumed less time.

1.2 Description of the Dataset

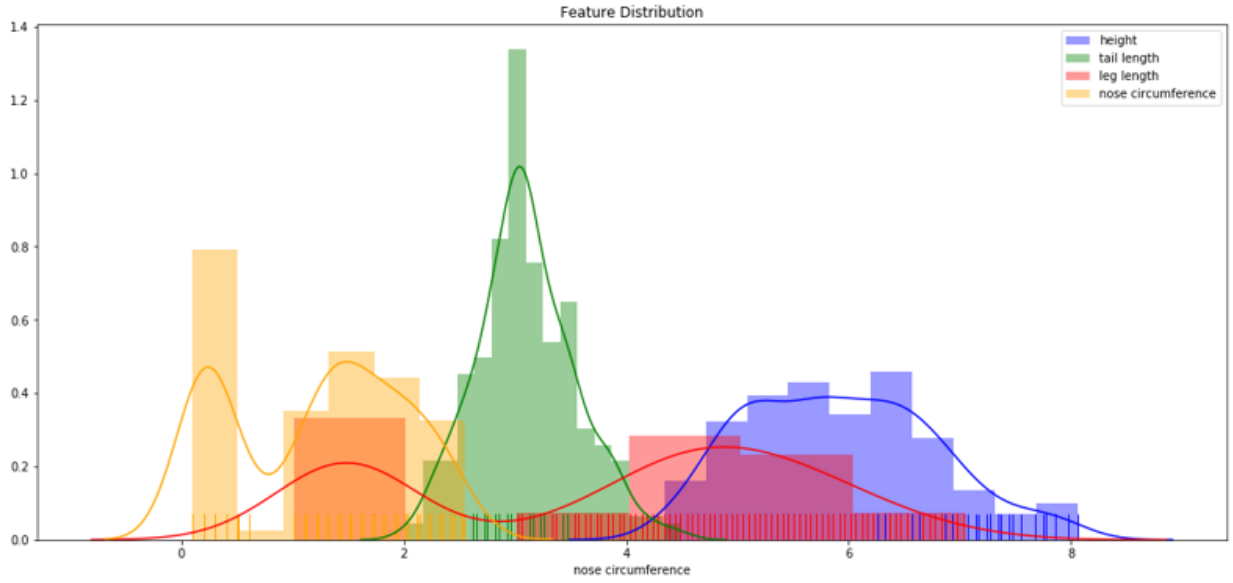
The Given Dog Breed Dataset contains data for 300 dogs detailing their properties like; height, tail length, leg length and nose circumference. No missing data was found in the dataset. After checking for the outliers, total 2 outliers were found in the 'tail length' of the dataset using Z score method. All the features in the dataset are numeric (float datatype). The dataset is in .csv format and it is being read using pandas library's read_csv method. Fig(1.1) below is the snippet showing some basic information of dataset.

```
In [6]: mydata.shape
Out[6]: (300, 4)

In [7]: mydata.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   height              300 non-null    float64
 1   tail length         300 non-null    float64
 2   leg length          300 non-null    float64
 3   nose circumference  300 non-null    float64
dtypes: float64(4)
memory usage: 9.5 KB
```

Fig(1.1)

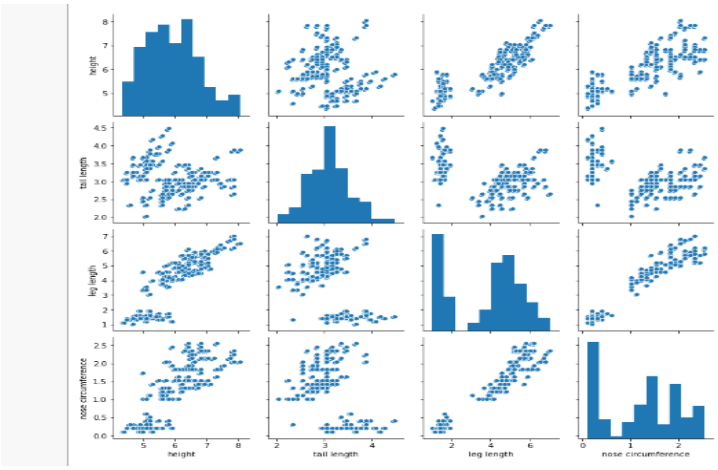
1.3 Exploratory data analysis (EDA)

First I checked for the null values in dataset using .isnull() function and results were negative as there is no missing data in the dog breed dataset.



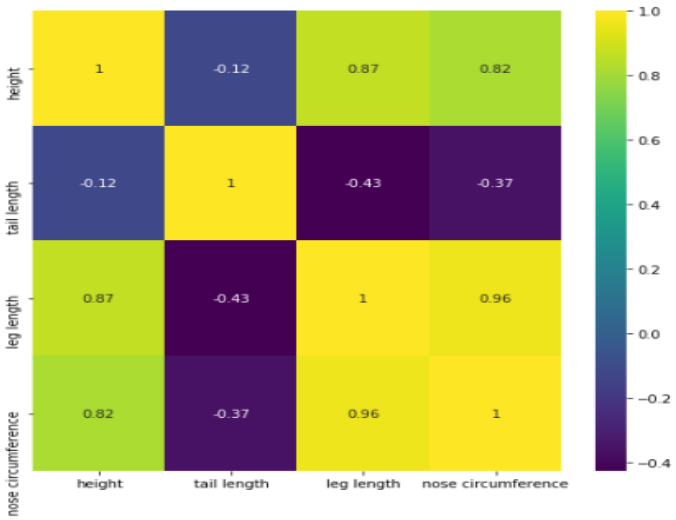
Fig(1.2)

From the above dist plot (Fig(1.2)) from seaborn library shows the combined feature distributions of all the features for a dog in the given dataset. As we can observe from the plot that blue and green i.e. Height feature and tail length feature is well distributed. Whereas, leg length and nose circumference are not well distributed and are imbalanced. Also from the leg length distribution we can see that it has minimum value as (1.010) and maximum value (7.038), there is hardly any value between (2) and (3.5) for leg length on X-axis, so we can say it is not well distributed.



Fig(1.3)

The above pairplot (Fig(1.3)) from seaborn library shows correlations between features. From the figure we can say that there are at least two clear clusters in the dataset. And with the help of seaborn heatmap (Fig(1.4)) we can conclude that features: leg length and nose circumference are closely correlated with each other.



Fig(1.4)

To find outliers in data I chose Z score method and found total 2 outliers in tail length feature. Other features did not have outliers. Details can be seen in code snippet below Fig(5).

Checking for Outliers

In [11]:
```
def detect_outlier(data_1):
    outliers=[]
    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)


    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return len(outliers)

datahead = ['height','tail length','leg length','nose circumference']

for i in range(0,4):
    outlier_datapoints = detect_outlier(mydata[datahead[i]])
    print("Number of Outliers in",datahead[i], " are " , outlier_datapoints)
```

```
Number of Outliers in height  are  0
Number of Outliers in tail length  are  2
Number of Outliers in leg length  are  0
Number of Outliers in nose circumference  are  0
```

1.4 K-means clustering algorithm and EM algorithm

I have used scikit-learn library for K-Means and EM algorithm, implemented for dog breed data clustering. Let us understand step by step:

K-Means:

First step was to assign K value i.e. number of desired clusters. According to the requirement given in assessment 1 Task1, K is supposed to be 3 and 4.

Next, I have initialized sklearn.cluster.KMeans model with following parameters:

- **n_clusters:** n number of clusters, in our case we have k as desired number of clusters in the first step.
- **init:** I choose Init as '***k-means++***' which is also by default value. It selects the initial cluster centers in a smart way for fast convergence. Where '***random***' choose at random from data as initial **k** centroids.
- **random_state:** it makes randomness deterministic.
- **algorithm:** I have used **'elkan'. 'auto'** as it allows to choose {"full" algorithm} for sparse data and {"elkan" algorithm} for dense data. "**elkan**" is considered to be efficient but it does not support sparse data. since our dog breed dataset does not have any missing values I have manually selected "elkan" algorithm.

```
# K-Means method for clustering (Unsupervised Learning)
k=3 # K number of clusters
kmean1 = KMeans(n_clusters = k, init='k-means++',n_init=10,verbose=0, random_state=42, algorithm='elkan')
data = mydata.iloc[:,[0,1,2,3]].values
kmean1.fit(data)
print("number of iterations", kmean1.n_iter_)
print(kmean1)
```

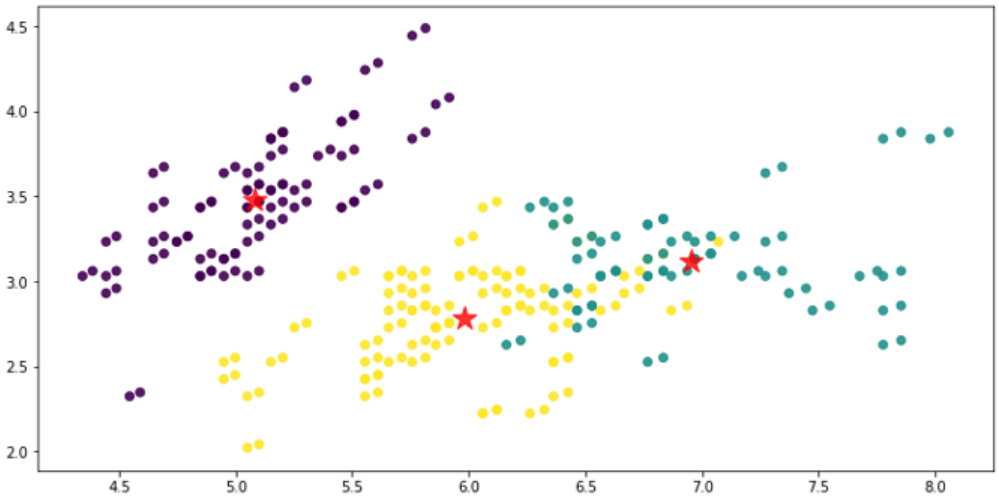
```
number of iterations 12
KMeans(algorithm='elkan', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

After this, when I fit the data into the above model and check for cluster centers the following results are obtained.

```
clusters = kmean1.cluster_centers_
print(clusters)
```

```
[[5.08109    3.47942    1.48393    0.24969   ]
 [6.95518182 3.12166234 5.81480519 2.09336364]
 [5.98078862 2.7857561  4.45673171 1.45560163]]
```

Following clusters and cluster centers in (fig(1.5)) for k=3 were obtained after using fit_predict() and matplotlib.pyplot scatter for visualization.



Fig(1.5)

Expected Maximization (EM):

EM algorithm is clustering algorithm similar to K-Means algorithm, but with different method of calculating the distance between data point and centroid (cluster center)

For Using Expected Maximization algorithm I have used GaussianMixture from sklearn.mixture.

Before initializing model, I have scaled the data using Standard scaler function from preprocessing.

```
data_scaler = preprocessing.StandardScaler()
data_scaler.fit(data)
scaled_data = data_scaler.transform(data)

sdata = pd.DataFrame(scaled_data, columns = mydata.columns)
dt = sdata.iloc[:,[0,1,2,3]].values
```

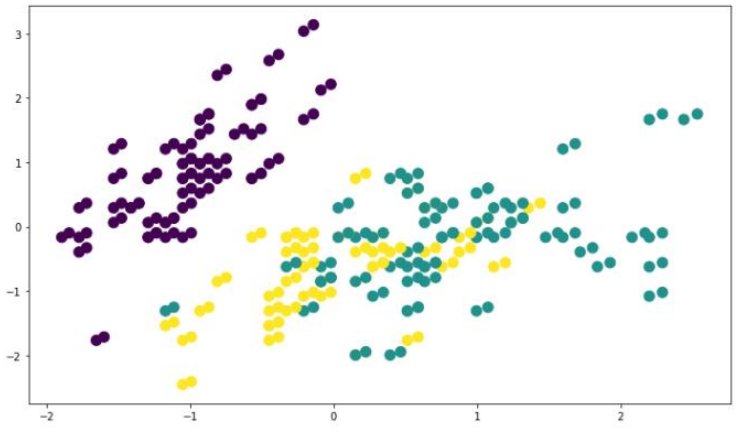
Then, I initialize GaussianMixture model with following parameters:

- n_components: n number of clusters i.e. 3 or 4 according to the assessment description.
- init_params: ‘kmeans’ this states that responsibilities are being initialized using kmeans.
- Random_state: random state is the seed used by random number generator.

```
n=3
gm = GaussianMixture(n_components=n,init_params='kmeans',random_state=32)
print(gm)
```

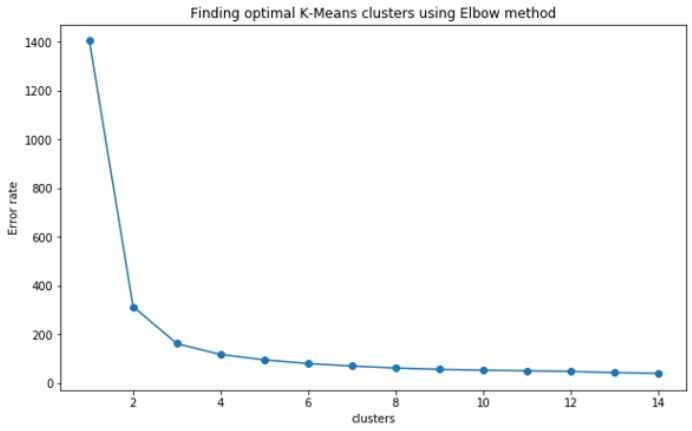
```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=1, precisions_init=None,
                random_state=32, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
```

After fitting the data and plotting it on matplotlib.pyplot scatter we get the following results shown in Fig(6)



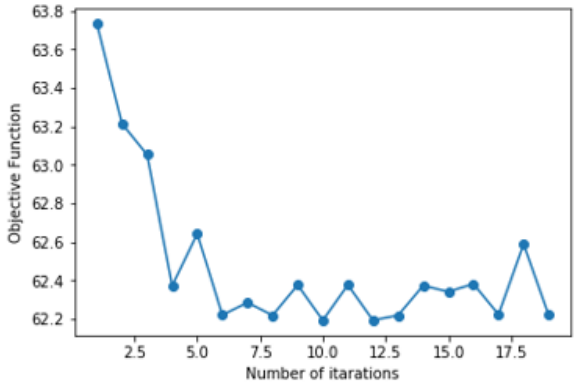
1.5 Internal Cluster Validation

There are several approaches for internal cluster validation i.e. Elbow method, Silhouette method, Dunn index, etc. For internal validation for this task, I have used the most common method i.e. the Elbow method:



Elbow method is a heuristic used in determining the number of clusters in our dog breed dataset. It helps us decide the right number of clusters based on the plotting as elbow of the curve is used as the number of clusters (k).

Below is the line plot, where *x* axis is the iteration step and *y* axis is the objective function:


63.8
63.6
63.4
63.2
63.0
62.8
62.6
62.4
62.2
Objective Function
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of itarations

2. Task 2: Time Series Classification of User Movement Data

1.6 Description of Dataset:

Ambient Living Wireless Sensor Network Dataset:

Dataset contains data from wireless sensor network deployed in real world office environments. Ambient Living dataset is given in 216 different .csv files, to read the data, all the files were uploaded on colab. Each file contain data given by 4 sensors. Given data files are of different length. There is no missing data in the dataset. All the sensor data have float datatype. Uploaded files are being read using pandas read_csv function and values are appended in a list. Shape of the appended list is (8080, 4).

	Sensor 1	Sensor 2	Sensor 3	Sensor 4
0	-0.90476	-0.48	0.28571	0.30
1	-0.57143	-0.32	0.14286	0.30
2	-0.38095	-0.28	-0.14286	0.35
3	-0.28571	-0.20	-0.47619	0.35
4	-0.14286	-0.20	0.14286	-0.20

(8080, 4)

Ambient Living Target Dataset:

Target labels are of int datatype.

This dataset contains the class labels (1,-1)with the data shape (216, 1). There are 111 examples of (-1) class and 105 examples of (1) class.

1.7 Data Analysis

I performed some data analysis to get basic information on data, data was scanned for null values, but no null value was found in the dataset.

Following is the data description shown in Fig(2.1):

```
data.describe()
```

	Sensor 1	Sensor 2	Sensor 3	Sensor 4
count	8080.000000	8080.000000	8080.000000	8080.000000
mean	-0.078667	-0.122319	-0.172958	-0.106438
std	0.474131	0.448930	0.497846	0.445637
min	-1.000000	-1.000000	-1.000000	-1.000000
25%	-0.454550	-0.466670	-0.590910	-0.446810
50%	-0.045455	-0.155560	-0.155560	-0.100000
75%	0.285710	0.200000	0.142860	0.200000
max	1.000000	1.000000	1.000000	1.000000

Fig(2.1)

Description shows the statistical information for sensor data with total count 8080.

There are total 111 cases of lass label -1 and 105 cases of label 1. So the data is slightly imbalanced.

```
[22] print(pd.value_counts(targets["label"]))

     -1    111
      1    105
     Name: label, dtype: int64
```

Fig(2.2)

1.8 Padding and Truncation

- **80^{th} Quartile:**

As the given data has variable lengths so it is must to convert data into same length. Assessment requirement ask to take 80^{th} quartile, so with the help of quantile() method, I got 80^{th} quartile as 54.

- **Padding:**

As calculated, 80^{th} quartile is 54. Hence, all the data must be padded to length 54. All the data was padded with 0 to avoid noise in the data till the length 103.

- **Truncation:**

All the data was cropped to length 54 using keras.preprocessing pad_sequences() method.

1.9 Building Base Model

Dataset is dived into training data and testing data. I divided the data in 80:20 ratio allotting 80% to the training dataset and 20% to testing dataset.

Mostly code was used with reference from the course practical. After making small changes to the data labels, I am all set to build base model (LSTM).

Long-Short Term Memory model are good with handling the time-series data, to implement LSTM, I have used keras library and Tensorflow in the backend. I have initialized a basic model with three layers: (LSTM(unit=8), LSTM(unit=4) and Dense(1, activation='sigmoid')) and epoch = 100 while fitting the models. Other parameters will be changed later to optimize the model to get higher accuracy.

My Base model accuracies fluctuated at each run, I got 64.28% accuracy for my base model.

1.10 10-fold cross validation

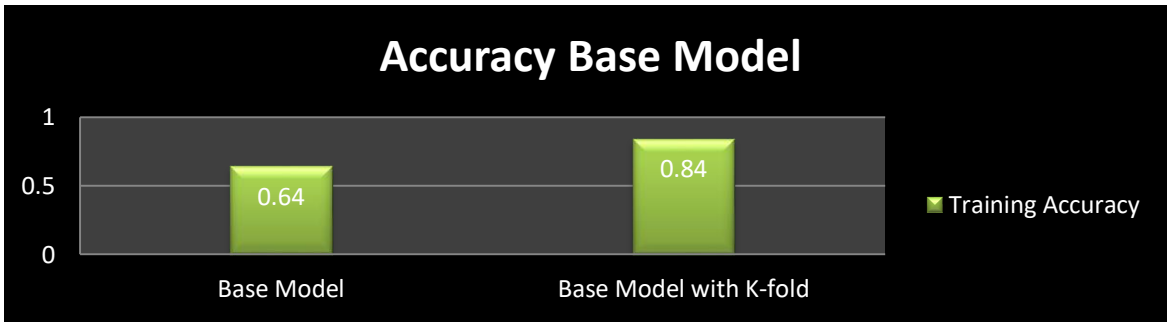
Cross validation was done using K-fold method with 10 number of folds. K-fold was used from sklearn library and to measure accuracy, f1 score, recall and confusion matrix I used sklearn.metrics. Using 10-fold cross validation splits the data into 90:10 and randomize the test and train data. With this cross validation technique accuracy result improves. Fig(2.3) shows the result accuracies for base model and base model with cross validation.

```
Epoch 98/100
3/3 [==============================] - 0s 37ms/step
Epoch 99/100
3/3 [==============================] - 0s 38ms/step
Epoch 100/100
3/3 [==============================] - 0s 38ms/step


 ******************************************


Accuracy for the base model is:  0.6428571428571429


******************************************
```

```
#################### Total Mean Scores ####################

Training Accuracy........ 0.8679526355996945

Validation Accuracy...... 0.7790849673202616


Training Precision....... 0.8751350315642613

Training Recall.......... 0.8660026221586549

Training f1 Score........ 0.8625985214994494


Validation Precision..... 0.7295383986928105

Validation Recall........ 0.7744444444444445

Validation f1 Score...... 0.7139421627414627
************************************************************
```

Fig(2.3)

Below is the comparison of the obtained results:



1.11 Optimizing models:

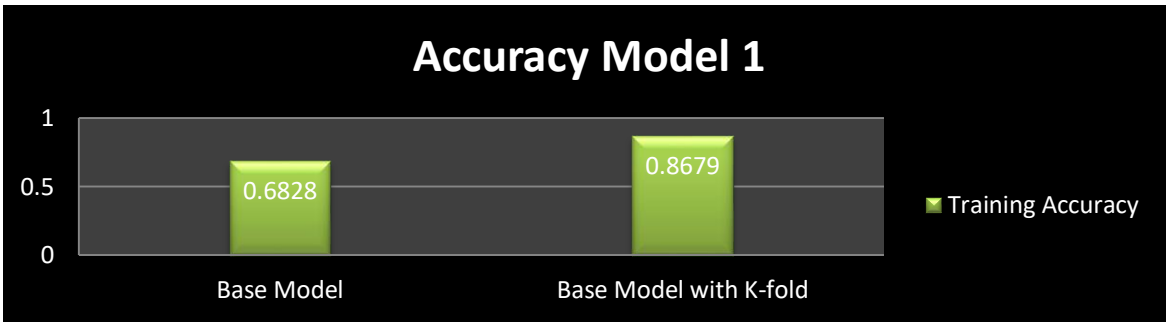
- **Optimizing Model 1:**

To optimize model I will add layers to the model and see the changes in the accuracy.

I have added two layers of LSTM having units 512 with return sequence true and 128. After fitting this model with 200 epoch, validation split as 0.1 and batch size 128 I got 64.28%

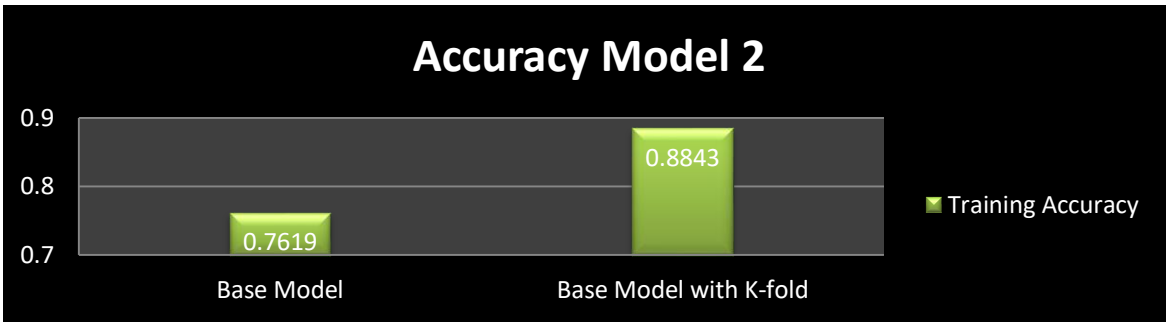
accuracy without implementation of K-fold. After using cross validation, Training Accuracy increased to 86.79%. Hence, I think just adding layer with more units does not help increase the accuracy of the model, now I will try using dropout for next model and try to optimize it.

The accuracy result comparison for Model 1 is as follows:



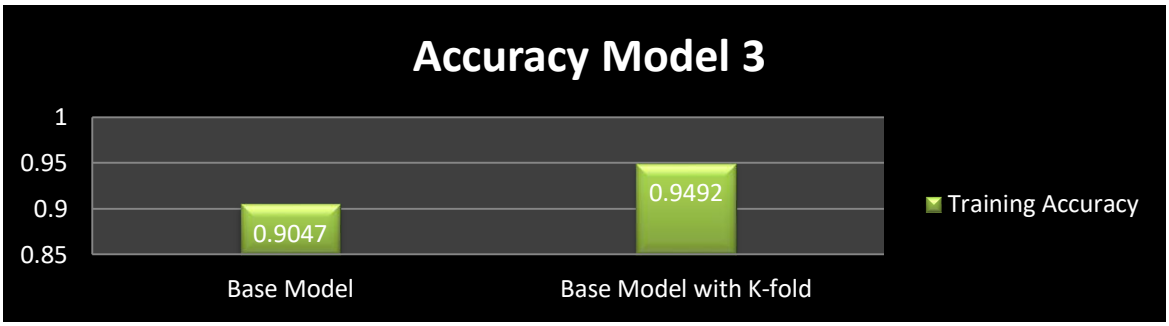
- **Optimizing Model 2:**

Now to optimize the model, I will add 2 dropout layers to prevent any over-fitting of data. Adding a dropout layer will reduce the interdependent learning amongst the neurons by regularizing neural network. Results were as following 76.19% accuracy without cross validation and 88.43% accuracy with 10-fold cross validation.

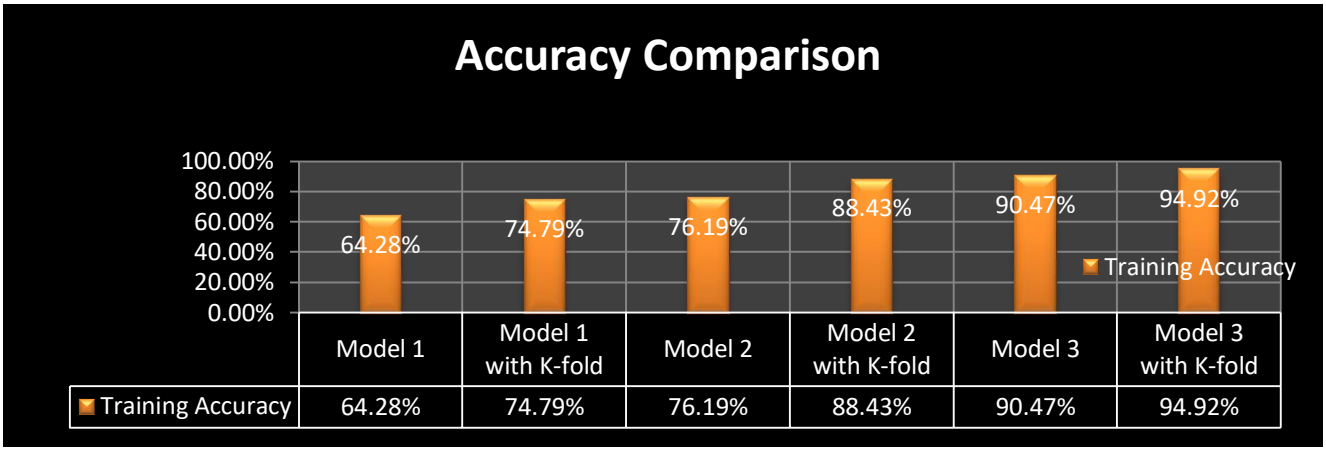


- **Optimizing Model 3:**

For optimizing further, I have tried to change the learning rate of the optimizer ‘adam’ from: 0.001 to 0.010. added another dense layer to the model and this time surprisingly I got better results. Accuracy results without cross validation gave me 90% accuracy and after implementing 10-fold cross validation on model 3, I got 94.92 (Approx 95%) accuracy.



Now let's have a look at total accuracy statistics for all models with and without implementation of 10-fold cross validation. According to the accuracy statistics per model in Fig(2.4), we can see that optimizing each model has shown increase in the accuracy.



1.12 Conclusion

This assessment task has given me opportunity to explore LSTM recurrent neural network, which is highly implemented in the field of Deep Learning. The time series data provided was really good for learning as it required some necessary pre-processing like padding and truncating.

It can be seen from the program output, how accuracy has increased in each attempt to optimize the model. Adding LSTM layers has slightly increased the accuracy but adding dense layer and dropout has significantly increased the accuracy. Adding dropout layer avoids over-fitting of data and clears some neurons.

Using 10-fold cross validation splits the data in 90:10 ratio each time and shuffles the data in such a way that it produces best splits, Hence giving more accuracy. sklearn, keras,Tensorflow, etc. libraries were very useful for performing this task. Accuracy, f1 score, recall and confusion metrics were used form sklearn.

Works Cited

Yong Gyu Juyng, M. S. (2014). Clustering performance comparison using K-means and expectation maximization algorithms. *Biotechnology, Biotechnological Equipment*, 28(sup1): S44–S48.