Data Mining Final Project

- Rajendra Prasad Patil

Glossary:

- Import libraries
- Load dataset
- Analysis on dataset
- Splitting the dataset into labels and features
- Performing normalization on dataset
- Splitting dataset using K fold
- Running the model
  - SVM Model
  - K Nearest Neighbors
  - Random Forest Classifier
- Output Performance Metrics

Importing libraries

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')

# SVM classifier
from sklearn import svm

# KNN classifier
from sklearn.neighbors import KNeighborsClassifier

#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

# Import libraries for lstm classification
from keras.layers import Dense, Dropout, LSTM, Embedding
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential

# for checking the model accuracy
from sklearn.metrics import confusion_matrix, accuracy_score, classification_
```

Loading the dataset

In [2]:
```
dataset = load_breast_cancer()
```

In [3]:
```
input_length = len(dataset['data'][0])
```

Preliminary analysis

In [4]:
```
class_names = dataset['target_names']
print('Target variables  : ', class_names)

(unique, counts) = np.unique(dataset['target'], return_counts=True)

print('Unique values of the target variable', unique)
print('Counts of the target variable :', counts)
```

```
Target variables  :  ['malignant' 'benign']
Unique values of the target variable [0 1]
Counts of the target variable : [212 357]
```

- The dataset is suited for binary classification
- The dataset has no skewed nature

The data is split into features and labels

In [5]:
```
X = dataset['data']
y = dataset['target']
```

Apply normalization operation for numerical stability

In [6]:
```
standardizer = StandardScaler()
X = standardizer.fit_transform(X)
```

Performance Metrics

Function to calculate all the available performance metrics

In [7]:
```
performance_metrics = ['True Negative', 'False Positive', 'False Negative', '
                       'Precision', 'Accuracy', 'F1 Score', 'Error Rate', 'Ne
                       'False Discovery Rate', 'False Negative Rate', 'Balanc
                       'Heidke Skill Score']

def compute_performance_metrics(prediction, y_test, df, is_lstm = False):

    if is_lstm:
        threshold = 0.80
        for i, each in enumerate(prediction):
            if each[0] > threshold:
                prediction[i] = 1
            else:
                prediction[i] = 0

    TN, FP, FN, TP = confusion_matrix(y_test, prediction).ravel()

    sensitivity = TP / (TP + FN)
    specificity = TN / (FP + TN)
    precision = TP / (TP + FP)
    accuracy =  (TP+TN) /(TP+FP+TN+FN)
    f1_score = 2 * TP / ((2 * TP) + FP + FN)
    error_rate = (FP + FN) / (TP + FP + FN + TN)
    negative_predicted_value = TN / (TN + FN)
    false_positive_rate = FP / (FP + TN)
    false_discovery_rate = FP / (FP + TP)
    false_negative_rate = FN / (FN + TP)
    balanced_accuracy = 0.5 * ((TP / (TP + FN)) + (TN / (TN + FP)))
    true_skill_statistics = ((TP / (TP + FN)) - (FP / (TN + FP)))
    heidke_skill_score = 2 * ((TP * TN) - (FP * FN)) / (((TP + FN) * (FN + T

    df = df.append({performance_metrics[0]: TN, performance_metrics[1]: FP, p
                    performance_metrics[3]: TP, performance_metrics[4]: sensi
                    performance_metrics[6]: precision, performance_metrics[7]
                    performance_metrics[9]: error_rate, performance_metrics[1
                    performance_metrics[11]: false_positive_rate, performance
                    performance_metrics[13]: false_negative_rate, performance
                    balanced_accuracy, performance_metrics[15]: true_skill_st
                    performance_metrics[16]: heidke_skill_score}, ignore_inde
    return df
```

K-fold cross validation

In [8]:
```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=10, shuffle=True, random_state=0)
```

Dataframes for performance metrics

In [9]:
```
svm_metrics_df = pd.DataFrame(columns=performance_metrics)
kn_metrics_df = pd.DataFrame(columns=performance_metrics)
rf_metrics_df = pd.DataFrame(columns=performance_metrics)
lstm_metrics_df = pd.DataFrame(columns=performance_metrics)
```

SVM Model

In [10]:

```
svm_model = svm.SVC()
for train_index, test_index in kfold.split(X):
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train

    # we train the algorithm with training data and training output
    svm_model.fit(X_train, y_train)

    # we pass the testing data to the stored algorithm to predict the outcome
    prediction = svm_model.predict(X_test)

    # print metrics
    svm_metrics_df = compute_performance_metrics(prediction, y_test, svm_metr

svm_metrics_df.index += 1
svm_metrics_df.loc['Average'] = svm_metrics_df.mean()
```

In [11]:

```
svm_metrics_df
```

Out[11]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy
1	22.0	0.0	0.0	35.0	1.000000	1.000000	1.000000	1.000000
2	23.0	2.0	1.0	31.0	0.968750	0.920000	0.939394	0.947368
3	15.0	1.0	1.0	40.0	0.975610	0.937500	0.975610	0.964912
4	20.0	0.0	0.0	37.0	1.000000	1.000000	1.000000	1.000000
5	17.0	1.0	0.0	39.0	1.000000	0.944444	0.975000	0.982456
6	19.0	3.0	0.0	35.0	1.000000	0.863636	0.921053	0.947368
7	22.0	1.0	1.0	33.0	0.970588	0.956522	0.970588	0.964912
8	23.0	0.0	2.0	32.0	0.941176	1.000000	1.000000	0.964912
9	18.0	0.0	0.0	39.0	1.000000	1.000000	1.000000	1.000000
10	25.0	0.0	0.0	31.0	1.000000	1.000000	1.000000	1.000000
Average	20.4	0.8	0.5	35.2	0.985612	0.962210	0.978164	0.977193

K-Nearest Neighbors

In [12]:
```
model = KNeighborsClassifier(n_neighbors=3) # this examines 3 neighbors for p

for train_index, test_index in kfold.split(X):
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train

    # we train the algorithm with training data and training output
    model.fit(X_train, y_train)

    # we pass the testing data to the stored algorithm to predict the outcome
    prediction = model.predict(X_test)

    # print metrics
    kn_metrics_df = compute_performance_metrics(prediction, y_test, kn_metric

kn_metrics_df.index += 1
kn_metrics_df.loc['Average'] = kn_metrics_df.mean()
```

In [13]:
```
kn_metrics_df
```

Out[13]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy
1	21.0	1.0	0.0	35.0	1.000000	0.954545	0.972222	0.982456
2	21.0	4.0	0.0	32.0	1.000000	0.840000	0.888889	0.929825
3	14.0	2.0	1.0	40.0	0.975610	0.875000	0.952381	0.947368
4	19.0	1.0	0.0	37.0	1.000000	0.950000	0.973684	0.982456
5	17.0	1.0	1.0	38.0	0.974359	0.944444	0.974359	0.964912
6	19.0	3.0	0.0	35.0	1.000000	0.863636	0.921053	0.947368
7	22.0	1.0	0.0	34.0	1.000000	0.956522	0.971429	0.982456
8	23.0	0.0	0.0	34.0	1.000000	1.000000	1.000000	1.000000
9	18.0	0.0	0.0	39.0	1.000000	1.000000	1.000000	1.000000
10	23.0	2.0	0.0	31.0	1.000000	0.920000	0.939394	0.964286
Average	19.7	1.5	0.2	35.5	0.994997	0.930415	0.959341	0.970113

Random Forest Classifier

In [14]:
```
#Create a Gaussian Classifier
model = RandomForestClassifier(n_estimators=100)

for train_index, test_index in kfold.split(X):
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train

    # we train the algorithm with training data and training output
    model.fit(X_train, y_train)

    # we pass the testing data to the stored algorithm to predict the outcome
    prediction = model.predict(X_test)

    # print metrics
    rf_metrics_df = compute_performance_metrics(prediction, y_test, rf_metric

rf_metrics_df.index += 1
rf_metrics_df.loc['Average'] = rf_metrics_df.mean()
```

In [15]:
```
rf_metrics_df
```

Out[15]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy
1	21.0	1.0	2.0	33.0	0.942857	0.954545	0.970588	0.947368
2	24.0	1.0	0.0	32.0	1.000000	0.960000	0.969697	0.982456
3	15.0	1.0	2.0	39.0	0.951220	0.937500	0.975000	0.947368
4	19.0	1.0	0.0	37.0	1.000000	0.950000	0.973684	0.982456
5	17.0	1.0	1.0	38.0	0.974359	0.944444	0.974359	0.964912
6	17.0	5.0	2.0	33.0	0.942857	0.772727	0.868421	0.877193
7	22.0	1.0	0.0	34.0	1.000000	0.956522	0.971429	0.982456
8	20.0	3.0	1.0	33.0	0.970588	0.869565	0.916667	0.929825
9	18.0	0.0	0.0	39.0	1.000000	1.000000	1.000000	1.000000
10	24.0	1.0	0.0	31.0	1.000000	0.960000	0.968750	0.982143
Average	19.7	1.5	0.8	34.9	0.978188	0.930530	0.958859	0.959618

LSTM classifier

In [16]:

```
model = Sequential()
model.add(LSTM(20, input_shape=(input_length, 1)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

for train_index, test_index in kfold.split(X):
    print('*'*100)
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_

    # we train the algorithm with training data and training output
    model.fit(X_train, y_train, batch_size=8, epochs=10, validation_data=(X_t

    # we pass the testing data to the stored algorithm to predict the outcome
    prediction = model.predict(X_test)

    # print metrics
    lstm_metrics_df = compute_performance_metrics(prediction, y_test, lstm_me

lstm_metrics_df.index += 1
lstm_metrics_df.loc['Average'] = lstm_metrics_df.mean()
```

```
Metal device set to: Apple M1
****************************************************************************
************************
2021-12-04 16:56:43.997537: I tensorflow/core/common_runtime/pluggable_device
/pluggable_device_factory.cc:305] Could not identify NUMA node of platform GP
U ID 0, defaulting to 0. Your kernel may not have been built with NUMA suppor
t.
2021-12-04 16:56:43.997965: I tensorflow/core/common_runtime/pluggable_device
/pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/r
eplica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (d
evice: 0, name: METAL, pci bus id: <undefined>)
2021-12-04 16:56:44.209424: W tensorflow/core/platform/profile_utils/cpu_util
s.cc:128] Failed to get CPU frequency: 0 Hz
Epoch 1/10
2021-12-04 16:56:44.590973: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
2021-12-04 16:56:44.711991: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
 5/64 [=>............................] - ETA: 0s - loss: 0.8705 - accuracy:
0.2250
2021-12-04 16:56:44.831839: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
64/64 [==============================] - 2s 18ms/step - loss: 0.7669 - accura
cy: 0.3457 - val_loss: 0.7148 - val_accuracy: 0.4912
2021-12-04 16:56:45.915998: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
2021-12-04 16:56:45.966215: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.6055 - accura
cy: 0.7324 - val_loss: 0.4008 - val_accuracy: 0.9298
Epoch 3/10
```

```
64/64 [==============================] - 1s 15ms/step - loss: 0.3536 - accura
cy: 0.9043 - val_loss: 0.3022 - val_accuracy: 0.8947
Epoch 4/10
64/64 [==============================] - 1s 14ms/step - loss: 0.3118 - accura
cy: 0.9199 - val_loss: 0.2717 - val_accuracy: 0.9123
Epoch 5/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2680 - accura
cy: 0.9277 - val_loss: 0.2278 - val_accuracy: 0.9474
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2960 - accura
cy: 0.9238 - val_loss: 0.2116 - val_accuracy: 0.9474
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2694 - accura
cy: 0.9219 - val_loss: 0.1653 - val_accuracy: 0.9649
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2853 - accura
cy: 0.9180 - val_loss: 0.2820 - val_accuracy: 0.9123
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.3803 - accura
cy: 0.8848 - val_loss: 0.2712 - val_accuracy: 0.9123
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.3596 - accura
cy: 0.8789 - val_loss: 0.2642 - val_accuracy: 0.9123
****************************************************************************
***********************
Epoch 1/10
 5/64 [=>............................] - ETA: 0s - loss: 0.3474 - accuracy:
0.8750
2021-12-04 16:56:54.706389: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
2021-12-04 16:56:54.742269: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
64/64 [==============================] - 1s 15ms/step - loss: 0.3186 - accura
cy: 0.8906 - val_loss: 0.3013 - val_accuracy: 0.8947
Epoch 2/10
64/64 [==============================] - 1s 14ms/step - loss: 0.3233 - accura
cy: 0.8945 - val_loss: 0.3014 - val_accuracy: 0.8947
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.3109 - accura
cy: 0.8926 - val_loss: 0.2979 - val_accuracy: 0.8947
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2944 - accura
cy: 0.9004 - val_loss: 0.2953 - val_accuracy: 0.8947
Epoch 5/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2919 - accura
cy: 0.9062 - val_loss: 0.3285 - val_accuracy: 0.8947
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2684 - accura
cy: 0.9121 - val_loss: 0.2973 - val_accuracy: 0.8947
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2754 - accura
cy: 0.9082 - val_loss: 0.2935 - val_accuracy: 0.8947
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2699 - accura
cy: 0.9102 - val_loss: 0.3311 - val_accuracy: 0.8947
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.3057 - accura
cy: 0.9102 - val_loss: 0.4018 - val_accuracy: 0.8421
Epoch 10/10
```

```
64/64 [==============================] - 1s 15ms/step - loss: 0.2662 - accura
cy: 0.9180 - val_loss: 0.3247 - val_accuracy: 0.8596
****************************************************************************
**********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2479 - accura
cy: 0.9141 - val_loss: 0.3898 - val_accuracy: 0.8596
Epoch 2/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2531 - accura
cy: 0.9141 - val_loss: 0.3712 - val_accuracy: 0.8596
Epoch 3/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2453 - accura
cy: 0.9121 - val_loss: 0.2918 - val_accuracy: 0.8947
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2315 - accura
cy: 0.9160 - val_loss: 0.2987 - val_accuracy: 0.8772
Epoch 5/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2359 - accura
cy: 0.9160 - val_loss: 0.2014 - val_accuracy: 0.9298
Epoch 6/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2201 - accura
cy: 0.9160 - val_loss: 0.2331 - val_accuracy: 0.9123
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2199 - accura
cy: 0.9141 - val_loss: 0.2189 - val_accuracy: 0.9123
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2314 - accura
cy: 0.9141 - val_loss: 0.2772 - val_accuracy: 0.8947
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2394 - accura
cy: 0.9043 - val_loss: 0.2581 - val_accuracy: 0.8772
Epoch 10/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2241 - accura
cy: 0.9082 - val_loss: 0.2674 - val_accuracy: 0.8947
****************************************************************************
**********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2142 - accura
cy: 0.9141 - val_loss: 0.1580 - val_accuracy: 0.9474
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2356 - accura
cy: 0.8926 - val_loss: 0.2279 - val_accuracy: 0.9123
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2346 - accura
cy: 0.9102 - val_loss: 0.2239 - val_accuracy: 0.9123
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2043 - accura
cy: 0.9160 - val_loss: 0.1522 - val_accuracy: 0.9474
Epoch 5/10
64/64 [==============================] - 1s 14ms/step - loss: 0.2003 - accura
cy: 0.9277 - val_loss: 0.1623 - val_accuracy: 0.9474
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2188 - accura
cy: 0.9199 - val_loss: 0.2400 - val_accuracy: 0.9123
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2053 - accura
cy: 0.9199 - val_loss: 0.2312 - val_accuracy: 0.9298
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2161 - accura
```

```
cy: 0.9141 - val_loss: 0.2019 - val_accuracy: 0.9298
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2124 - accura
cy: 0.9238 - val_loss: 0.1460 - val_accuracy: 0.9649
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2222 - accura
cy: 0.9082 - val_loss: 0.2161 - val_accuracy: 0.9298
****************************************************************************
***********************
Epoch 1/10
64/64 [==============================] - 1s 16ms/step - loss: 0.2019 - accura
cy: 0.9121 - val_loss: 0.2104 - val_accuracy: 0.9298
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1840 - accura
cy: 0.9277 - val_loss: 0.2069 - val_accuracy: 0.9474
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1915 - accura
cy: 0.9219 - val_loss: 0.1769 - val_accuracy: 0.9649
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1822 - accura
cy: 0.9238 - val_loss: 0.1090 - val_accuracy: 0.9649
Epoch 5/10
64/64 [==============================] - 1s 16ms/step - loss: 0.1923 - accura
cy: 0.9238 - val_loss: 0.1216 - val_accuracy: 0.9649
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1908 - accura
cy: 0.9316 - val_loss: 0.1075 - val_accuracy: 0.9649
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1693 - accura
cy: 0.9395 - val_loss: 0.1110 - val_accuracy: 0.9649
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1678 - accura
cy: 0.9316 - val_loss: 0.1257 - val_accuracy: 0.9825
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1720 - accura
cy: 0.9375 - val_loss: 0.1286 - val_accuracy: 0.9474
Epoch 10/10
64/64 [==============================] - 1s 14ms/step - loss: 0.1693 - accura
cy: 0.9414 - val_loss: 0.1326 - val_accuracy: 0.9474
****************************************************************************
***********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1635 - accura
cy: 0.9395 - val_loss: 0.3626 - val_accuracy: 0.8772
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1831 - accura
cy: 0.9277 - val_loss: 0.3327 - val_accuracy: 0.8947
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1677 - accura
cy: 0.9316 - val_loss: 0.2351 - val_accuracy: 0.9123
Epoch 4/10
64/64 [==============================] - 1s 14ms/step - loss: 0.1603 - accura
cy: 0.9336 - val_loss: 0.2359 - val_accuracy: 0.9474
Epoch 5/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1675 - accura
cy: 0.9316 - val_loss: 0.3012 - val_accuracy: 0.9298
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1513 - accura
cy: 0.9395 - val_loss: 0.2920 - val_accuracy: 0.9298
```

```
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1738 - accura
cy: 0.9277 - val_loss: 0.2759 - val_accuracy: 0.9298
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1522 - accura
cy: 0.9453 - val_loss: 0.2827 - val_accuracy: 0.9298
Epoch 9/10
64/64 [==============================] - 1s 14ms/step - loss: 0.1707 - accura
cy: 0.9316 - val_loss: 0.2190 - val_accuracy: 0.9649
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1744 - accura
cy: 0.9180 - val_loss: 0.2344 - val_accuracy: 0.9474
****************************************************************************
**********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1676 - accura
cy: 0.9258 - val_loss: 0.2037 - val_accuracy: 0.9298
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1861 - accura
cy: 0.9258 - val_loss: 0.2001 - val_accuracy: 0.9474
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1627 - accura
cy: 0.9355 - val_loss: 0.2029 - val_accuracy: 0.9298
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1667 - accura
cy: 0.9395 - val_loss: 0.2240 - val_accuracy: 0.9123
Epoch 5/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1604 - accura
cy: 0.9434 - val_loss: 0.2157 - val_accuracy: 0.9123
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1699 - accura
cy: 0.9414 - val_loss: 0.2422 - val_accuracy: 0.9123
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1987 - accura
cy: 0.9316 - val_loss: 0.2378 - val_accuracy: 0.9123
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1646 - accura
cy: 0.9492 - val_loss: 0.2165 - val_accuracy: 0.9123
Epoch 9/10
64/64 [==============================] - 1s 14ms/step - loss: 0.1866 - accura
cy: 0.9375 - val_loss: 0.2758 - val_accuracy: 0.9123
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2179 - accura
cy: 0.9375 - val_loss: 0.2483 - val_accuracy: 0.8947
****************************************************************************
**********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2129 - accura
cy: 0.9395 - val_loss: 0.1529 - val_accuracy: 0.9474
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2066 - accura
cy: 0.9414 - val_loss: 0.1470 - val_accuracy: 0.9474
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2414 - accura
cy: 0.9277 - val_loss: 0.1848 - val_accuracy: 0.9298
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2199 - accura
cy: 0.9297 - val_loss: 0.1057 - val_accuracy: 0.9825
Epoch 5/10
```

```
64/64 [==============================] - 1s 15ms/step - loss: 0.1854 - accura
cy: 0.9355 - val_loss: 0.1057 - val_accuracy: 0.9825
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1809 - accura
cy: 0.9551 - val_loss: 0.1102 - val_accuracy: 0.9649
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2096 - accura
cy: 0.9336 - val_loss: 0.1125 - val_accuracy: 0.9649
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1885 - accura
cy: 0.9414 - val_loss: 0.1413 - val_accuracy: 0.9649
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1840 - accura
cy: 0.9492 - val_loss: 0.0939 - val_accuracy: 0.9649
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1776 - accura
cy: 0.9492 - val_loss: 0.0966 - val_accuracy: 0.9649
******************************************************************************
***********************
Epoch 1/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1588 - accura
cy: 0.9492 - val_loss: 0.0905 - val_accuracy: 0.9649
Epoch 2/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1563 - accura
cy: 0.9512 - val_loss: 0.0881 - val_accuracy: 0.9474
Epoch 3/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1671 - accura
cy: 0.9492 - val_loss: 0.1140 - val_accuracy: 0.9474
Epoch 4/10
64/64 [==============================] - 1s 15ms/step - loss: 0.2109 - accura
cy: 0.9238 - val_loss: 0.1048 - val_accuracy: 0.9649
Epoch 5/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1595 - accura
cy: 0.9453 - val_loss: 0.0938 - val_accuracy: 0.9649
Epoch 6/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1598 - accura
cy: 0.9531 - val_loss: 0.0954 - val_accuracy: 0.9825
Epoch 7/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1864 - accura
cy: 0.9453 - val_loss: 0.0929 - val_accuracy: 0.9649
Epoch 8/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1632 - accura
cy: 0.9551 - val_loss: 0.1017 - val_accuracy: 0.9649
Epoch 9/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1546 - accura
cy: 0.9531 - val_loss: 0.0889 - val_accuracy: 0.9649
Epoch 10/10
64/64 [==============================] - 1s 15ms/step - loss: 0.1475 - accura
cy: 0.9551 - val_loss: 0.0781 - val_accuracy: 0.9649
******************************************************************************
***********************
Epoch 1/10
2021-12-04 16:58:11.170527: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
2021-12-04 16:58:11.313024: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
 7/65 [==>...........................] - ETA: 1s - loss: 0.0560 - accuracy:
1.0000
2021-12-04 16:58:11.390411: I tensorflow/core/grappler/optimizers/custom_grap
```

```
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
65/65 [==============================] - 2s 23ms/step - loss: 0.1538 - accura
cy: 0.9552 - val_loss: 0.1336 - val_accuracy: 0.9286
Epoch 2/10
 1/65 [..............................] - ETA: 1s - loss: 0.1620 - accuracy:
1.0000
2021-12-04 16:58:12.789349: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
2021-12-04 16:58:12.837932: I tensorflow/core/grappler/optimizers/custom_grap
h_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
65/65 [==============================] - 1s 19ms/step - loss: 0.1701 - accura
cy: 0.9513 - val_loss: 0.0907 - val_accuracy: 0.9821
Epoch 3/10
65/65 [==============================] - 1s 19ms/step - loss: 0.1622 - accura
cy: 0.9513 - val_loss: 0.0672 - val_accuracy: 1.0000
Epoch 4/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1448 - accura
cy: 0.9552 - val_loss: 0.0881 - val_accuracy: 0.9821
Epoch 5/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1416 - accura
cy: 0.9649 - val_loss: 0.0796 - val_accuracy: 0.9821
Epoch 6/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1351 - accura
cy: 0.9552 - val_loss: 0.0843 - val_accuracy: 0.9643
Epoch 7/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1481 - accura
cy: 0.9571 - val_loss: 0.1075 - val_accuracy: 0.9643
Epoch 8/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1487 - accura
cy: 0.9532 - val_loss: 0.1130 - val_accuracy: 0.9643
Epoch 9/10
65/65 [==============================] - 1s 18ms/step - loss: 0.1562 - accura
cy: 0.9532 - val_loss: 0.0835 - val_accuracy: 0.9821
Epoch 10/10
65/65 [==============================] - 1s 19ms/step - loss: 0.1422 - accura
cy: 0.9552 - val_loss: 0.0801 - val_accuracy: 0.9821
```

In [17]:

```
lstm_metrics_df
```

Out[17]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy
1	17.0	5.0	0.0	35.0	1.000000	0.772727	0.875000	0.912281
2	23.0	2.0	6.0	26.0	0.812500	0.920000	0.928571	0.859649
3	15.0	1.0	7.0	34.0	0.829268	0.937500	0.971429	0.859649
4	18.0	2.0	1.0	36.0	0.972973	0.900000	0.947368	0.947368
5	17.0	1.0	6.0	33.0	0.846154	0.944444	0.970588	0.877193
6	21.0	1.0	11.0	24.0	0.685714	0.954545	0.960000	0.789474
7	21.0	2.0	3.0	31.0	0.911765	0.913043	0.939394	0.912281
8	22.0	1.0	5.0	29.0	0.852941	0.956522	0.966667	0.894737
9	17.0	1.0	3.0	36.0	0.923077	0.944444	0.972973	0.929825

True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy

Cumulative metrics

In [29]:

```
all_dfs = [svm_metrics_df, kn_metrics_df, lstm_metrics_df]
all_names = ['SVM', 'KNN', 'LSTM']
```

1st Fold

In [34]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 1
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[34]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F'
SVM	22.0	0.0	0.0	35.0	1.0	1.000000	1.000000	1.000000	1.(
KNN	21.0	1.0	0.0	35.0	1.0	0.954545	0.972222	0.982456	0.!
LSTM	17.0	5.0	0.0	35.0	1.0	0.772727	0.875000	0.912281	0.9

2nd Fold

In [35]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 2
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[35]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F'
SVM	23.0	2.0	1.0	31.0	0.96875	0.92	0.939394	0.947368	0.9
KNN	21.0	4.0	0.0	32.0	1.00000	0.84	0.888889	0.929825	0.
LSTM	23.0	2.0	6.0	26.0	0.81250	0.92	0.928571	0.859649	0.8

3rd Fold

In [36]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 3
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[36]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	15.0	1.0	1.0	40.0	0.975610	0.9375	0.975610	0.964912	0.
KNN	14.0	2.0	1.0	40.0	0.975610	0.8750	0.952381	0.947368	0.9
LSTM	15.0	1.0	7.0	34.0	0.829268	0.9375	0.971429	0.859649	0.8

4th Fold

In [37]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 4
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[37]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	20.0	0.0	0.0	37.0	1.000000	1.00	1.000000	1.000000	1.(
KNN	19.0	1.0	0.0	37.0	1.000000	0.95	0.973684	0.982456	0.9
LSTM	18.0	2.0	1.0	36.0	0.972973	0.90	0.947368	0.947368	0.9

5th Fold

In [38]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 5
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[38]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	17.0	1.0	0.0	39.0	1.000000	0.944444	0.975000	0.982456	0.9

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1

6th Fold

In [39]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 6
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[39]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	19.0	3.0	0.0	35.0	1.000000	0.863636	0.921053	0.947368	0.9
KNN	19.0	3.0	0.0	35.0	1.000000	0.863636	0.921053	0.947368	0.9
LSTM	21.0	1.0	11.0	24.0	0.685714	0.954545	0.960000	0.789474	0.8

7th Fold

In [40]:

```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 7
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[40]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	22.0	1.0	1.0	33.0	0.970588	0.956522	0.970588	0.964912	0.9
KNN	22.0	1.0	0.0	34.0	1.000000	0.956522	0.971429	0.982456	0.9
LSTM	21.0	2.0	3.0	31.0	0.911765	0.913043	0.939394	0.912281	0.9

8th Fold

In [42]:

```python
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 8
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[42]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	23.0	0.0	2.0	32.0	0.941176	1.000000	1.000000	0.964912	0.9
KNN	23.0	0.0	0.0	34.0	1.000000	1.000000	1.000000	1.000000	1.0
LSTM	22.0	1.0	5.0	29.0	0.852941	0.956522	0.966667	0.894737	0.9

9th Fold

In [43]:

```python
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 9
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[43]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	18.0	0.0	0.0	39.0	1.000000	1.000000	1.000000	1.000000	1.0
KNN	18.0	0.0	0.0	39.0	1.000000	1.000000	1.000000	1.000000	1.0
LSTM	17.0	1.0	3.0	36.0	0.923077	0.944444	0.972973	0.929825	0.9

10th Fold

In [44]:

```python
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 10
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[44]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	25.0	0.0	0.0	31.0	1.00000	1.00	1.000000	1.000000	1.0

True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1

Average of all

In [45]:
```
# lstm_metrics_df.loc[1,:]
df = pd.DataFrame(columns=performance_metrics)
fold_count = 'Average'
for i, each_df in enumerate(all_dfs):
    temp_df = each_df.xs(fold_count)
    temp_df.name = all_names[i]
    df = df.append(temp_df)
df
```

Out[45]:

	True Negative	False Positive	False Negative	True Positivity	Sensitivity	Specificity	Precision	Accuracy	F1
SVM	20.4	0.8	0.5	35.2	0.985612	0.962210	0.978164	0.977193	0.9
KNN	19.7	1.5	0.2	35.5	0.994997	0.930415	0.959341	0.970113	0.9
LSTM	19.6	1.6	4.7	31.0	0.867310	0.924323	0.953199	0.889317	0.9

Observation

- I consider balanced accuracy to be the optimal metric to find the best model.
- The case being, SVM is the model which is giving the highest balanced accuracy.

Why is SVM performing better?

- SVM doesn't get affected by outliers
- It does not suffer from overfitting
- It is more efficient than other ML algorithms listed here