Iris Binary Classification – Random Forest, SVM, and LSTM

- Random Forest (ensemble tree-based model)
- Support Vector Machine (SVM) (classical ML model)
- LSTM (Long Short-Term Memory neural network)

Steps

- Load and preprocess the Iris data (iris.data).
- Convert the 3-class problem into a binary problem (Iris-setosa vs Iris-versicolor).
- Apply 10-fold cross-validation.
- Manually compute evaluation metrics: TP, TN, FP, FN, Accuracy, Precision, Recall, F1, and Error Rate for each model.

Imports

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

Loading and Inspecting the data

In [2]:
```
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width'
data = pd.read_csv('iris.data', header=None, names=column_names)
binary_data = data[data['class'].isin(['Iris-setosa', 'Iris-versicolor'])]
X = binary_data.iloc[:, :-1].values
y = LabelEncoder().fit_transform(binary_data['class'])
```

LSTM & ML

In [3]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_svm = X_scaled.copy()
X_lstm = X_scaled.reshape(X_scaled.shape[0], 1, X_scaled.shape[1])
print(f"SVM shape: {X_svm.shape}, LSTM shape: {X_lstm.shape}")
```

```
SVM shape: (100, 4), LSTM shape: (100, 1, 4)
```

Calculation Function

This function calculates confusion matrix based metrics required for analysis:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)
- Accuracy, Precision, Recall, F1-Score, Error Rate

In [8]:
```
def compute_metrics(y_true, y_pred):
    tp, fn = confusion_matrix[0][0], confusion_matrix[0][1]
    fp, tn = confusion_matrix[1][0], confusion_matrix[1][1]
    tpr = tp / (tp + fn)
    tnr = tn / (tn + fp)
    fpr = fp / (tn + fp)
    fnr = fn / (tp + fn)
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp)
    f1 = 2 * tp / (2 * tp + fp + fn)
    error_rate = (fp + fn) / (tp + tn + fp + fn)
    bacc = (tpr + tnr) / 2
    tss = tpr - fpr
    hss = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp

    return {
        'TP': tp,
        'TN': tn,
        'FP': fp,
        'FN': fn,
        'Accuracy': accuracy,
        'Precision': precision,
        'F1': f1,
        'Error Rate': error_rate,
        'BACC': bacc,
        'TSS': tss,
        'HSS': hss
    }
```

10-Fold Cross-Validation

Dataset is splitted into 10 folds, train and test each model on every fold

In [5]:
```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
metrics_rf = []
metrics_svm = []
metrics_lstm = []
```

Evaluate Random Forest, SVM and LSTM per fold

In [6]:
```
fold = 1

for train_idx, test_idx in kf.split(X_svm):
    print(f"\n===== Fold {fold} =====")

    X_train_svm, X_test_svm = X_svm[train_idx], X_svm[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    X_train_lstm, X_test_lstm = X_lstm[train_idx], X_lstm[test_idx]

    rf = RandomForestClassifier(random_state=42)
    rf.fit(X_train_svm, y_train)
    y_pred_rf = rf.predict(X_test_svm)
    m_rf = compute_metrics(y_test, y_pred_rf)
    metrics_rf.append(m_rf)
    print("Random Forest:", m_rf)

    svm = SVC(kernel='rbf', probability=True, random_state=42)
    svm.fit(X_train_svm, y_train)
    y_pred_svm = svm.predict(X_test_svm)
    m_svm = compute_metrics(y_test, y_pred_svm)
    metrics_svm.append(m_svm)
    print("SVM:", m_svm)

    lstm_model = Sequential([
        LSTM(16, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2]),
        Dropout(0.1),
        Dense(1, activation='sigmoid')
    ])

    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics
    lstm_model.fit(X_train_lstm, y_train, epochs=30, batch_size=8, verbose=0
    y_pred_prob_lstm = lstm_model.predict(X_test_lstm)
    y_pred_lstm = (y_pred_prob_lstm > 0.5).astype(int).reshape(-1)
    m_lstm = compute_metrics(y_test, y_pred_lstm)
    metrics_lstm.append(m_lstm)
    print("LSTM:", m_lstm)

    fold += 1
```

```
===== Fold 1 =====
Random Forest: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
LSTM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 2 =====
Random Forest: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
LSTM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 3 =====
Random Forest: {'TP': 5, 'TN': 5, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 5, 'TN': 5, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
LSTM: {'TP': 5, 'TN': 5, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 4 =====
Random Forest: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
LSTM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 5 =====
Random Forest: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.
make_predict_function.<locals>.one_step_on_data_distributed at 0x17c2ae020>
triggered tf.function retracing. Tracing is expensive and the excessive numb
er of tracings could be due to (1) creating @tf.function repeatedly in a loo
p, (2) passing tensors with different shapes, (3) passing Python objects ins
tead of tensors. For (1), please define your @tf.function outside of the loo
p. For (2), @tf.function has reduce_retracing=True option that can avoid unn
ecessary retracing. For (3), please refer to https://www.tensorflow.org/guid
e/function#controlling_retracing and https://www.tensorflow.org/api_docs/pyt
hon/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
LSTM: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 6 =====
Random Forest: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.
make_predict_function.<locals>.one_step_on_data_distributed at 0x17d6a9a80>
triggered tf.function retracing. Tracing is expensive and the excessive numb
er of tracings could be due to (1) creating @tf.function repeatedly in a loo
p, (2) passing tensors with different shapes, (3) passing Python objects ins
tead of tensors. For (1), please define your @tf.function outside of the loo
p. For (2), @tf.function has reduce_retracing=True option that can avoid unn
ecessary retracing. For (3), please refer to https://www.tensorflow.org/guid
e/function#controlling_retracing and https://www.tensorflow.org/api_docs/pyt
hon/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
LSTM: {'TP': 3, 'TN': 7, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 7 =====
Random Forest: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
LSTM: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 8 =====
Random Forest: {'TP': 7, 'TN': 3, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 7, 'TN': 3, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
LSTM: {'TP': 7, 'TN': 3, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 9 =====
Random Forest: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
LSTM: {'TP': 4, 'TN': 6, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}

===== Fold 10 =====
Random Forest: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precis
ion': 1.0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
SVM: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0,
'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:199:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
LSTM: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
LSTM: {'TP': 8, 'TN': 2, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.
0, 'Recall': 1.0, 'F1': 1.0, 'Error Rate': 0.0}
```

Summarization of Output

In [7]:
```
df_rf = pd.DataFrame(metrics_rf)
df_svm = pd.DataFrame(metrics_svm)
df_lstm = pd.DataFrame(metrics_lstm)

print("\n Per-fold metrics: Random Forest ")
print(df_rf)
print("\nAverage RF metrics:")
print(df_rf.mean())

print("\n Per-fold metrics: SVM ")
print(df_svm)
print("\nAverage SVM metrics:")
print(df_svm.mean())

print("\n Per-fold metrics: LSTM ")
print(df_lstm)
print("\n Average LSTM metrics: ")
print(df_lstm.mean())
```

```
 Per-fold metrics: Random Forest
   TP  TN  FP  FN  Accuracy  Precision  Recall   F1  Error Rate
0   4   6   0   0       1.0        1.0     1.0  1.0         0.0
1   4   6   0   0       1.0        1.0     1.0  1.0         0.0
2   5   5   0   0       1.0        1.0     1.0  1.0         0.0
3   4   6   0   0       1.0        1.0     1.0  1.0         0.0
4   3   7   0   0       1.0        1.0     1.0  1.0         0.0
5   3   7   0   0       1.0        1.0     1.0  1.0         0.0
6   8   2   0   0       1.0        1.0     1.0  1.0         0.0
7   7   3   0   0       1.0        1.0     1.0  1.0         0.0
8   4   6   0   0       1.0        1.0     1.0  1.0         0.0
9   8   2   0   0       1.0        1.0     1.0  1.0         0.0

Average RF metrics:
TP            5.0
TN            5.0
FP            0.0
FN            0.0
Accuracy      1.0
Precision     1.0
Recall        1.0
F1            1.0
Error Rate    0.0
dtype: float64

 Per-fold metrics: SVM
   TP  TN  FP  FN  Accuracy  Precision  Recall   F1  Error Rate
0   4   6   0   0       1.0        1.0     1.0  1.0         0.0
1   4   6   0   0       1.0        1.0     1.0  1.0         0.0
2   5   5   0   0       1.0        1.0     1.0  1.0         0.0
3   4   6   0   0       1.0        1.0     1.0  1.0         0.0
4   3   7   0   0       1.0        1.0     1.0  1.0         0.0
5   3   7   0   0       1.0        1.0     1.0  1.0         0.0
6   8   2   0   0       1.0        1.0     1.0  1.0         0.0
7   7   3   0   0       1.0        1.0     1.0  1.0         0.0
8   4   6   0   0       1.0        1.0     1.0  1.0         0.0
9   8   2   0   0       1.0        1.0     1.0  1.0         0.0

Average SVM metrics:
TP            5.0
TN            5.0
FP            0.0
FN            0.0
Accuracy      1.0
Precision     1.0
Recall        1.0
F1            1.0
Error Rate    0.0
dtype: float64

 Per-fold metrics: LSTM
   TP  TN  FP  FN  Accuracy  Precision  Recall   F1  Error Rate
0   4   6   0   0       1.0        1.0     1.0  1.0         0.0
1   4   6   0   0       1.0        1.0     1.0  1.0         0.0
2   5   5   0   0       1.0        1.0     1.0  1.0         0.0
3   4   6   0   0       1.0        1.0     1.0  1.0         0.0
```

```
4   3   7   0   0       1.0        1.0     1.0  1.0         0.0
5   3   7   0   0       1.0        1.0     1.0  1.0         0.0
6   8   2   0   0       1.0        1.0     1.0  1.0         0.0
7   7   3   0   0       1.0        1.0     1.0  1.0         0.0
8   4   6   0   0       1.0        1.0     1.0  1.0         0.0
9   8   2   0   0       1.0        1.0     1.0  1.0         0.0
```

```
 Average LSTM metrics:
TP            5.0
TN            5.0
FP            0.0
FN            0.0
Accuracy      1.0
Precision     1.0
Recall        1.0
F1            1.0
Error Rate    0.0
dtype: float64
```

Summary of the project

In this project, I applied three different machine learning algorithms—Random Forest, Support Vector Machine (SVM), and Long Short-Term Memory (LSTM) neural network—to perform binary classification on the Iris dataset. The dataset was adapted for binary classification by selecting only the classes Iris-setosa and Iris-versicolor.

I preprocessed the data by encoding labels and standardizing features, with special reshaping for the LSTM model to meet its input requirements. Using 10-fold cross-validation, each algorithm was trained and tested on different splits of the dataset to provide robust performance estimates.

Performance metrics including true positives, true negatives, false positives, false negatives, accuracy, precision, recall, F1-score, and error rate were computed manually to deeply understand each model's strengths and weaknesses.

The results showed that all three models performed well in distinguishing the two Iris classes, with some variation in precision and recall that could guide further tuning or model selection for practical applications.