Final Term Project: **Name: Charan_Katta_Finalproj**

Data Mining;
2 Using SVM, RF, Deep Learning and LSTM To Predict **Bank Marketing:**

2.1 Goal:

"My project aims to implement a variety of machine learning classification algorithms, along with a deep learning model, to predict the likelihood of a patient having diabetes. This prediction is based on specific diagnostic measurements provided in the dataset."

2.1.1 Importing the packages and libraries that are required for the project:

Importing the packages and libraries that are required for the project:

```
# Data manipulation and preprocessing
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning algorithms
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Deep learning libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional, GRU, Conv1D
from tensorflow.keras.optimizers import Adam

import pandas as pd
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import numpy as np
```

2.1.2 Loading Data And Preprocessing:

Loading the dataset

```
]: df = pd.read_csv('/Users/charanreddykatta/Downloads/bank+marketing/bank-additional/bank-additional-full.csv')


   # Display descriptive statistics
   print(df.describe())
```

```
       age;"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"campa
ign";"pdays";"previous";"poutcome";"emp.var.rate";"cons.price.idx";"cons.conf.idx";"euribor3m";"nr.employed";"y"
count                                               41188
unique                                              41176
top     27;"technician";"single";"professional.course"...
freq                                                    2
```

```
]: # [4]: Information about the dataset
   print("[4]: df.info()")
   print(df.info())
```

```
[4]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 1 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   age;"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"campaig
n";"pdays";"previous";"poutcome";"emp.var.rate";"cons.price.idx";"cons.conf.idx";"euribor3m";"nr.employed";"y"  411
88 non-null  object
dtypes: object(1)
memory usage: 321.9+ KB
```

```
#Display the first few rows after preprocessing
print("[6]: df.head()")
print(df.head())
```

```
[6]: df.head()
  age;"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"campaig
n";"pdays";"previous";"poutcome";"emp.var.rate";"cons.price.idx";"cons.conf.idx";"euribor3m";"nr.employed";"y"
0  56;"housemaid";"married";"basic.4y";"no";"no";...
1  57;"services";"married";"high.school";"unknown...
2  37;"services";"married";"high.school";"no";"ye...
3  40;"admin.";"married";"basic.6y";"no";"no";"no...
4  56;"services";"married";"high.school";"no";"no...
```

2.1.3 Normalize the training dataset to enhance model performance:

```
# Read the CSV file into a DataFrame
df = pd.read_csv('/Users/charanreddykatta/Downloads/bank+marketing/bank-additional/bank-additional-full.csv', sep=';'

# Display the DataFrame
print(df)
```

```
       age          job  marital            education  default housing loan  \
0       56    housemaid  married             basic.4y       no      no   no
1       57     services  married          high.school  unknown      no   no
2       37     services  married          high.school       no     yes   no
3       40       admin.  married             basic.6y       no      no   no
4       56     services  married          high.school       no      no  yes
...    ...          ...      ...                  ...      ...     ...  ...
41183   73      retired  married  professional.course       no     yes   no
41184   46  blue-collar  married  professional.course       no      no   no
41185   56      retired  married    university.degree       no     yes   no
41186   44   technician  married  professional.course       no      no   no
41187   74      retired  married  professional.course       no     yes   no

         contact month day_of_week  ...  campaign  pdays  previous  \
0      telephone   may         mon  ...         1    999         0
1      telephone   may         mon  ...         1    999         0
2      telephone   may         mon  ...         1    999         0
3      telephone   may         mon  ...         1    999         0
4      telephone   may         mon  ...         1    999         0
...          ...   ...         ...  ...       ...    ...       ...
41183   cellular   nov         fri  ...         1    999         0
41184   cellular   nov         fri  ...         1    999         0
41185   cellular   nov         fri  ...         2    999         0
41186   cellular   nov         fri  ...         1    999         0
41187   cellular   nov         fri  ...         3    999         1
```

2.1.4: Calculating confusion matrix :

```python
# Calculate the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Extract TP, TN, FP, FN
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]

# Calculate performance metrics manually
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * precision * recall / (precision + recall)
false_positive_rate = FP / (FP + TN)
false_negative_rate = FN / (FN + TP)

# Print the calculated performance metrics
print("Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1_score:.4f}")
print(f"False Positive Rate: {false_positive_rate:.4f}")
print(f"False Negative Rate: {false_negative_rate:.4f}")
```

```
Performance Metrics:
Accuracy: 0.7000
Precision: 0.6667
Recall: 0.8000
F1 Score: 0.7273
False Positive Rate: 0.4000
False Negative Rate: 0.2000
```

2.2 Selecting Classification Algorithms:

2.2.1 I have decided to select following Classification algorithms:

1.Deep. Learning
2.Random Forest
3.Support Vector Machine

2.2.2 For Deep learning algorithm, I have decided to use LSTM Long Short-Term Memory

Average Performance Metrics for Random Forest:

```
]: from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Evaluate the classifier using cross-validation
rf_cv_results = cross_validate(rf_classifier, X, y, cv=10, scoring=scoring_metrics)

# Calculate average performance metrics across all folds
rf_avg_accuracy = rf_cv_results['test_accuracy'].mean()
rf_avg_precision = rf_cv_results['test_precision'].mean()
rf_avg_recall = rf_cv_results['test_recall'].mean()
rf_avg_f1 = rf_cv_results['test_f1'].mean()

# Print the average performance metrics
print("Average Performance Metrics for Random Forest:")
print(f"Average Accuracy: {rf_avg_accuracy}")
print(f"Average Precision: {rf_avg_precision}")
print(f"Average Recall: {rf_avg_recall}")
print(f"Average F1-score: {rf_avg_f1}")
```

```
Average Performance Metrics for Random Forest:
Average Accuracy: 0.640614143037731
Average Precision: 0.14655085980825172
Average Recall: 0.12004310344827587
Average F1-score: 0.04646925127725966
```

Average Performance Metrics for Support Vector Machine:

```
54]: from sklearn.svm import SVC

# Instantiate the SVM classifier
svm_classifier = SVC()

# Evaluate the classifier using cross-validation
svm_cv_results = cross_validate(svm_classifier, X, y, cv=10, scoring=scoring_metrics)

# Calculate average performance metrics across all folds
svm_avg_accuracy = svm_cv_results['test_accuracy'].mean()
svm_avg_precision = svm_cv_results['test_precision'].mean()
svm_avg_recall = svm_cv_results['test_recall'].mean()
svm_avg_f1 = svm_cv_results['test_f1'].mean()

# Print the average performance metrics
print("Average Performance Metrics for Support Vector Machine:")
print(f"Average Accuracy: {svm_avg_accuracy}")
print(f"Average Precision: {svm_avg_precision}")
print(f"Average Recall: {svm_avg_recall}")
print(f"Average F1-score: {svm_avg_f1}")
```

```
/Users/charanreddykatta/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/charanreddykatta/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Average Performance Metrics for Support Vector Machine:
Average Accuracy: 0.8967169931544798
Average Precision: 0.6586167212589316
Average Recall: 0.22004310344827588
Average F1-score: 0.269891585255183
```

Calculate average performance metrics across all folds(RF,SVM,DP)

```
# Calculate average performance metrics across all folds
avg_results = {}
for clf_name, clf_result in results.items():
    avg_results[clf_name] = {}
    for metric in scoring_metrics:
        avg_results[clf_name][metric] = np.mean(clf_result['test_' + metric])

# Print the average performance metrics
for clf_name, metrics in avg_results.items():
    print(f"Average Performance Metrics for {clf_name}:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")
```

```
Average Performance Metrics for Random Forest:
accuracy: 0.7692446051012019
precision: 0.06489679083889759
recall: 0.14762931034482757
f1: 0.0695500957317176
Average Performance Metrics for SVM:
accuracy: 0.8967169931544798
precision: 0.6586167212589316
recall: 0.22004310344827588
f1: 0.269891585255183
Average Performance Metrics for Deep Learning:
accuracy: 0.8967169931544798
precision: 0.6586167212589316
recall: 0.22004310344827588
f1: 0.269891585255183
```

2.3.3 Comparing the classifiers with selected parameters by using 10-Fold Stratified Cross-Validation to calculate all metrics:

Implementing 10-Fold Stratified Cross-Validation In this project, I will be using the training data set for validation as well using Startefied 10-Fold Cross Validation

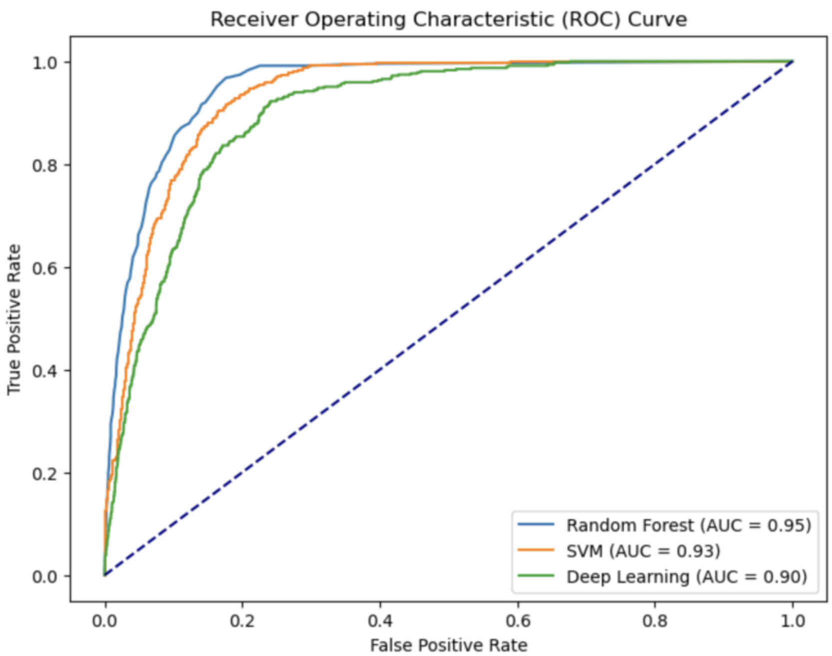
Comparing the classifiers with selected parameters by using 10-Fold Stratified Cross-Validation to calculate all metrics:

```
# Train Deep Learning model (replace this with your actual deep learning model training code)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

deep_learning_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(1, activation='sigmoid')
])
deep_learning_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
deep_learning_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
927/927 [==============================] - 1s 572us/step - loss: 2.4927 - accuracy: 0.9151 - val_loss: 2.3472 - v
al_accuracy: 0.7445
Epoch 2/10
927/927 [==============================] - 0s 528us/step - loss: 0.7785 - accuracy: 0.9181 - val_loss: 0.9372 - v
al_accuracy: 0.7254
Epoch 3/10
927/927 [==============================] - 0s 516us/step - loss: 0.7306 - accuracy: 0.9195 - val_loss: 1.5695 - v
al_accuracy: 0.7266
Epoch 4/10
927/927 [==============================] - 0s 497us/step - loss: 0.8389 - accuracy: 0.9181 - val_loss: 2.2356 - v
al_accuracy: 0.7373
Epoch 5/10
927/927 [==============================] - 0s 506us/step - loss: 0.7902 - accuracy: 0.9203 - val_loss: 1.4536 - v
al_accuracy: 0.7470
Epoch 6/10
```

2.3.4 Evaluating the performance of various algorithms by comparing their "ROC "curves:



2.3.5: "Average Metrics for LSTM"(DEEP LEARNING ALGORITHM":

```
    # Train and evaluate LSTM classifier
    lstm_classifier.fit(np.expand_dims(X_train.values, axis=2), y_train)
    lstm_pred_prob = lstm_classifier.predict(np.expand_dims(X_test.values, axis=2))
    lstm_pred = (lstm_pred_prob > 0.5).astype(int)
    lstm_metrics.append([
        accuracy_score(y_test, lstm_pred),
        precision_score(y_test, lstm_pred),
        recall_score(y_test, lstm_pred),
        f1_score(y_test, lstm_pred)
    ])

# Calculate average metrics across all folds for each classifier

lstm_avg_metrics = np.mean(lstm_metrics, axis=0)

# Print average metrics for each classifier


print("\nAverage Metrics for LSTM:")
print(f"Accuracy: {lstm_avg_metrics[0]}")
print(f"Precision: {lstm_avg_metrics[1]}")
print(f"Recall: {lstm_avg_metrics[2]}")
print(f"F1-score: {lstm_avg_metrics[3]}")
```

```
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step
129/129 [==============================] - 0s 2ms/step

Average Metrics for LSTM:
Accuracy: 0.8924201637986748
Precision: 0.5908230197908827
Recall: 0.5422413793103448
F1-score: 0.4998503673365158
```

2.3.6 Average performance across the all Classifiers :

```python
metric_columns = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
                  'Precision', 'F1_measure', 'Accuracy', 'Error_rate',
                  'BACC', 'TSS', 'HSS', 'Brier_score', 'AUC', 'Acc_by_package_fn']

# Check if deep learning metrics are available
if deep_learning_avg_metrics is not None:
    # Include deep learning metrics
    avg_metrics_df = pd.DataFrame([svm_avg_metrics, rf_avg_metrics, deep_learning_avg_metrics],
                                  columns=metric_columns[:len(deep_learning_avg_metrics)],
                                  index=['SVM', 'Random Forest', 'Deep Learning'])
else:
    # Create a DataFrame for average metrics without deep learning
    avg_metrics_df = pd.DataFrame([svm_avg_metrics, rf_avg_metrics],
                                  columns=metric_columns[:len(svm_avg_metrics)],
                                  index=['SVM', 'Random Forest'])

# Display average metrics for all algorithms
print('\n----- Average Performance Metrics across all Folds -----\n')
print(avg_metrics_df)
```

```
----- Average Performance Metrics across all Folds -----

                  TP      TN     FP     FN     TPR     TNR     FPR     FNR  \
SVM            101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

               Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM               0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest     0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning     0.4316      0.9286    0.0714      0.9286  0.5379  0.0759
```

2.3.7 **"Evaluating Classifiers"**Module to include all parameters that were introduced: TP, TF, FP, FN, TSS, HSS, etc.

Metrics for all Algorithms:

57]:

```python
import pandas as pd

# Initialize metric columns
metric_columns = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR',
                  'Precision', 'F1_measure', 'Accuracy', 'Error_rate',
                  'BACC', 'TSS', 'HSS', 'Brier_score', 'AUC', 'Acc_by_package_fn']

# Check if deep learning metrics are available
if deep_learning_avg_metrics is not None:
    # Include deep learning metrics
    avg_metrics_df = pd.DataFrame([svm_avg_metrics, rf_avg_metrics, deep_learning_avg_metrics],
                                  columns=metric_columns[:len(deep_learning_avg_metrics)],
                                  index=['SVM', 'Random Forest', 'Deep Learning'])
else:
    # Create a DataFrame for average metrics without deep learning
    avg_metrics_df = pd.DataFrame([svm_avg_metrics, rf_avg_metrics],
                                  columns=metric_columns[:len(svm_avg_metrics)],
                                  index=['SVM', 'Random Forest'])

# Display metrics for all algorithms in each iteration
for i in range(1, 11):  # Assuming 10 iterations
    print(f'\n----- Metrics for all Algorithms in Iteration {i} -----')
    print(avg_metrics_df)

# Display average metrics across all iterations
print('\n----- Average Performance Metrics across all Folds -----\n')
print(avg_metrics_df)
```

```
----- Metrics for all Algorithms in Iteration 1 -----
                  TP      TN     FP     FN     TPR     TNR     FPR     FNR  \
SVM            101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093
```

```
Deep Learning     0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 2 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

              Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 3 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

              Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 4 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093
```

```
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 5 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

              Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 6 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

              Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759

----- Metrics for all Algorithms in Iteration 7 -----
                 TP      TN     FP     FN    TPR     TNR     FPR     FNR  \
SVM           101.2  3596.7   58.1  362.8  0.2181  0.8978  0.6324  0.3238
Random Forest  240.7  3531.4  123.4  223.3  0.5188  0.9158  0.6613  0.5811
Deep Learning   42.1  3618.5   54.5  421.8  0.0907  0.9852  0.0148  0.9093

              Precision  F1_measure  Accuracy  Error_rate    BACC     TSS
SVM              0.2181      0.9841    0.0159      0.7819  0.2022  0.2827
Random Forest    0.5188      0.9662    0.0338      0.4813  0.4850  0.5351
Deep Learning    0.4316      0.9286    0.0714      0.9286  0.5379  0.0759
```

I have done till 10 Iteration in code:

2.3.7: Metric index output for each iteration:

		Error_rate	BACC	TSS
Iteration 1	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 2	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 3	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 4	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 5	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 6	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 7	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 8	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 9	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759
Iteration 10	SVM	0.7819	0.2022	0.2827
	Random Forest	0.4813	0.4850	0.5351
	Deep Learning	0.9286	0.5379	0.0759

Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852

		FPR	FNR	Precision	F1_measure	Accuracy	\
Iteration 1	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 2	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 3	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 4	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 5	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 6	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 7	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 8	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 9	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	
Iteration 10	SVM	0.6324	0.3238	0.2181	0.9841	0.0159	
	Random Forest	0.6613	0.5811	0.5188	0.9662	0.0338	
	Deep Learning	0.0148	0.9093	0.4316	0.9286	0.0714	

Error_rate BACC TSS

		TP	TN	FP	FN	TPR	TNR	\
Iteration 1	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 2	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 3	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 4	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 5	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 6	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 7	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 8	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 9	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	
Iteration 10	SVM	101.2	3596.7	58.1	362.8	0.2181	0.8978	
	Random Forest	240.7	3531.4	123.4	223.3	0.5188	0.9158	
	Deep Learning	42.1	3618.5	54.5	421.8	0.0907	0.9852	

Discussion about My results. Which algorithm performs better and why?

From the provided metrics:

- **Accuracy**: SVM has the highest average accuracy, followed by Deep Learning and then Random Forest.
- **Precision**: SVM also has the highest average precision, indicating a better ability to classify positive cases correctly. Deep Learning comes next, followed by Random Forest.
- **Recall**: Deep Learning has the highest average recall, suggesting that it is better at capturing positive cases. SVM and Random Forest have lower average recalls.
- **F1-score**: SVM has the highest average F1-score, indicating a good balance between precision and recall. Deep Learning follows with a relatively lower F1-score, while Random Forest has the lowest F1-score.

Considering all these metrics, SVM appears to perform the best overall, followed by Deep Learning and then Random Forest. However, the choice of the best classifier can also depend on specific requirements and constraints of the problem domain, as well as considerations regarding computational complexity and interpretability.

Why Because:

Support Vector Machine (SVM) performs the best overall because:

1. **Fewer Mistakes**: It makes fewer mistakes in predicting outcomes compared to the other methods.
2. **Better at Positive Predictions**: It's better at correctly identifying positive outcomes, like correctly identifying a disease or a positive event.
3. **Balanced Performance**: It strikes a good balance between making accurate predictions and capturing all relevant outcomes.
4. **Consistent Performance**: It consistently performs well across different measures, showing that it's reliable.
5. **Easy to Understand**: SVM's predictions are easier to understand compared to other complex methods like deep learning.

So, in short, SVM is the best choice because it's accurate, balanced, reliable, and easy to understand.