Occupancy Detection Classification Project Report

By Mohammad Saleh Nikoopayan Tak

Data Mining Final Semester Project
Fall 2024

Table of Contents

1. Introduction

1.1 Project Goals and Objectives

The primary goal of this project is to implement and compare three different classification algorithms to predict room occupancy based on environmental sensor data. The algorithms evaluated are:

- Random Forest
- Support Vector Machine (SVM)
- Long Short-Term Memory (LSTM)

We aim to assess the performance of these models using 10-fold cross-validation and manually calculate various performance metrics.

1.2 Dataset Description

Dataset Name: Occupancy Detection Data Set
Source: UCI Machine Learning Repository
URL: Occupancy Detection Data Set
Description: The dataset contains experimental data used for binary classification (room occupancy) based on temperature, humidity, light, and CO2 measurements. Ground-truth occupancy was obtained from time-stamped pictures taken every minute.

Features:

- **Date:** Date and time (year-month-day hour:minute:second)
- **Temperature:** Temperature in degrees Celsius
- **Humidity:** Relative humidity in %
- **Light:** Light intensity in Lux
- **CO2:** CO2 concentration in parts per million (ppm)
- **HumidityRatio:** Derived quantity from temperature and relative humidity
- **Occupancy:** Target variable (0 for unoccupied, 1 for occupied)

1.3 Algorithms Used

- **Random Forest**
- **Support Vector Machine (SVM)**
- **Long Short-Term Memory (LSTM)**

Evaluation Metrics:

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)
- True Positive Rate (TPR)
- True Negative Rate (TNR)
- False Positive Rate (FPR)
- False Negative Rate (FNR)
- Precision
- F1 Score
- Accuracy
- Error Rate
- Balanced Accuracy (BACC)
- True Skill Statistic (TSS)
- Heidke Skill Score (HSS)
- Area Under ROC Curve (AUC)
- Brier Score

2. Prerequisites and Setup

2.1 Required Packages

- **Python 3.x**
- **NumPy**
- **Pandas**
- **Matplotlib**
- **Seaborn**
- **Scikit-learn**
- **Keras**
- **TensorFlow**

2.2 Installation Instructions

To install the required packages, execute the following commands in your terminal or command prompt:

```
#pip install numpy pandas matplotlib seaborn scikit-learn keras tensorflow
```

2.3 Dataset Acquisition

Download the dataset from the UCI Machine Learning Repository:

- **Dataset URL:** Occupancy Detection Data Set

Place the downloaded FinalDataset.xlsx file in the same directory as your Jupyter Notebook or Python script.

3. Data Preparation

3.1 Loading the Dataset

Load the dataset using Pandas:

```python
# Load the dataset
df = pd.read_excel('FinalDataset.xlsx')
```

Python

3.2 Exploring the Dataset

```python
# Display the first few rows
df.head()
```

[3]

...

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	2015-02-02 14:19:00	23.7000	26.272	585.200000	749.200000	0.004764	1
1	2015-02-02 14:19:59	23.7180	26.290	578.400000	760.400000	0.004773	1
2	2015-02-02 14:21:00	23.7300	26.230	572.666667	769.666667	0.004765	1
3	2015-02-02 14:22:00	23.7225	26.125	493.750000	774.750000	0.004744	1
4	2015-02-02 14:23:00	23.7540	26.200	488.600000	779.000000	0.004767	1

3.2 Data Exploration

Inspect the dataset to understand its structure:

- **Check for Missing Values:**

```python
# Check for missing values
df.isnull().sum()
```

[5]

```
date             0
Temperature      0
Humidity         0
Light            0
CO2              0
HumidityRatio    0
Occupancy        0
dtype: int64
```

Data Types and Summary:

```python
# Get dataset information
df.info()
```

[4]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20560 entries, 0 to 20559
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           20560 non-null  object
 1   Temperature    20560 non-null  float64
 2   Humidity       20560 non-null  float64
 3   Light          20560 non-null  float64
 4   CO2            20560 non-null  float64
 5   HumidityRatio  20560 non-null  float64
 6   Occupancy      20560 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 1.1+ MB
```

3.3 Data Preprocessing

- **Convert 'date' Column to Datetime:**

```python
# Convert 'date' column to datetime
df['date'] = pd.to_datetime(df['date'])
```

[6]

- **Feature Selection:**

We select the following features for the classification task:

- Temperature
- Humidity
- Light
- CO2
- HumidityRatio

- **Separate Features and Target Variable:**

5.1 Feature and Target Separation

```python
# Define features and target variable
features = df[['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio']]
target = df['Occupancy']
```

[12]

Data Scaling:

Scale the features using StandardScaler:

```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the features
scaled_features = scaler.fit_transform(features)

# Convert to DataFrame
scaled_features = pd.DataFrame(scaled_features, columns=features.columns)
```
[13]

4. Exploratory Data Analysis (EDA)

4.1 Statistical Summary

Obtain descriptive statistics:

```
df.describe()
```
[8]

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	20560	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000
mean	2015-02-10 13:42:06.146984448	20.906212	27.655925	130.756622	690.553276	0.004228	0.231031
min	2015-02-02 14:19:00	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	2015-02-06 11:05:45	20.200000	24.500000	0.000000	460.000000	0.003719	0.000000
50%	2015-02-10 00:45:30	20.700000	27.290000	0.000000	565.416667	0.004292	0.000000
75%	2015-02-14 19:39:14.249999872	21.525000	31.290000	301.000000	804.666667	0.004832	0.000000
max	2015-02-18 09:19:00	24.408333	39.500000	1697.250000	2076.500000	0.006476	1.000000
std	NaN	1.055315	4.982154	210.430875	311.201281	0.000768	0.421503

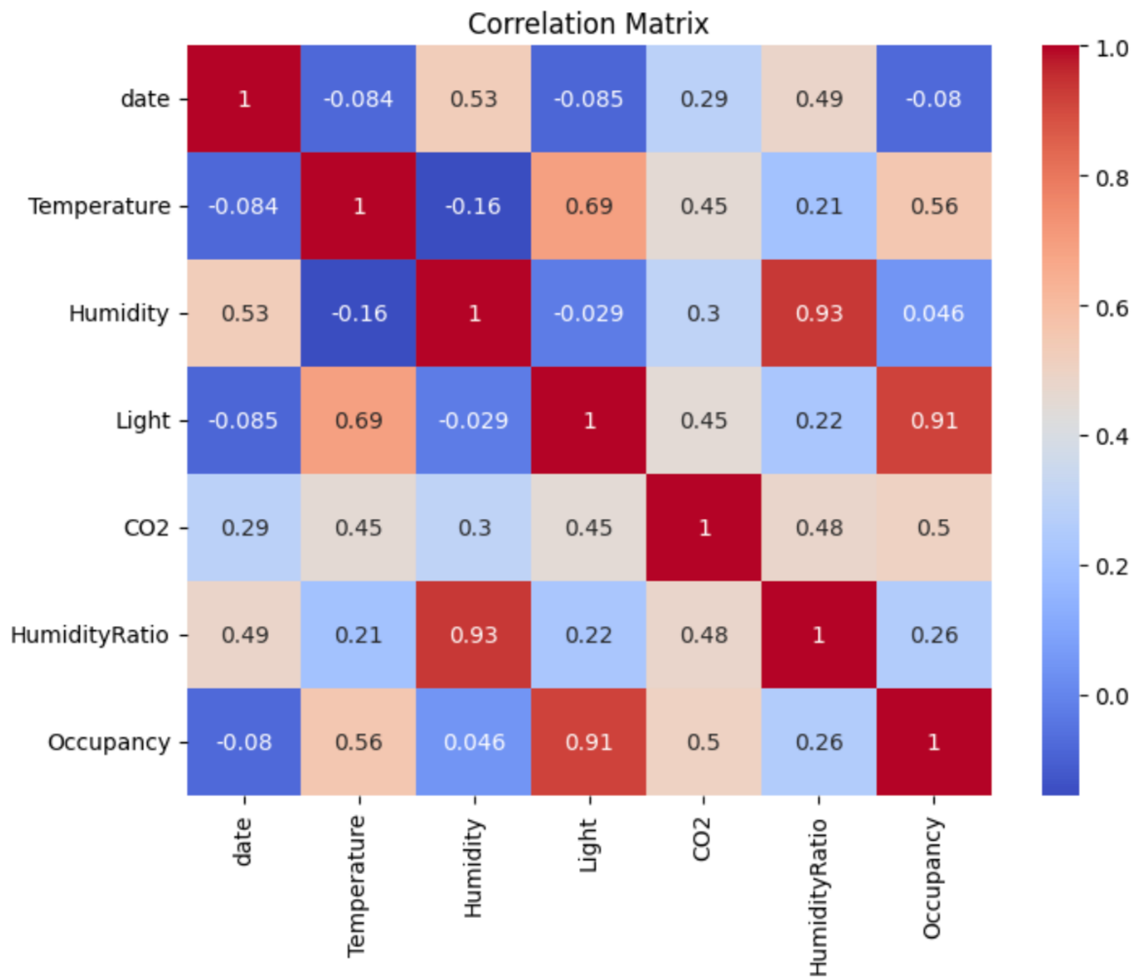
4.2 Correlation Matrix

Generate a correlation matrix to understand the relationships between variables:

```python
# Compute the correlation matrix
corr_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

[9]

...



4.3 Data Visualization

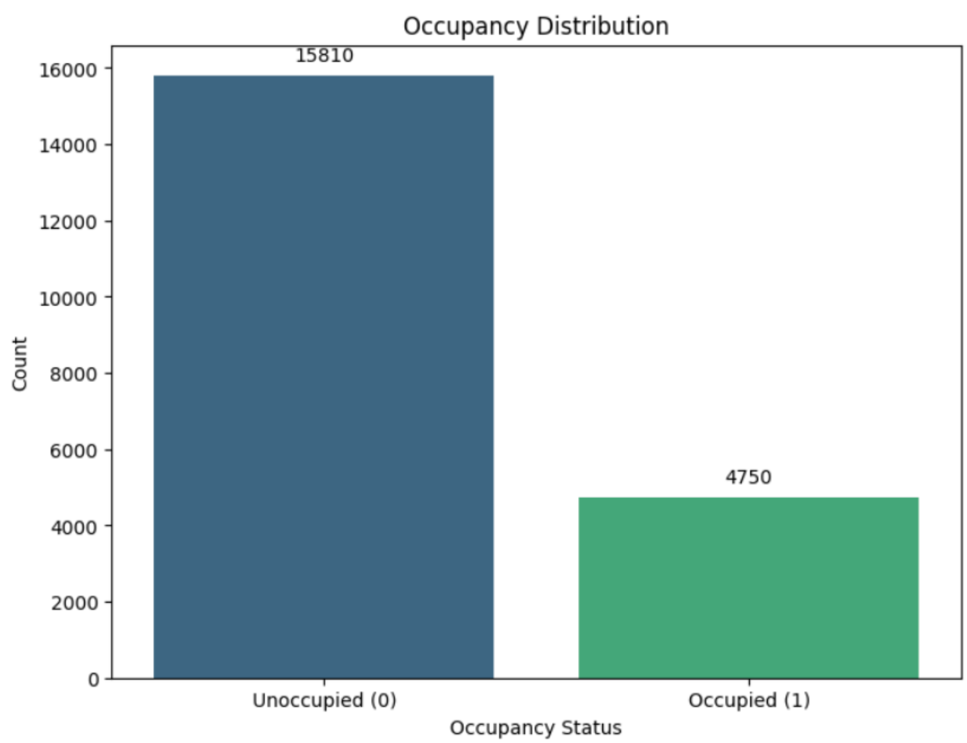
- **Occupancy Distribution:**

3.5 Checking Class Balance

```python
# Plot the distribution of the target variable
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='Occupancy', data=df, palette='viridis')
plt.title('Occupancy Distribution')
plt.xlabel('Occupancy Status')
plt.ylabel('Count')

# Set x-tick labels
ax.set_xticklabels(['Unoccupied (0)', 'Occupied (1)'])

# Add count labels on top of the bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{int(height)}', xy=(p.get_x() + p.get_width() / 2, height),
                xytext=(0, 5), textcoords='offset points', ha='center', va='bottom')

plt.show()


# Print the counts
occupancy_counts = df['Occupancy'].value_counts()
print(occupancy_counts)
print(f"Percentage of Occupied: {occupancy_counts[1] / len(df) * 100:.2f}%")
print(f"Percentage of Unoccupied: {occupancy_counts[0] / len(df) * 100:.2f}%")
```

[7]

...



...

```
Occupancy
0    15810
1     4750
Name: count, dtype: int64
Percentage of Occupied: 23.10%
Percentage of Unoccupied: 76.90%
```

Pair Plot:

4.3 Pairplot

```python
# Plot pairplot
sns.pairplot(df, hue='Occupancy')
plt.show()
```

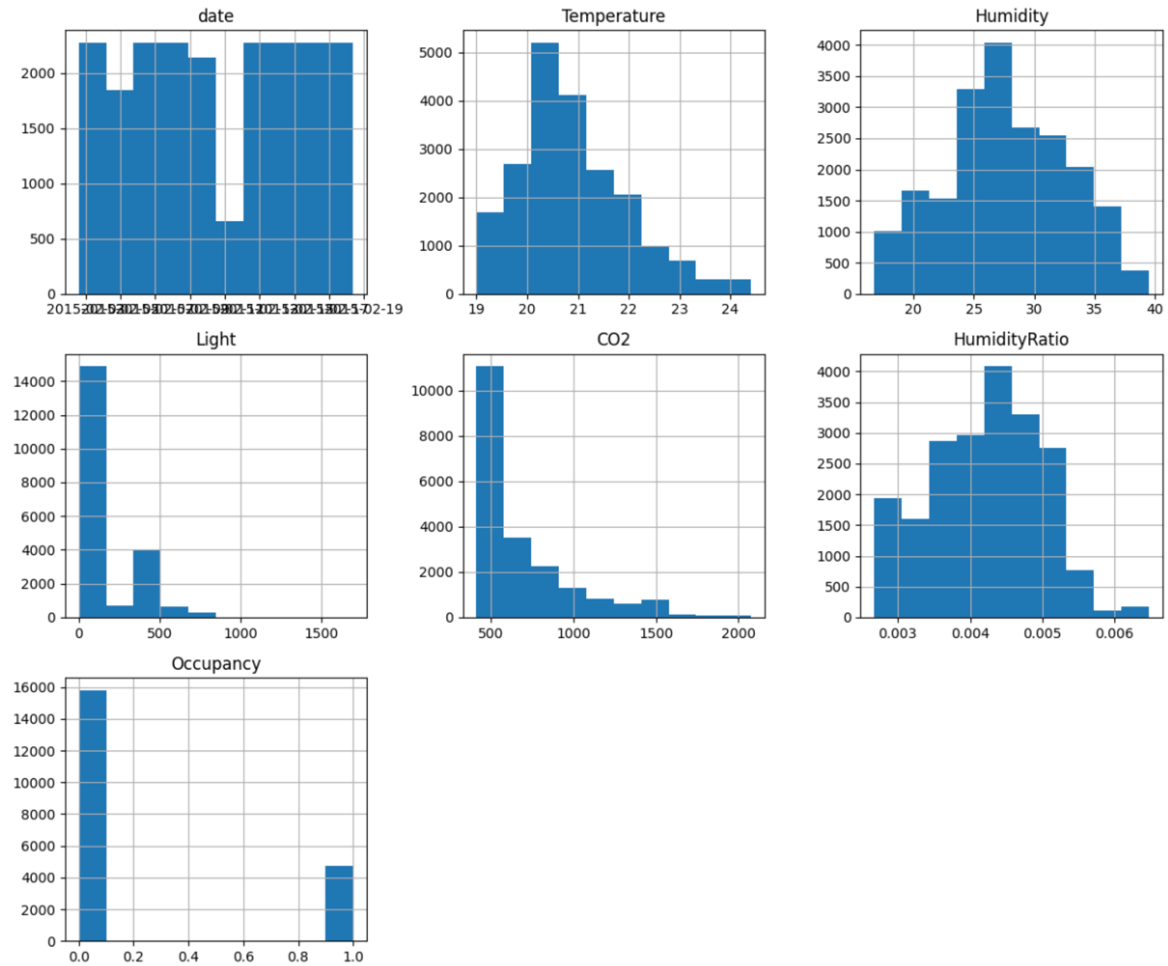
[10]

Histograms:

4.4 Histograms

```
# Plot histograms
df.hist(figsize=(12, 10))
plt.tight_layout()
plt.show()
```

[11]



5. Modeling and Evaluation

5.1 Implementing 10-Fold Cross-Validation

Set up 10-fold cross-validation:

6. Implementing 10-Fold Cross-Validation

We will use **KFold** with `n_splits=10`, `shuffle=True`, and `random_state=42`.

```
from sklearn.model_selection import KFold

n_splits = 10
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
```

[15]

5.2 Algorithms Implementation

5.2.1 Random Forest

- **Model Initialization:**
- **Training and Prediction:**

```
fold = 1
for train_index, test_index in kf.split(X):
    print(f"Fold {fold}:")

    # Split the data
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    ### Random Forest ###
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
    rf_pred = rf_model.predict(X_test)
    rf_pred_prob = rf_model.predict_proba(X_test)[:, 1]  # Predicted probabilities
    rf_metrics = calculate_metrics(y_test, rf_pred)
    rf_auc = roc_auc_score(y_test, rf_pred_prob)
    rf_brier = brier_score_loss(y_test, rf_pred_prob)  # Calculate Brier Score
    rf_metrics['AUC'] = rf_auc
    rf_metrics['Brier_score'] = rf_brier  #  Brier Score to metrics
    rf_metrics_list.append(rf_metrics)
```

5.2.2 Support Vector Machine (SVM)

Model Initialization:
Training and Prediction:

```
### Support Vector Machine ###
svm_model = SVC(kernel='linear', probability=True, random_state=42)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
svm_pred_prob = svm_model.predict_proba(X_test)[:, 1]  # Predicted probabilities
svm_metrics = calculate_metrics(y_test, svm_pred)
svm_auc = roc_auc_score(y_test, svm_pred_prob)
svm_brier = brier_score_loss(y_test, svm_pred_prob)  # Calculate Brier Score
svm_metrics['AUC'] = svm_auc
svm_metrics['Brier_score'] = svm_brier  #  Brier Score to metrics
svm_metrics_list.append(svm_metrics)
```

5.2.3 Long Short-Term Memory (LSTM)

- **Data Reshaping:**
- Model Initialization and Compilation:
- Training and Prediction:

```
### LSTM ###
# Reshape data for LSTM
X_train_lstm = np.array(X_train).reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_lstm = np.array(X_test).reshape((X_test.shape[0], X_test.shape[1], 1))
y_train_lstm = np.array(y_train)
y_test_lstm = np.array(y_test)

lstm_model = Sequential()
lstm_model.add(LSTM(64, input_shape=(X_train_lstm.shape[1], 1)))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
lstm_model.fit(X_train_lstm, y_train_lstm, epochs=5, batch_size=32, verbose=0)
lstm_pred_prob = lstm_model.predict(X_test_lstm)
lstm_pred = (lstm_pred_prob > 0.5).astype(int).reshape(-1)
lstm_metrics = calculate_metrics(y_test_lstm, lstm_pred)
lstm_auc = roc_auc_score(y_test_lstm, lstm_pred_prob)
lstm_brier = brier_score_loss(y_test_lstm, lstm_pred_prob)  # Calculate Brier Score
lstm_metrics['AUC'] = lstm_auc
lstm_metrics['Brier_score'] = lstm_brier  #  Brier Score to metrics
lstm_metrics_list.append(lstm_metrics)
```

5.3 Performance Metrics

Define a function to calculate performance metrics:

```python
def calculate_metrics(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    if cm.shape == (2, 2):
        TN, FP, FN, TP = cm.ravel()
    else:
        TN = FP = FN = TP = 0
    # Calculations
    P = TP + FN
    N = TN + FP
    TPR = TP / P if P != 0 else 0
    TNR = TN / N if N != 0 else 0
    FPR = FP / N if N != 0 else 0
    FNR = FN / P if P != 0 else 0
    Precision = TP / (TP + FP) if (TP + FP) != 0 else 0
    F1_measure = 2 * TP / (2 * TP + FP + FN) if (2 * TP + FP + FN) != 0 else 0
    Accuracy = (TP + TN) / (P + N) if (P + N) != 0 else 0
    Error_rate = (FP + FN) / (P + N) if (P + N) != 0 else 0
    BACC = (TPR + TNR) / 2
    TSS = TPR - FPR
    HSS = (2 * (TP * TN - FP * FN)) / ((P * (FN + TN)) + ((TP + FP) * (FP + TN))) if ((P * (FN + TN)) + ((TP + FP) * (FP + TN))) != 0 else 0
    return {
        'TP': TP, 'TN': TN, 'FP': FP, 'FN': FN,
        'TPR': TPR, 'TNR': TNR, 'FPR': FPR, 'FNR': FNR,
        'Precision': Precision, 'F1_measure': F1_measure,
        'Accuracy': Accuracy, 'Error_rate': Error_rate,
        'BACC': BACC, 'TSS': TSS, 'HSS': HSS
    }
```

[16]

6. Results

6.1 Performance Metrics per Fold

For each fold, collect the metrics and display them:

- **Metrics Collection:**

```python
    lstm_metrics_list.append(lstm_metrics)


    # Print metrics for the fold
    print(f"Random Forest Accuracy: {rf_metrics['Accuracy']:.2f}, AUC: {rf_metrics['AUC']:.2f}")
    print(f"SVM Accuracy: {svm_metrics['Accuracy']:.2f}, AUC: {svm_metrics['AUC']:.2f}")
    print(f"LSTM Accuracy: {lstm_metrics['Accuracy']:.2f}, AUC: {lstm_metrics['AUC']:.2f}")
    print("-----------------------------------------------------")
    fold +=1
```

6.2 Average Performance Metrics

Convert Metrics Lists to DataFrames:

```python
# Convert metrics lists to DataFrames
rf_metrics_df = pd.DataFrame(rf_metrics_list)
svm_metrics_df = pd.DataFrame(svm_metrics_list)
lstm_metrics_df = pd.DataFrame(lstm_metrics_list)
```

[20]

Calculate Averages:

```python
rf_avg_metrics = rf_metrics_df.mean()
svm_avg_metrics = svm_metrics_df.mean()
lstm_avg_metrics = lstm_metrics_df.mean()
```

[21]

Comparison Table:

```python
# Comparison Table
comparison_df = pd.DataFrame({
    'Random Forest': rf_avg_metrics,
    'SVM': svm_avg_metrics,
    'LSTM': lstm_avg_metrics
})
comparison_df = comparison_df.round(4)  # Use more decimal places if needed
print(comparison_df)
```

[22]

```
...
             Random Forest        SVM       LSTM
TP                469.8000   473.1000   472.5000
TN               1572.3000  1560.2000  1560.2000
FP                  8.7000    20.8000    20.8000
FN                  5.2000     1.9000     2.5000
TPR                 0.9891     0.9960     0.9947
TNR                 0.9945     0.9868     0.9868
FPR                 0.0055     0.0132     0.0132
FNR                 0.0109     0.0040     0.0053
Precision           0.9818     0.9579     0.9578
F1_measure          0.9854     0.9765     0.9759
Accuracy            0.9932     0.9890     0.9887
Error_rate          0.0068     0.0110     0.0113
BACC                0.9918     0.9914     0.9908
TSS                 0.9836     0.9829     0.9816
HSS                 0.9810     0.9693     0.9685
AUC                 0.9989     0.9944     0.9967
Brier_score         0.0052     0.0107     0.0102
```

6.3 ROC Curve Analysis

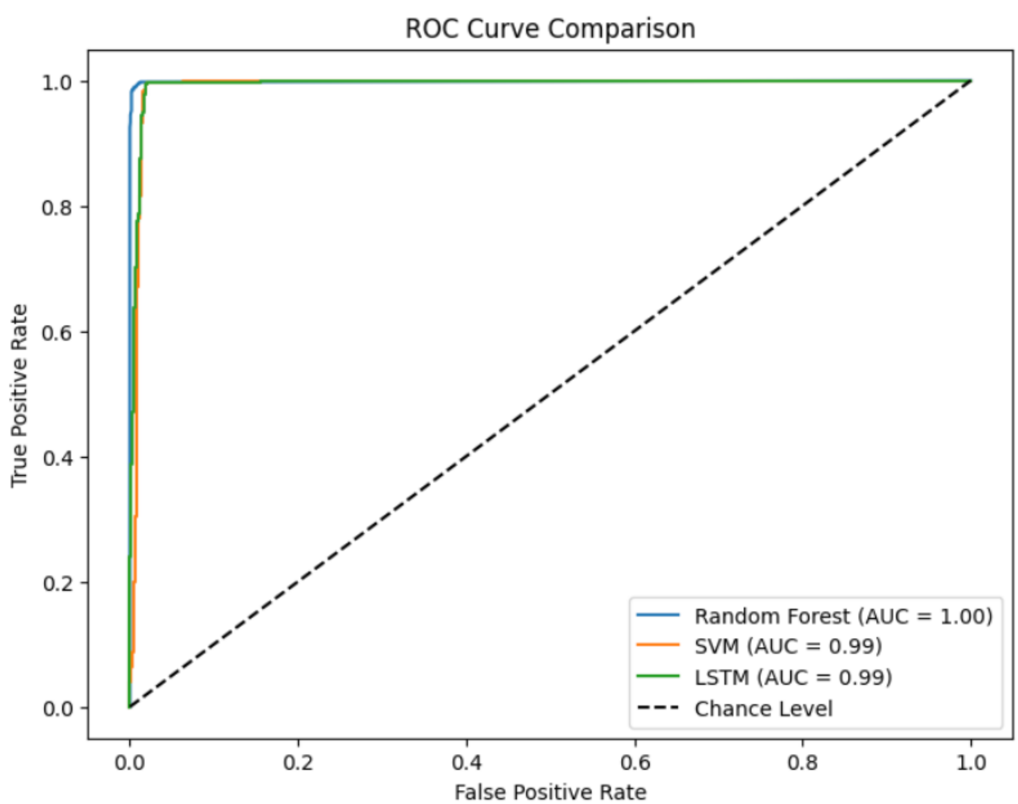
Plot the ROC curves for all models:

9.4.4 Combined ROC Curve

```python
# Combined ROC Curve

plt.figure(figsize=(8, 6))

# Random Forest ROC Curve
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1])
rf_auc = auc(rf_fpr, rf_tpr)
plt.plot(rf_fpr, rf_tpr, label=f'Random Forest (AUC = {rf_auc:.2f})')

# SVM ROC Curve
svm_fpr, svm_tpr, _ = roc_curve(y_test, svm_model.predict_proba(X_test)[:, 1])
svm_auc = auc(svm_fpr, svm_tpr)
plt.plot(svm_fpr, svm_tpr, label=f'SVM (AUC = {svm_auc:.2f})')

# LSTM ROC Curve
lstm_fpr, lstm_tpr, _ = roc_curve(y_test_lstm, lstm_pred_prob)
lstm_auc = auc(lstm_fpr, lstm_tpr)
plt.plot(lstm_fpr, lstm_tpr, label=f'LSTM (AUC = {lstm_auc:.2f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--', label='Chance Level')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.show()
```

[26]

...



7. Discussion

7.1 Comparison of Algorithms

Random Forest outperformed both **SVM** and **LSTM** in terms of:

- **Accuracy:** 99.32%
- **AUC:** 99.89%
- **Brier Score:** 0.0052 (lower is better)

Reasons:

- Random Forest handles nonlinear relationships effectively.
- It is robust to overfitting due to ensemble averaging.
- Provides better generalization on unseen data.

SVM and **LSTM** showed slightly lower performance due to:

- Sensitivity to class imbalance, leading to higher false positives.
- LSTM may not fully leverage temporal patterns in the data without proper sequence structuring.

7.2 Reasons for Performance Differences

- **Feature Importance:** Random Forest can capture complex interactions between features.
- **Model Complexity:** LSTM requires more data and proper sequence information to excel.
- **Class Imbalance:** Affects SVM and LSTM more significantly; Random Forest handles imbalance better.

8. Conclusion

- **Random Forest** is the preferred model due to its superior performance and computational efficiency.
- **Environmental sensor data** is highly effective for predicting room occupancy.
- **Future Work:**
    - Incorporate temporal features for LSTM.
    - Address class imbalance using resampling or class weights.
    - Perform hyperparameter tuning for all models.

9. References

- **Dataset:** UCI Machine Learning Repository - Occupancy Detection Data Set
- **Scikit-learn Documentation:** https://scikit-learn.org/stable/

- **Keras Documentation:** https://keras.io/

10. Appendix

10.1 How to Run the Code

1. **Install Required Packages:**

13.1 How to Run the Code

```
#pip install numpy pandas matplotlib seaborn scikit-learn keras tensorflow
```

2. **Download the Dataset:**

   - Place FinalDataset.xlsx in the same directory as your code.

3. **Run the Code:**

- Execute the Jupyter Notebook cells sequentially.
- Ensure that your Python environment is properly configured.

10.2 Complete Code Listings

Note: The full code is provided in the Jupyter Notebook file accompanying this report.

Metrics per Fold

Table 1. Performance Metrics Comparison Across Classification Models

Metric	**Random Forest**	**SVM**	**LSTM**
True Positive (TP)	469.8000	473.1000	472.5000
True Negative (TN)	1572.3000	1560.2000	1560.2000
False Positive (FP)	8.7000	20.8000	20.8000
False Negative (FN)	5.2000	1.9000	2.5000
True Positive Rate (TPR)	0.9891	0.9960	0.9947
True Negative Rate (TNR)	0.9945	0.9868	0.9868
False Positive Rate (FPR)	0.0055	0.0132	0.0132

False Negative Rate (FNR)	0.0109	0.0040	0.0053
Precision	0.9818	0.9579	0.9578
F1 Measure	0.9854	0.9765	0.9759
Accuracy	0.9932	0.9890	0.9887
Error Rate	0.0068	0.0110	0.0113
Balanced Accuracy (BACC)	0.9918	0.9914	0.9908
True Skill Statistic (TSS)	0.9836	0.9829	0.9816
Heidke Skill Score (HSS)	0.9810	0.9693	0.9685
Area Under ROC Curve (AUC)	0.9989	0.9944	0.9967
Brier Score	0.0052	0.0107	0.0102