Predicting the occupancy of a room

Kalle Kankaanpää

October 7, 2022

1 Introduction

Knowing whether someone is currently using a room or space is beneficial. With that knowledge IOT automation systems can make more ecological decisions, like shutting the lights off or turning down the air conditioning if the room is empty. The simple and "dumb" motion sensors used to control lights aren't always enough. For example, working in a space where the lights get turned off time after time because a movement sensor can't see any movement, while in fact there is someone working.

This report compares two different machine learning models. First the problem is formulated, then the methods are introduced briefly. Then each model is discussed more thoroughly and compared against eachother. Finally the comparison results are analyzed and a conclusion is drawn.

2 Problem Formulation

Sensor readings from a $27.6\,\mathrm{m}^2$ room are used as datapoints. The datapoints contain four light, sound, and temperature readings; two binary motion sensor readings; CO_2 reading; a CO_2 slope which was calculated by fitting a line on a sliding window of 25 CO_2 readings; timestamp; and occupancy count. The sensor readings were collected every 30 seconds and the occupancy count in the room was noted manually. The dataset was made by Adarsh Pal Singh and Dr. Sachin Chaudhari as part of their research on the subject[5]. It was acquired from the UCI Maching Learning Repository[4]. There are 10129 datapoints with 0 missing values. The data collection period was 4 days and the occupancy in the room was between 0 and 3 people.

The problem is a supervised multivariate multi-class classification problem. The features of the problem are the sensor readings and the calculated CO_2 slope. The timestamp is intentionally discraded to allow for a more general model. The label for the problem is the occupancy count which is an integer in range $[0, 3]$.

The scikit-learn[3] python package is used to train and evaluate the models. Pandas[7] and NumPy[1] are used to manipulate the data. Seaborn[6] and matplotlib[2] are used to visualise the findings.

3 Methods

Kmeans clustering was used to study the continuous features. Clustering with $k = 4$ showed that the four continuos features formed four quite distinctive clusters. The fact

that clustering showed four distinct clusters means that the classification methods should work quite well on the problem.

Logistic regression with linear maps and logistic loss was chosen as the first method. It was chosen since there seems to be some correlation between the room occupancy and sound, light, CO_2, and temperature readings, like shown in figure 1. Logistic regression also seemed like a great starting point, because it's one of the more simple model available. Since the dataset is not huge it's also a plus that logistic regression has good accuracy for smaller datasets.

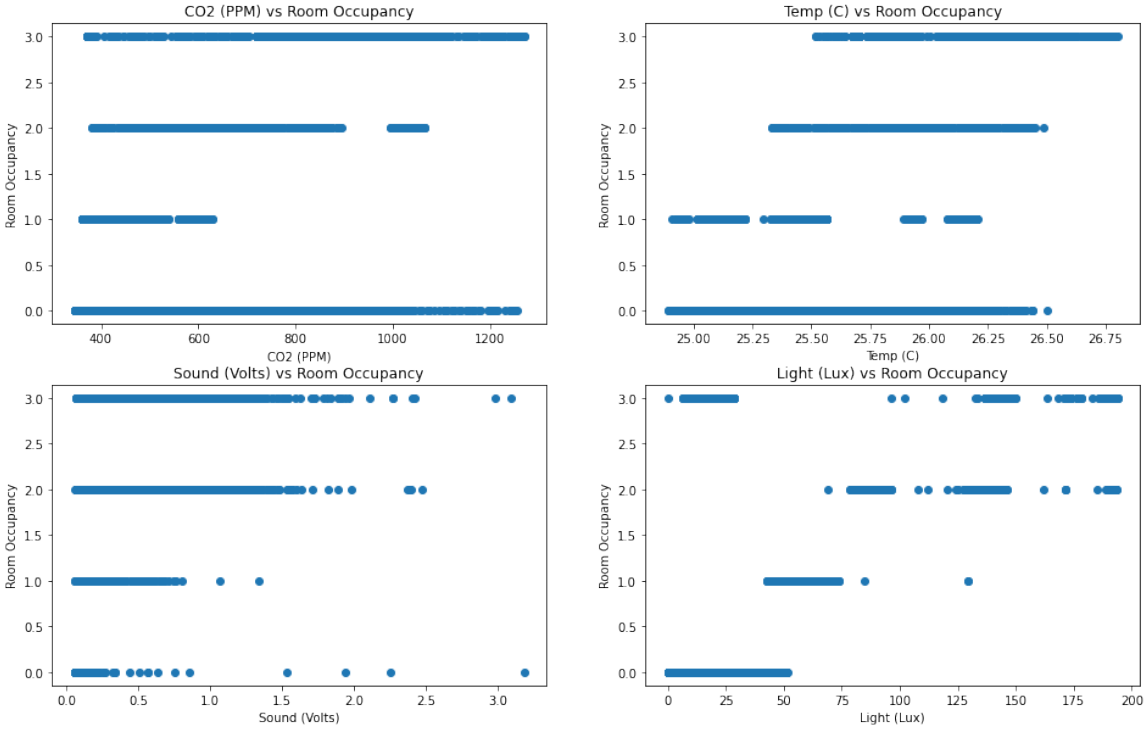


Figure 1: Scatterplots of the continuous features and the occupancy as coordinates

The second method is Random Forest with gini impurity loss function. Random Forest was chosen over Decision Tree to combat overfitting of the data. Decision Tree based classifier seemed like a good model because some data, like the motion sensor readings, correlated heavily with the room being empty or not. Based on the visualisations most of the features seemed to have a lot of information. For example, the motion sensor reading is really certain way to determine that the room is empty. Only in 24 out of 8204 motion sensor readings where the signal was 0 the room was actually occupied, like seen in figure 2.

The dataset was split to training, validation, and test sets with a 2:1:1 split. Test and validation sets are quite big to lessen the probability the one set would contain only datapoints from empty room. Having datapoints only from a empty room could be possible with smaller testing and validation sets, because great majority of the data is from an empty room. Splitting was done with `train_test_split` function from the `sklearn.model_selection` package.

The data was preprosessed to fit a more general usecase. Averages were calculated for temperature, sound, and light and a maximum value was taken from the two motion sensors.

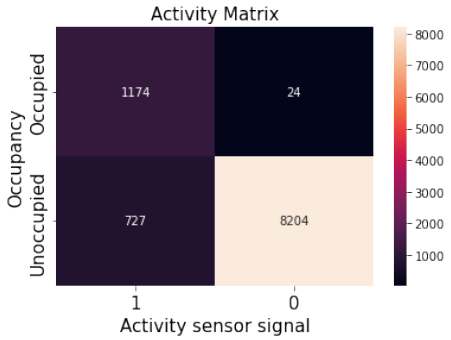


Figure 2: Occupancy compared to signal from movement sensor

This way the model could be used in any similar room with any number of sensors instead of needing, for example, 4 sensors for temperature, sound, and light.

3.1 Logistic regression model

Feature standardization is used to help the logistic regression model converge. The features were standardized with `sklearn.preprosessing.StandardScaler` to values in range $[-1, 1]$. `StandardScaler` was used since it produced the best results from the different scalers tested.

The grid search method was used to tune the hyperparameters for the logistic regression. Grid search tries all defined parameter candidates and compares the outputs in order to find the best possible parameters for the model. Accuracy of the model was used as the scoring criteria and tuned hyperparameters for my model are listed in table 1.

Parameter	C	penalty	solver
Value	10	l1	saga

Table 1: Hyperparameters for logistic regression

Errors for training and validation are calculated with logistic loss function. Logistic loss was used since it was the only allowed loss function for logistic regression.

Confusion matrix and validation accuracy are used to compare the two models. The `accuracy_score` and `confusion_matrix` functions from `sklearn.metrics` package are used to compute these metrics. Both functions are used to get better understanding of the underlying properties of the models. It's important to use other metrics alongside the the accuracy, because large part of the datapoints in the dataset are from an empty room which could skew the model.

3.2 Random Forest

The Random Forest model didn't use any standardization of the features. Standardization wasn't used because the underlying decision trees don't benefit from standardization.

Gini impurity was used as the loss function because it's faster to compute than the information gain. Which in turn makes model learn faster a bit faster. With my dataset the speedup was not really a noticeable but with a bigger set or larger number of estimators it could make a difference. The Gini impurity is also baked into the Random Forest

implementation of `sklearn`.

The same method of tuning hyperparameters was used on Random Forest as was used on Logistic Regression. The tuned parameters were number of estimators, maximum features, and minimum samples for a leaf. The grid search showed that the default values of `RandomForestClassifier` (listed in table 2) produced the most accurate result.

Parameter	n_estimators	min_samples_leaf	max_features
Value	100	1	sqrt

Table 2: Hyperparameters for random forest

4 Results

Accuracy scores are used to compare the models. The accuracy score of classification model is equivalent of the validation error of linear models. The logistic regression model managed to classify the validation data with 95,6% accuracy. When looking at the confusion matrix of the result, it can be seen that the model has some difficulties when the room has three people in it. It's quite unintuitive why this happens because one would think that it would be much easier to detect three people in a room compared to just one.

The random forest model performed even better and classified the validation data with 99,7% accuracy. And the confusion matrix of the model shows that it had no difficulties detecting the room being empty versus occupied. The only missclassifications happened with minimun two people in the room, which is logical.

Based on these considerations the random forest model was chosen. The test accuracy of the random forest model is 99,4% which is in line with the validation accuracy.

5 Conclusion

In this report classification models were used to classify the number of people in a room based on temperature, CO_2 level, etc. in the room. The two models compared in this report were logistic regression and random forest. Both models performed quite well, but the random forest model was chosen as the final model due to it's exceptional accuracy.

There doesn't seem to be a lot of room for improvement, since the accuracy is already so high. The high accuracy isn't a fluke either since the validation set contained data from each lable with appropriate porportion. Overfitting shouldn't be a problem either because of the overfitting control random forest provides.

The generality of the model is a bit of an question mark, since the data was collected from controlled environment. It would be interesting to see how the model would perform on data from a different room with different people. Another interesting thing is how the model would scale with bigger rooms and larger number of people.

References

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Adarsh Pal Singh and Dr. Sachin Chaudhari. *Room Occupancy Estimation Data Set.* Jan. 16, 2021. URL: https://archive.ics.uci.edu/ml/datasets/Room+Occupancy+Estimation (visited on 09/23/2022).

[5] Adarsh Pal Singh et al. "Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes". In: *IEEE Globecom Workshops* (2018).

[6] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[7] The pandas development team. *pandas-dev/pandas: Pandas.* Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

Appendix A

October 6, 2022

[92]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings

from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.exceptions import FitFailedWarning
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, silhouette_score,␣
 ↪davies_bouldin_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

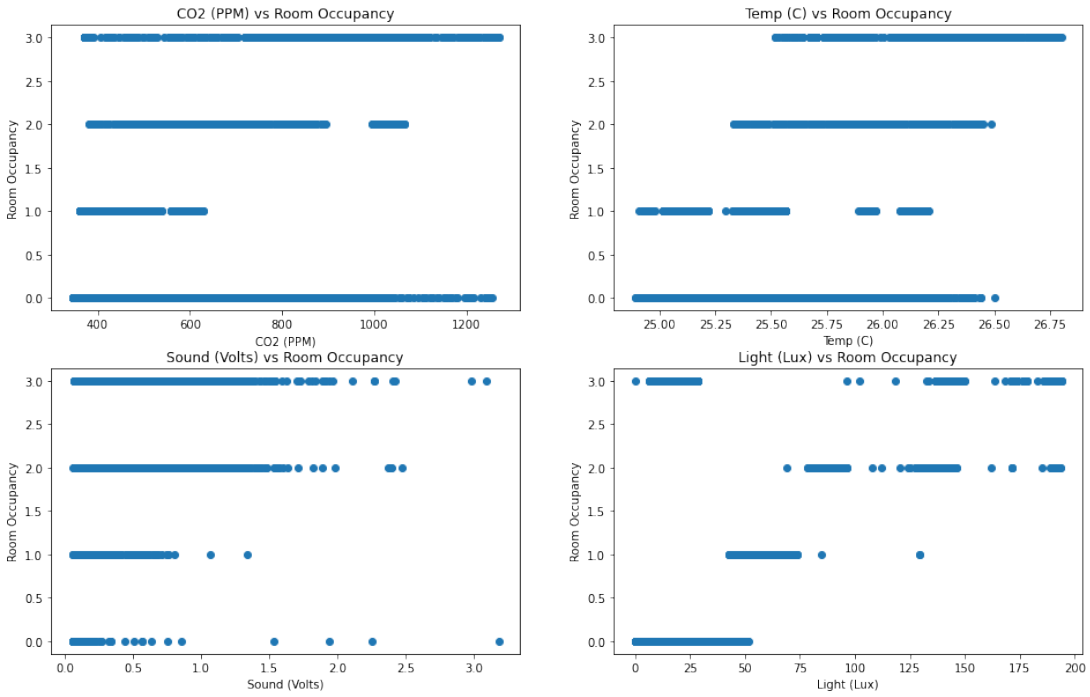
0.1 Preprocessing

[93]:
```
df = pd.read_csv('occupancy_estimation.csv')

average_light = [(s1 + s2 + s3 + s4) / 4 for s1, s2, s3, s4 in zip(
    df["S1_Light"], df["S2_Light"], df["S3_Light"], df["S4_Light"])]

average_sound = [(s1 + s2 + s3 + s4) / 4 for s1, s2, s3, s4 in zip(
    df["S1_Sound"], df["S2_Sound"], df["S3_Sound"], df["S4_Sound"])]

average_temp = [(s1 + s2 + s3 + s4) / 4 for s1, s2, s3, s4 in zip(
    df["S1_Temp"], df["S2_Temp"], df["S3_Temp"], df["S4_Temp"])]

pir = [max(s6, s7) for s6, s7 in zip(df["S6_PIR"], df["S7_PIR"])]

features = pd.DataFrame({
    "Light (Lux)": average_light,
    "Sound (Volts)": average_sound,
    "Temp (C)": average_temp,
    "PIR": pir,
    "CO2 (PPM)": df["S5_CO2"],
    "CO2 Slope": df["S5_CO2_Slope"]
```

```
})

X = features.to_numpy()
y = df['Room_Occupancy_Count'].to_numpy()

# Split data to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=77)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=77)
```

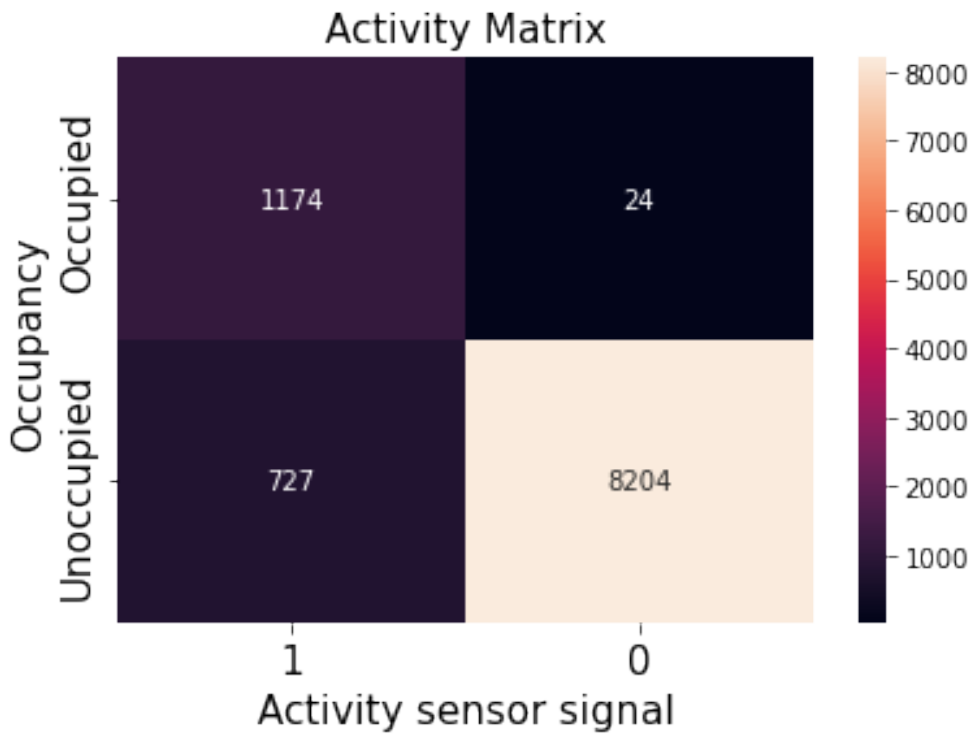
0.2 Plots

[94]:
```
columns = ['CO2 (PPM)', 'Temp (C)', 'Sound (Volts)', 'Light (Lux)']

plt.figure(figsize=(16, 10))
for index, column in enumerate(columns):
    data = features[column].to_numpy().reshape(-1, 1)
    plt.subplot(len(columns) // 2, 2, index + 1)
    plt.scatter(data, y)
    plt.xlabel(column)
    plt.ylabel('Room Occupancy')
    plt.title(f'{column} vs Room Occupancy')

plt.show()
```



[95]:
```
occupied_active, unoccupied_active, occupied_inactive, unoccupied_inactive =␣
 ↪(0, 0, 0, 0)

for movement, occupancy in zip(pir, y):
    if movement == 1 and occupancy > 0:
        occupied_active += 1
    elif movement == 1 and occupancy == 0:
        unoccupied_active += 1
    elif movement == 0 and occupancy > 0:
        occupied_inactive += 1
    else:
        unoccupied_inactive += 1

matrix = [[occupied_active, unoccupied_active], [occupied_inactive,␣
 ↪unoccupied_inactive]]

ax= plt.subplot()
sns.heatmap(matrix, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Activity sensor signal',fontsize=15)
ax.set_ylabel('Occupancy',fontsize=15)
ax.set_title('Activity Matrix',fontsize=15)
ax.xaxis.set_ticklabels(['1', '0'],fontsize=15)
ax.yaxis.set_ticklabels(['Occupied', 'Unoccupied'],fontsize=15)
```

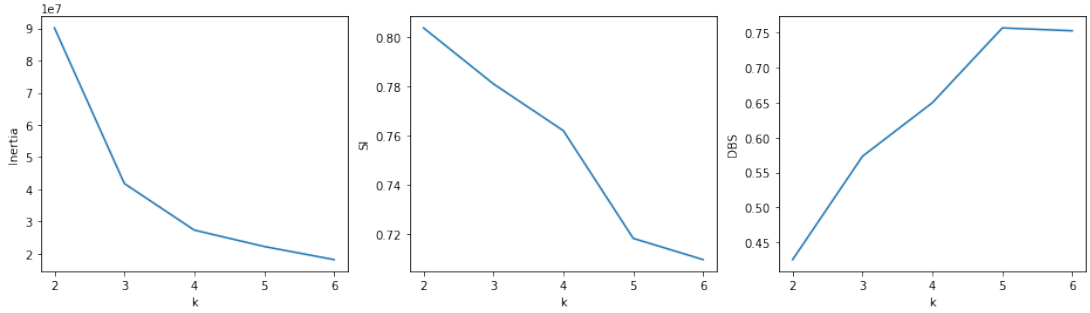
[95]:
```
[Text(0, 0.5, 'Occupied'), Text(0, 1.5, 'Unoccupied')]
```



0.3 KMeans

[96]:
```
inertia = []
SI = []
DBS = []

X_cluster = features.drop(['PIR', 'CO2 Slope'], axis=1)

for k in range(2, 7):
    kmeans = KMeans(n_clusters=k).fit(X_cluster)

    si = silhouette_score(X_cluster, kmeans.labels_)
    dbs = davies_bouldin_score(X_cluster, kmeans.labels_)

    inertia.append(kmeans.inertia_)
    SI.append(si)
    DBS.append(dbs)

plt.figure(figsize=(16,4))
plt.subplot(1,3,1)
plt.plot(np.arange(2,7),inertia)
plt.xlabel('k')
```

```
plt.ylabel('Inertia')
plt.xticks(np.arange(2,7))
plt.subplot(1,3,2)
plt.plot(np.arange(2,7),SI)
plt.xlabel('k')
plt.ylabel('SI')
plt.xticks(np.arange(2,7))
plt.subplot(1,3,3)
plt.plot(np.arange(2,7),DBS)
plt.xlabel('k')
plt.ylabel('DBS')
plt.xticks(np.arange(2,7))
plt.show()
```



[97]:
```
kmeans = KMeans(n_clusters = 4)

kmeans.fit(X_cluster)
si = silhouette_score(X_cluster, kmeans.labels_)
dbs = davies_bouldin_score(X_cluster, kmeans.labels_)

print(f'inertia: {kmeans.inertia_}')
print(f'silhouette: {si}')
print(f'davies_bouldin_score: {dbs}')
```

```
inertia: 27318043.8502322
silhouette: 0.7621564198222979
davies_bouldin_score: 0.6499888563399825
```

0.4 Logistic regression

[98]:
```
# Standardization
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

scaler = StandardScaler().fit(X_val)
```

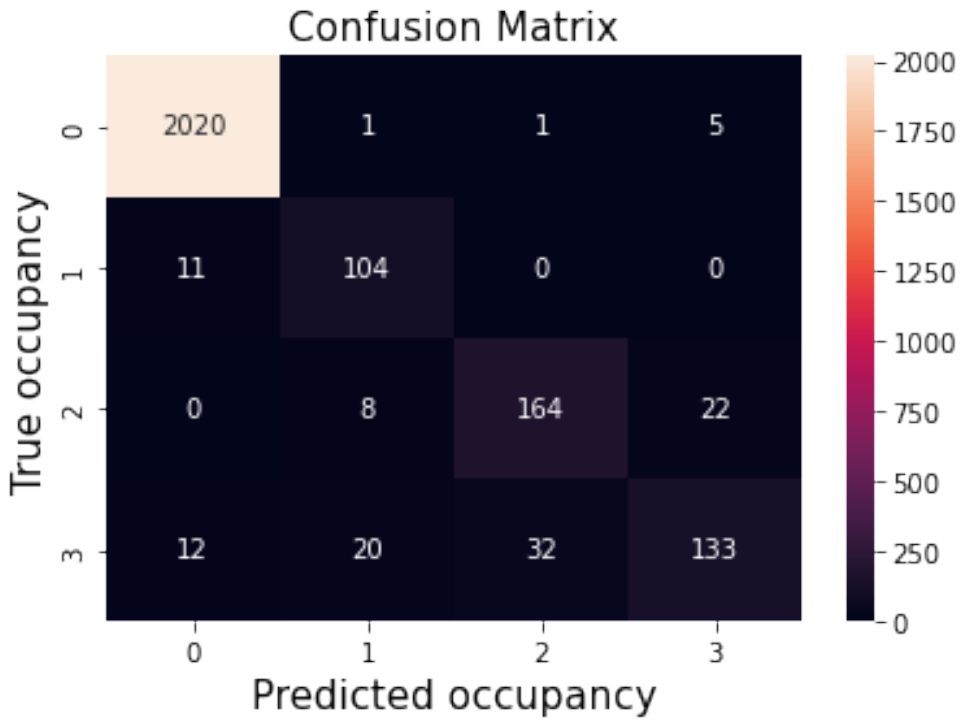
```
X_val_scaled = scaler.transform(X_val)
```

[99]:
```
# Find the best params for LogisticRegression with grid search
C_candidates = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
penalty_candidates = ['l1', 'l2', 'elasticnet']
solver_candidates = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
grid = { "solver": solver_candidates, "penalty": penalty_candidates, "C":␣
 ↪C_candidates, "max_iter": [10000] }
grid_search = GridSearchCV(estimator = LogisticRegression(), param_grid=grid,␣
 ↪n_jobs=-1, scoring='accuracy', error_score=0)
warnings.filterwarnings('ignore', category=FitFailedWarning) # Supress the fit␣
 ↪failed warnings
grid_result = grid_search.fit(X_train_scaled, y_train)


print(f'{grid_result.best_params_} resulted in best score ({grid_result.
 ↪best_score_})')
```

```
{'C': 100, 'max_iter': 10000, 'penalty': 'l1', 'solver': 'saga'} resulted in
best score (0.9571483754667582)
```

[100]:
```
# Fit to logistic regression model
log_regr = LogisticRegression(C=100, max_iter=10000, solver="saga",␣
 ↪penalty="l1")
log_regr.fit(X_train_scaled, y_train)

# Calculating training accuracy
y_train_pred = log_regr.predict(X_train_scaled)
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculating testing accuracy
y_val_pred = log_regr.predict(X_val_scaled)
val_accuracy = accuracy_score(y_val, y_val_pred)

# Confusion matrix
conf_mat = confusion_matrix(y_val, y_val_pred)

ax= plt.subplot()
sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted occupancy',fontsize=15)
ax.set_ylabel('True occupancy',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)

print(f'Training accuracy: {train_accuracy}')
print(f'Validation accuracy: {val_accuracy}')
plt.show()
```

```
Training accuracy: 0.9581358609794629
Validation accuracy: 0.9557836557441769
```



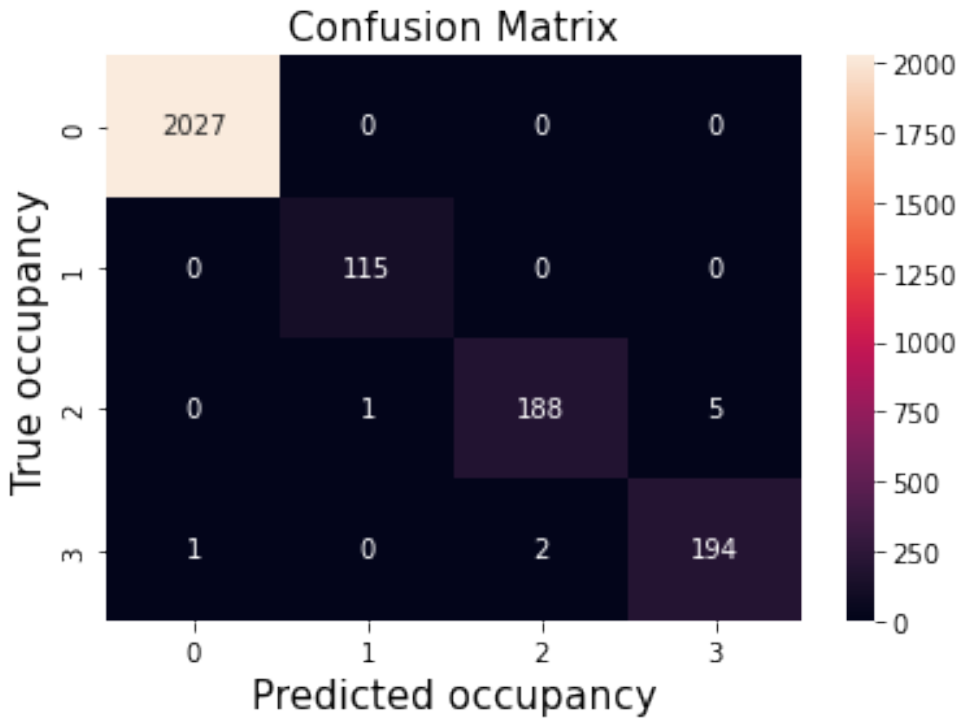
0.5 Random Forest

[101]:
```
# Find the best params for RandomForest with grid search
estimator_candidates = [10, 100, 200]
max_features_candidates = ['sqrt', 'log2', None]
min_samples_leaf_candidates = list(range(10))
grid = { "n_estimators": estimator_candidates, "max_features":␣
 ↪max_features_candidates, "min_samples_leaf": min_samples_leaf_candidates }
grid_search = GridSearchCV(estimator = RandomForestClassifier(),␣
 ↪param_grid=grid, n_jobs=-1, scoring='accuracy', error_score=0)
warnings.filterwarnings('ignore', category=FitFailedWarning) # Supress the fit␣
 ↪failed warnings
grid_result = grid_search.fit(X_train, y_train)


print(f'{grid_result.best_params_} resulted in best score ({grid_result.
 ↪best_score_})')
```

```
{'max_features': 'sqrt', 'min_samples_leaf': 1, 'n_estimators': 100} resulted in
best score (0.9968404808634002)
```

[102]:
```
random_forest = RandomForestClassifier()

random_forest.fit(X_train, y_train)

# Calculating training accuracy
y_train_pred = random_forest.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculating testing accuracy
y_val_pred = random_forest.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)

# Confusion matrix
conf_mat = confusion_matrix(y_val, y_val_pred)

ax= plt.subplot()
sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted occupancy',fontsize=15)
ax.set_ylabel('True occupancy',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)

print(f'Training accuracy: {train_accuracy}')
print(f'Validation accuracy: {val_accuracy}')
plt.show()
```

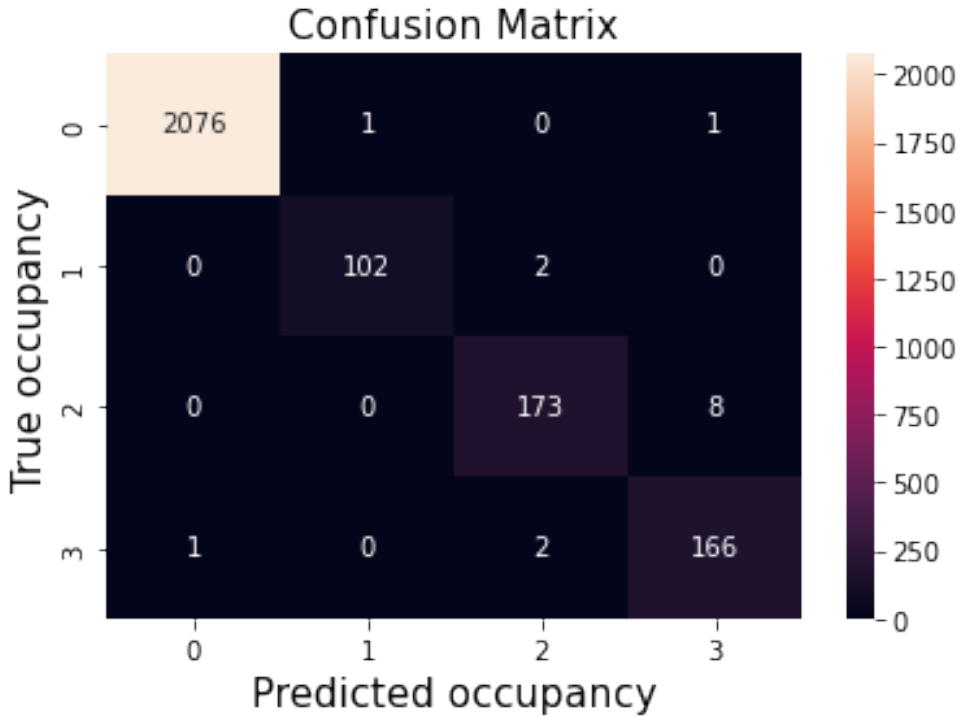
```
Training accuracy: 1.0
Validation accuracy: 0.9964469009080142
```



0.6 Test with the chosen method with the test set

[105]:
```
# Calculate test accuracy
y_test_pred = random_forest.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
conf_mat = confusion_matrix(y_test, y_test_pred)
ax= plt.subplot()
sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted occupancy',fontsize=15)
ax.set_ylabel('True occupancy',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)

print(f'Test accuracy: {test_accuracy}')
plt.show()
```

```
Test accuracy: 0.9940758293838863
```


Confusion Matrix
2076
1
0
1
0
102
2
0
0
0
173
8
1
0
2
166
True occupancy
Predicted occupancy
0
1
2
3
2000
1750
1500
1250
1000
750
500
250
0