# Technical Documentation

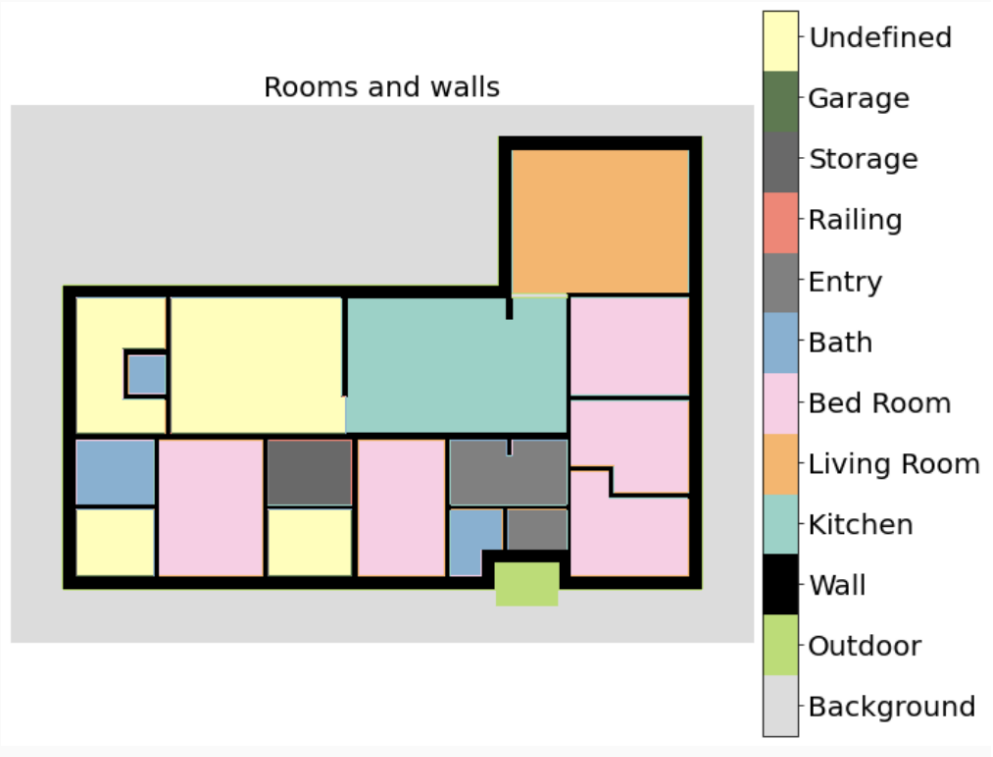


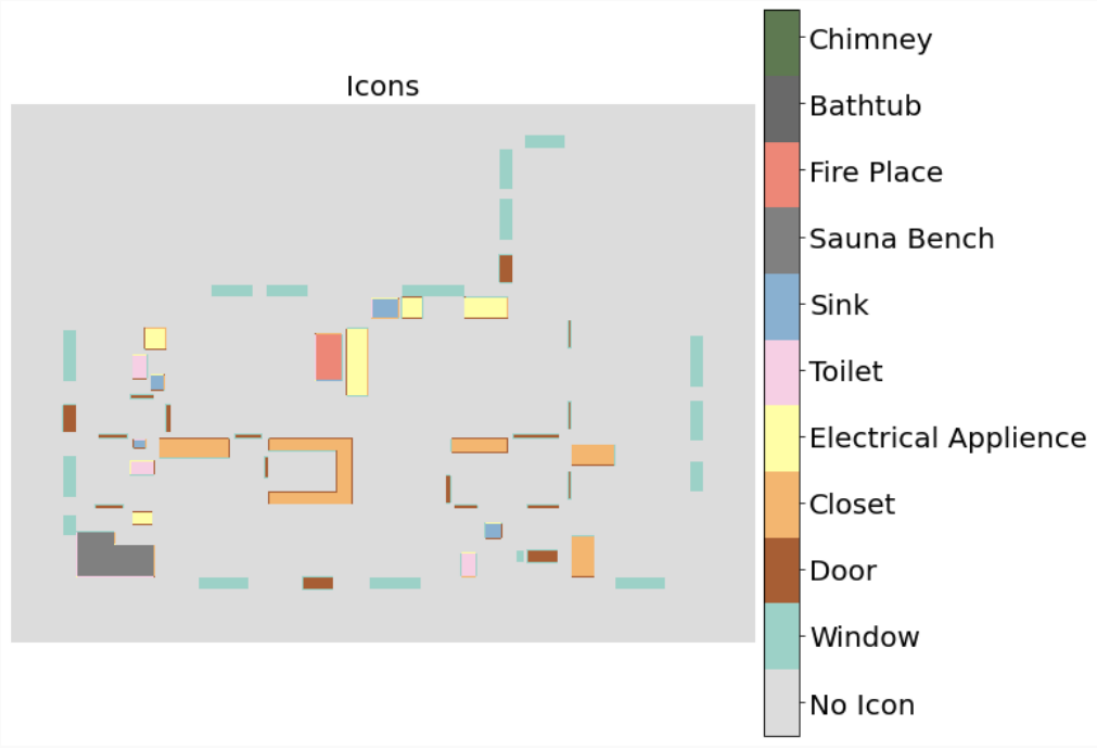
## Dataset Overview

The Cubicasa 5k dataset is a large-scale dataset of floorplans that contains 5,000 floorplans from various building types, such as apartments, houses, and offices. It is partitioned into train (4200), validation (400), and test (400).

What makes this dataset particularly useful is the fact that the floorplans are not only annotated with room type labels, such as bedroom, kitchen, living room, etc., but also with icon labels, such as electrical appliance, sauna, closet, etc. This makes it a valuable resource for training and evaluating room type classification models.

For each floorplan in the dataset, there are two corresponding 2D numpy arrays (images), one for the room layer and one for the icon layer. Both arrays are of the same size and contain numerical values that represent the room type or icon label for each pixel in the floorplan.




The goal is to use this dataset to train classification models for room recognition. However, the dataset contains some rooms that are undefined, which can pose a challenge for training. I

decided not to train on floor plans that contain undefined rooms. This is because the undefined room class should never be predicted.

It turns out that a large proportion of the floor plans in the dataset contain undefined rooms - around 90.5%, in fact. This means that I need to be careful about how to handle this issue, as simply discarding all floor plans with undefined rooms would result in a significant loss of data. Therefore, I will need to find a way to salvage as much of the data as possible while still ensuring that we only train on floor plans that do not contain undefined rooms.

By addressing this issue head-on, I can ensure that my models are trained on high-quality data and are able to generalize well to new floor plans.

Let's start to salvage the data. The first step is to classify some of the unidentified rooms.

# floorplans_to_counts.ipynb

This code extracts features from floor plan images in the CubiCasa5k dataset by counting the number of occurrences of 11 different types of icons in each of the 12 different types of rooms. The output is a pandas DataFrame where each row represents a room in a floorplan, with columns for the count of each icon type in each room type. The code also handles cases where rooms have less than four points by storing the image index and room type in a separate errors DataFrame.

## Libraries

The code imports several libraries at the beginning, including:

- Pandas: a library for data manipulation and analysis.
- NumPy: a library for numerical computing in Python.
- OpenCV: a library for computer vision and image processing.
- Matplotlib: a library for creating data visualizations.
- Shapely: a library for working with geometric objects.
- PyTorch: a machine learning library that provides Tensors and dynamic computational graphs.

The code also imports classes and functions from the floortrans package, which is used for loading and manipulating floor plan images. Specifically, it uses classes for loading floor plan SVG files, transforming them into tensors, and applying various image processing techniques. Additionally, it uses functions for plotting segmentation maps and polygons, as well as for splitting predictions and getting polygons from segmentation maps.

## Code Explanation

The code reads in floor plan images and labels from the CubiCasa5K dataset and performs an analysis of the overlap between room types and icon classes.

The floor plan images and labels are loaded using the FloorplanSVG class from the floortrans.loaders module. The data is then iterated through, with the isolate_class function used to select the region on the image where a particular class exists. The findContours function from the cv2 library is then used to find individual incidents of the class. The incidents are looped through and any polygons with less than four vertices are ignored. The Shapely library is used to create polygons from the vertex points, and the buffer(0) operation is used to ensure that the polygons stay valid. The polygons are then compared to find incidents of overlap between the room type and icon class, with the number of overlaps counted for each room type and icon class combination.

The analysis results are stored in a pandas DataFrame, with the number of overlaps between each room type and icon class combination recorded. The DataFrame includes an "Image_idx" column to identify which floorplan each row of data came from, as well as a "Room" column to identify which room type each row of data corresponds to.

| | No Icon | Window | Door | Closet | Electrical Applience | Toilet | Sink | Sauna Bench | Fire Place | Bathtub | Chimney | Image_idx | Room |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 16.0 | 12.0 | 3.0 | 6.0 | 2.0 | 2.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0 | 0 |
| **1** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 1 |
| **2** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 1 |
| **3** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 1 |
| **4** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **6162** | 1.0 | 0.0 | 0.0 | 2.0 | 3.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 379 | 7 |
| **6163** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 379 | 8 |
| **6164** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 379 | 8 |
| **6165** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 379 | 8 |
| **6166** | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 379 | 11 |

6167 rows × 13 columns

The code also includes error handling for polygons with less than four vertices, with any errors recorded in a separate DataFrame named "errors", for later inspection.

## Output and next steps

I ran this code for train.txt (file containing the filenames of all the floorplans in the training set), val.txt, and test.txt. The outputs can be seen in the following files:

- dataframes/train_df.pkl
- dataframes/val_df.pkl
- dataframes/test_df.pkl

Now let’s use this information to clean up the dataset.

# dataset_cleaning.ipynb

The code provided attempts to solve a classification problem, where the task is to predict the type of a room given the counts of icons in the room. The code tests several classification models and finds that they all perform similarly. After testing the models, the logistic regression model was chosen as the best model for the task.

To improve the accuracy of the logistic regression model, the code finds a probability threshold that ensures predictions with a probability of accuracy equal to or higher than the threshold have a 90% accuracy rate on the validation set. This probability threshold is then used to make predictions on rooms.

This code then relabel rooms in floorplans that are initially labeled as "Undefined" using the model and additional manually inputted rules.

## Libraries

- pickle: A module used for serializing and de-serializing Python objects.
- Pandas: a library for data manipulation and analysis.
- NumPy: a library for numerical computing in Python.
- sklearn: A machine learning library that includes a variety of classification, regression, and clustering algorithms.
- sklearn.naive_bayes.MultinomialNB: A classifier that uses the multinomial distribution to model the likelihood of each feature given each class, and the prior probability of each class.
- sklearn.ensemble.RandomForestClassifier: A classifier that fits multiple decision trees to the data and aggregates their predictions.
- sklearn.neighbors.KNeighborsClassifier: A classifier that uses the k-nearest neighbors of a data point to predict its class.
- sklearn.linear_model.LogisticRegression: A classifier that models the log odds of each class as a linear function of the input features.
- matplotlib: provides plotting and visualization
- torch: provides machine learning framework
- cv2: provides image processing and computer vision functionality
- shapely: provides geometric operations on polygons

## Code Explanation

The code uses the training_df and val_df created in floorplans_to_counts.ipynb

Four classification algorithms, namely Multinomial Naive Bayes, Random Forest Classifier, K-Nearest Neighbors Classifier, and Logistic Regression Classifier are used to train models on the training data and predict the target variable on the validation data. For each algorithm, the classification report and accuracy score are printed.

## Model Explanation

Multinomial Naive Bayes (MNB) Classifier:

- A probabilistic classifier that uses Bayes' theorem to predict the probability of an instance belonging to a certain class based on its feature vector.
- Assumes that the features are independent of each other and follow a multinomial distribution.
- Commonly used for text classification and spam filtering.

Random Forest (RF) Classifier:

- A decision tree-based ensemble method that creates multiple decision trees using different subsets of the training data and features.
- Each decision tree independently predicts the target variable, and the final prediction is made by taking the majority vote of all the trees.
- Helps to reduce overfitting and improve generalization.

K-Nearest Neighbors (KNN) Classifier:

- A non-parametric classifier that uses the distance between instances to determine their similarity and predict the target variable.
- K is a hyperparameter that specifies the number of nearest neighbors to consider.
- Simple to implement but can be computationally expensive and sensitive to the choice of K and the distance metric.

Multiclass Logistic Regression Classifier:

- A linear model that uses logistic regression to predict the probability of an instance belonging to each class.
- The predicted probabilities are then converted into class labels using a decision threshold.
- Can handle multiple classes and is commonly used for image classification and natural language processing.

| Model | Accuracy |
|---|---|
| Multinomial Naive Bayes | 0.558 |
| Random Forest | 0.586 |
| KNN | 0.575 |
| Multiclass Logistic Regression | 0.565 |

I chose to go with the multiclass logistic regression model for several reasons:

1. Simplicity: Multiclass logistic regression is a simple algorithm that is easy to understand and implement. It is based on the principle of maximum likelihood estimation, which makes it easy to interpret the results.
2. Interpretability: Logistic regression produces a linear decision boundary that can be easily interpreted. The coefficients of the linear equation can be used to identify the most important features for the classification task.
3. Random forest: While random forest is a powerful algorithm that can handle nonlinear relationships between features, it can be slow to train on large datasets and is difficult to interpret.
4. KNN: KNN is a simple algorithm that can handle nonlinear decision boundaries, but it can be slow to predict on large datasets and requires a large amount of memory to store the training data.
5. Multinomial NB: Multinomial NB is a simple algorithm that is efficient for text classification problems, but it assumes that the input features are independent, which may not be true in practice.

In summary, while each algorithm has its own strengths and weaknesses, multiclass logistic regression is a simple and efficient algorithm that is often a good choice for multiclass classification problems.

## Relabeling Undefined Rooms in Floorplans

The process for relabeling rooms in the floorplan is as follows:

1. Import the necessary libraries.
2. Load a set of floorplans.
3. Iterate through each floorplan.
4. Isolate the rooms that are labeled as "Undefined" in the floorplan.
5. For each isolated undefined room, check for the number of instances of each icon class that intersects with the room.
6. Use the trained logistic regression classifier to predict the room label for the undefined room.
7. Relabel the undefined room in the floorplan with the predicted label.
8. Save the modified floorplans to disk.

Overall, this code provides a useful tool for improving the accuracy and completeness of room labels in floorplans.

Now that our dataset is more cleaned up, we can use this to create our dataset for the GNN.

# gnn_inputs.ipynb

This code processes floor plan images to extract features and embeddings for use in machine learning models.

## Libraries

Several libraries are imported at the beginning of the code, including:

- Pandas: a library for data manipulation and analysis.
- PyTorch: a machine learning library that provides Tensors and dynamic computational graphs.
- OpenCV: a library for computer vision and image processing.
- NetworkX: a library for creating and manipulating graphs.
- Matplotlib: a library for creating data visualizations.
- Shapely: a library for working with geometric objects.
- Scikit-image: a collection of algorithms for image processing.
- In addition, the floortrans package is imported, which contains functions and classes for loading and manipulating floor plan images.

## Helper Functions

Several helper functions are defined in the code, including:

### isolate_class

This function takes an image of a floor plan and a target class as input and returns a binary mask with 1's at the locations where the target class is present in the image.
Inputs:

- rooms (numpy array): The input image of the floor plan.
- CLASS (int): The target class.

Outputs:

- template (numpy array): The binary mask with 1's at the locations where the target class is present in the input image.

Example:

```
import numpy as np

# Define an example input image
rooms = np.array([[1, 2, 2], [0, 2, 1], [2, 2, 1]])

# Call the function to isolate class 1
template = isolate_class(rooms, 1)

print(template)
```

Output of the example:

```
[[0 0 0]
 [0 0 1]
 [0 0 1]]
```

## vis_nodes

This function takes an image and a list of significant nodes (classes) as input and returns the contours of the rooms, the contours of the doors, and the centroid locations of the significant nodes. The function initializes empty lists to hold the room contours and nodes, gets a binary mask with 1's at the locations where the current class is present in the image, finds the contours of the connected components in the binary mask, adds each contour to the list of room contours, computes the centroid of the contour and adds it to the list of node locations. The function then returns the room contours, door contours, and node locations.

## get_edges

This function takes the contours of the rooms and doors as input and returns the connections between the rooms as a list of pairs of room indices and as a list of pairs of room centroid locations. The function initializes empty lists to hold the room connections as indices and as centroid locations, iterates over each room contour to compare with other room contours, checks if the polygons intersect and if there is a door between the two rooms, and adds the pair of room indices or room centroids to the appropriate list if the conditions are met. The function then returns the list of connections as indices and as centroid locations.

## Create_dataframes

The last helper function in the code is "create_dataframes", which takes as input a sparse adjacency matrix representing the connections between nodes in a graph, the embeddings of the nodes, a list of node indices, the areas of the nodes, the relative areas of the nodes, the degrees of the nodes, and a list of the names of the node classes.

The function creates two pandas DataFrames from this input: edges_df and attributes_df. The edges_df DataFrame contains two columns: "source" and "target", which represent the indices of the source and target nodes connected by an edge in the graph. The attributes_df DataFrame contains four columns: "Area", "Relative Area", "Number of neighboring rooms", and "Room Type".

The "Area" column contains the areas of the nodes, the "Relative Area" column contains the relative areas of the nodes, the "Number of neighboring rooms" column contains the degrees of the nodes, and the "Room Type" column contains the names of the node classes.

The function returns the edges_df and attributes_df DataFrames. These DataFrames can be used as input to machine learning algorithms to predict properties of nodes or edges in the graph.

## Main Function

The process_file function takes a file path as input and extracts the room contours, door contours, and nodes using the previously defined helper functions. It then creates a NetworkX graph object and computes the adjacency matrix of the graph. It calculates the degree of each node, converts the node classes to one-hot vectors, and concatenates them with the relative areas. It then creates embeddings for the graph and generates dataframes for edges and attributes.

Finally, the function returns embeddings, embeddings2, and Y, where embeddings and Y are input data for the graph neural network model. It also returns a dataframe with embeddings for the non-graph model, and the dataframes for edges and attributes.

In conclusion, the code processes floorplan images and extracts various features, including relative areas and number of neighboring rooms, to create graph representations of the floorplan. It generates embeddings for the graph and returns input data for the graph neural network model.

This is a extremely computationally heavy piece of code to run. I had limited memory, so I was only able to run it on 2400 floorplans from the training data. I was able to run it on the test and validation sets in full. So moving forward, training data is smaller.

# baseline_gnn.ipynb

The purpose of this code is to train and evaluate a GCN model for predicting room types in floorplans. The code loads preprocessed data from .pkl files, creates PyTorch Geometric Data objects for each floorplan, trains a GCN model using the training data, and evaluates the GCN model on the testing data. The code outputs the accuracy of the GCN model on the test data.

## Libraries

- pickle: for loading the preprocessed data from .pkl files
- Pandas: a library for data manipulation and analysis.
- torch: for building and training the GCN model
- torch.nn.functional: for defining the forward function of the GCN model
- torch_geometric.data: for creating PyTorch Geometric Data objects
- torch_geometric.nn: for defining GCN layers
- torch_geometric.data: for loading data batches during training

## Helper Functions

The code defines the following helper functions:

### create_data

This function takes in the attributes and edges for a floorplan and returns a PyTorch Geometric Data object. The function converts the data into tensors and sets the x, edge_index, and y attributes of the Data object. This excludes “Area” as it led to poor performance.

### Net

This class defines the GCN model. The model consists of two GCNConv layers with ReLU activation and a final log_softmax layer. The model takes in the number of input features, number of classes, number of hidden units, and dropout rate as hyperparameters. The forward function applies the GCNConv layers and returns the final logits after applying the log_softmax activation.

### train

This function defines the training loop for the GCN model. The function takes in the model, optimizer, and data loader as inputs. It loops over the data batches in the data loader, computes the forward pass through the model, calculates the loss, computes gradients, and updates the model parameters. The function returns the average loss across all batches.

# baseline_prediction.ipynb

This code uses two machine learning models, Logistic Regression and Random Forest, to predict room types for floor plans. The code loads pre-processed data from two pickle files, one for training data and one for testing data. The data is then used to train both models and make predictions on the testing data. This excludes “Area” as it led to poor performance. The performance of each model is evaluated using classification metrics, including accuracy, precision, recall, and f1-score.

## Libraries:

The code imports several popular Python libraries for machine learning and data analysis:

- pickle: used for loading data from pickle files
- Pandas: a library for data manipulation and analysis.
- sklearn.linear_model.LogisticRegression: used to train a logistic regression model
- sklearn.metrics: used to calculate classification metrics
- sklearn.ensemble.RandomForestClassifier: used to train a random forest model

## Code Explanation

The code is divided into two parts, one for training and evaluating a logistic regression model, and one for training and evaluating a random forest model. Both parts follow a similar structure:

1. Load pre-processed training and testing data from pickle files
2. Train the model on the training data
3. Use the model to make predictions on the testing data
4. Evaluate the performance of the model using classification metrics