Process Documentation

Creating the dataset and the model training process was developed parallely.
I picked xgboost, as it is a Gradient Boosting algorithm and works well in most cases.

I was creating the dataset of list of tuple (str, str, label). for the pair of string, I converted them in to the numerical features.
A single list of numeric feature are created to predict the label. Thus, it becomes the binary classification in machine Learning.

Feature Creation

After iterations of experimentation I have used following features:

1. Jaccard Similarality score
2. Is substring flag
3. cosine similarity score of the embedding of string: This helps to capture the semantic meaning, without using the large dimension of embedding models.
4. char_ngram_jaccard. (not my personal preferences, I have added it just to experiment)
5. classical tfidf cosine similarity score.
6. sequence matcher ratio , based on sequencematcher module in python

I combined them all together in a weighted patern , [0.8 *cos_sim, jac, substr, seq_ratio, 1.5* emb_cos_sim, char_ngram_jaccard_score]
This was done for priortizing the semantics over syntax,

I tried standard scaler and feature selection method, none impacted significantly in the output.
This might be due to the small dataset and around 35% of label 1 strings are different, more than half of the string are already exactly same. Thus, models needs to rely on the outliers to learn from the dataset.

I also tried pure embedding based model, but it didn't make sense in the room names settings and the names are not a complete sentences. This data needs multi step feature extraction for semantics. Asking semantic questions like:

1. How many rooms ?
2. what kind of property ?
3. Any more info like : pets / smoking ?

After the answer, and generating a full length of sentences, May be embedding models make sense.

Model Training

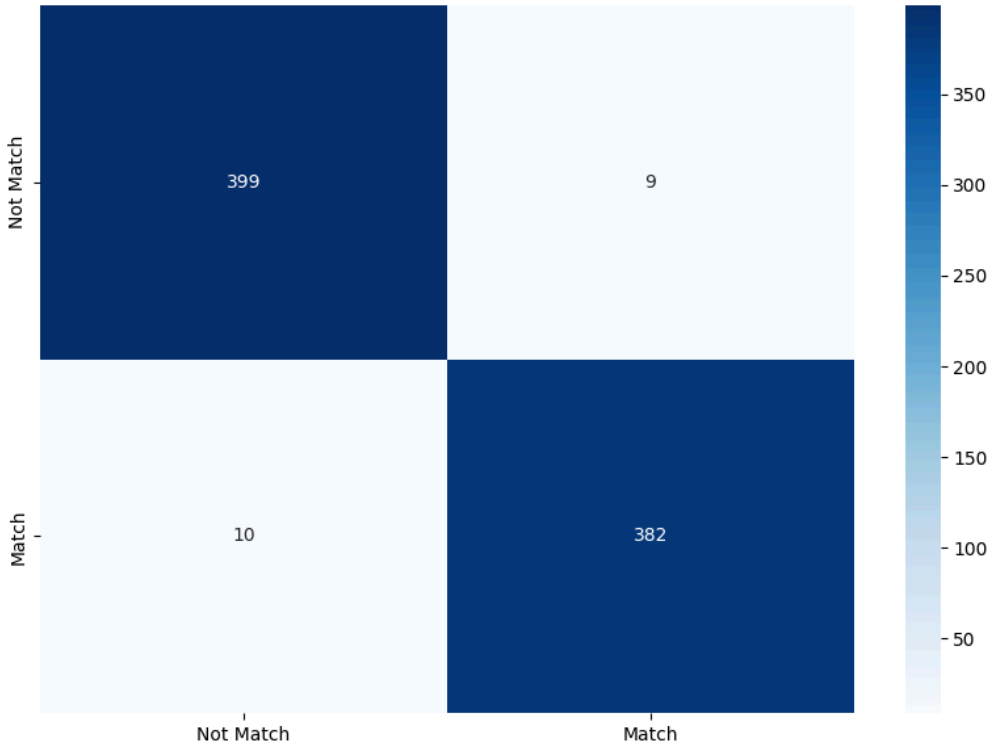
XGboost Model is trained. logloss is used as the eval metrics

The training accuracy of the model : 0.97625
The classfification report of the model :

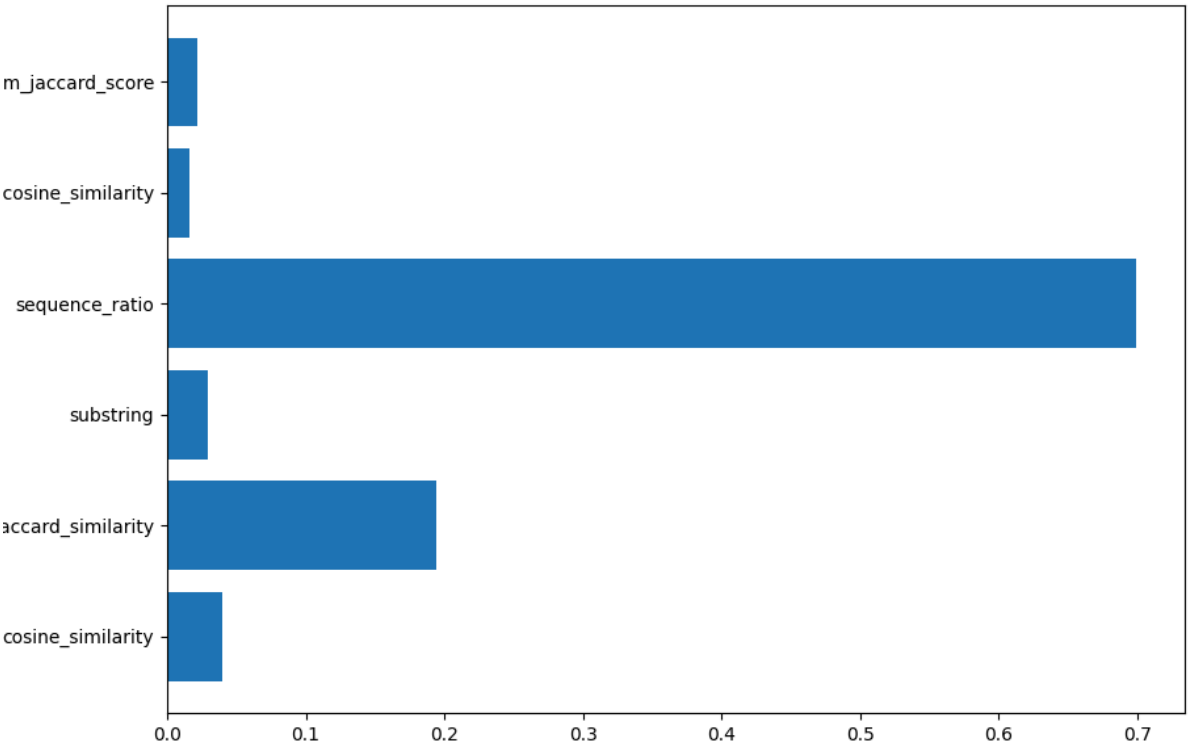
	precision	recall	f1-score	support
0	0.98	0.98	0.98	408
1	0.98	0.97	0.98	392
accuracy			0.98	800

macro avg 0.98 0.98 0.98 800

weighted avg 0.98 0.98 0.98 800



Feature importance:



Likewise, Hyperparameter is tuned based on the following objective :

```
"objective": "binary:logistic",

"eval_metric": self.config["model_configs"]["xgb"]["fixed_params"]
["eval_metric"],

"n_estimators": trial.suggest_int("n_estimators", 100, 300, step=100),

"max_depth": trial.suggest_int("max_depth", 3, 10),

"learning_rate": trial.suggest_float("learning_rate", 0.01, 0.1,
log=True),

"min_child_weight": trial.suggest_int("min_child_weight", 1, 5),

"gamma": trial.suggest_float("gamma", 0.0, 0.5),

"subsample": trial.suggest_float("subsample", 0.5, 1.0),

"colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),

"reg_alpha": trial.suggest_float("reg_alpha", 0.0, 0.5),
```

```
"reg_lambda": trial.suggest_float("reg_lambda", 0.0, 0.5),

"random_state": self.config["random_state"],
```

results:

```
- 0.7962979498154026

gamma:

- 0.07407655998633511

learning_rate:

- 0.02053083889963104

max_depth:

- 8

min_child_weight:

- 3

n_estimators:

- 300

reg_alpha:

- 0.3100421704998185

reg_lambda:

- 0.34097532561863875

subsample:

- 0.7093174872587061
```

More digging in Hyper parameter is needed, as I wanted to try other models too, may be next time.

likewise, Test and coverage reports are created using fastapi testclient and are integration test instead of unit tests.

Likewise, most of the notebooks are scrappy , as I experiment over it and convert the code later in the python modules.

Inference Speed

Currently, All possible features are implemented in the system, which make it slow in the inference end.
Selecting the feature and optimization is the next step to improve the inference speed.