

Machine Learning

| Fine-Tune Your Model

Prof. Dr. Mostafa Elhosseini

<https://youtube.com/drmelhosseini>

| Agenda

- Fine-Tune Your Model
- Grid Search

| Grid Search

- Manually adjusting the hyperparameters to identify an optimal set of values. However, this method can be exceedingly laborious, potentially limiting the extent of exploration due to time constraints.
- Instead, you can use Scikit-Learn's **GridSearchCV** class to search for you. All you need to do is tell it **which hyperparameters** you want it to experiment with and **what values to try out**, and it will **use cross-validation to evaluate all the possible combinations** of hyperparameter values

| Grid Search

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])

param_grid = [
    {'preprocessing__geo__n_clusters': [5, 8, 10],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo__n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
```

```
def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"] # feature names out

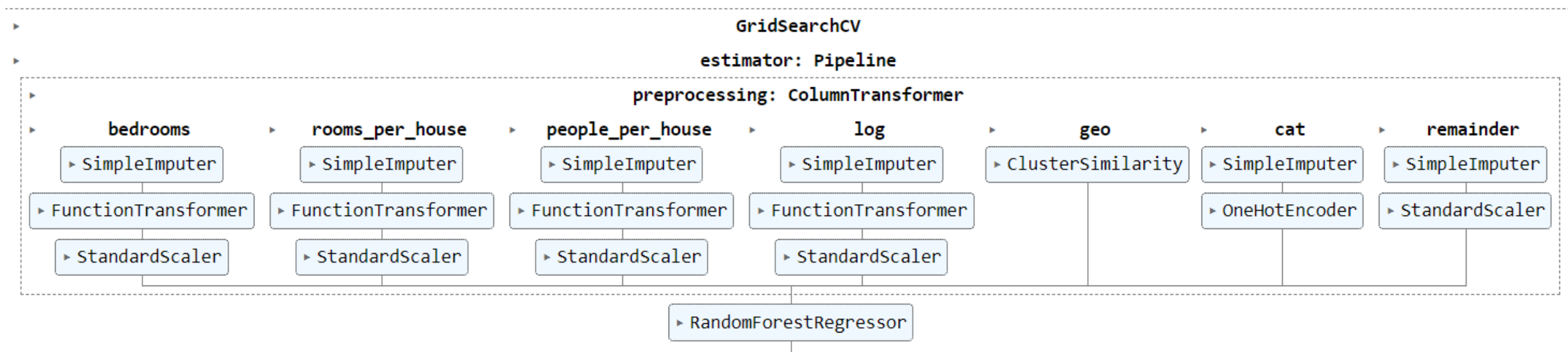
def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())

log_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(np.log, feature_names_out="one-to-one"),
    StandardScaler())

cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                                     StandardScaler())

preprocessing = ColumnTransformer([
    ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]),
    ("rooms_per_house", ratio_pipeline(), ["total_rooms", "households"]),
    ("people_per_house", ratio_pipeline(), ["population", "households"]),
    ("log", log_pipeline, ["total_bedrooms", "total_rooms", "population",
                          "households", "median_income"]),
    ("geo", cluster_simil, ["latitude", "longitude"]),
    ("cat", cat_pipeline, make_column_selector(dtype_include=object)),
],
remainder=default_num_pipeline) # one column remaining: housing_median_age
```

| Grid Search



| Grid Search

- Notice that you can refer to any hyperparameter of any estimator in a pipeline, even if this estimator is nested deep inside several pipelines and column transformers.
- For example, when Scikit-Learn sees **"preprocessing__geo__n_clusters"**, it splits this string at the double underscores, then it looks for an estimator named "preprocessing" in the pipeline and finds the preprocessing ColumnTransformer. Next, it looks for a transformer named "geo" inside this ColumnTransformer and finds the ClusterSimilarity transformer we used on the latitude and longitude attributes. Then it finds this transformer's n_clusters hyperparameter

| Grid Search

- Wrapping preprocessing steps in a Scikit-Learn pipeline allows you to **tune** the preprocessing hyperparameters **along with** the model **hyperparameters**. This is a good thing since they often interact.
- For example, perhaps increasing `n_clusters` requires increasing `max_features` as well. If fitting the pipeline transformers is computationally expensive, you can set the pipeline's memory hyperparameter to the path of a caching directory: when you first fit the pipeline, Scikit-Learn will save the fitted transformers to this directory. If you then fit the pipeline again with the same hyperparameters, Scikit-Learn will just load the cached transformers.

| Grid Search

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])

param_grid = [
    {'preprocessing__geo__n_clusters': [5, 8, 10],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo__n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
```

- $3 \times 3 = 9$
- $2 \times 3 = 6$
- In total $9 + 6 = 15$ combinations
- Using 3-fold cross validation
- Grand total of $15 \times 3 = 45$ rounds

| Grid Search

- You can get the full list of hyperparameters available for tuning by looking at `full_pipeline.get_params().keys()`

```
print(str(full_pipeline.get_params().keys())[:1000] + "...")
```

| Grid Search

- you can get the best combination of parameters like this

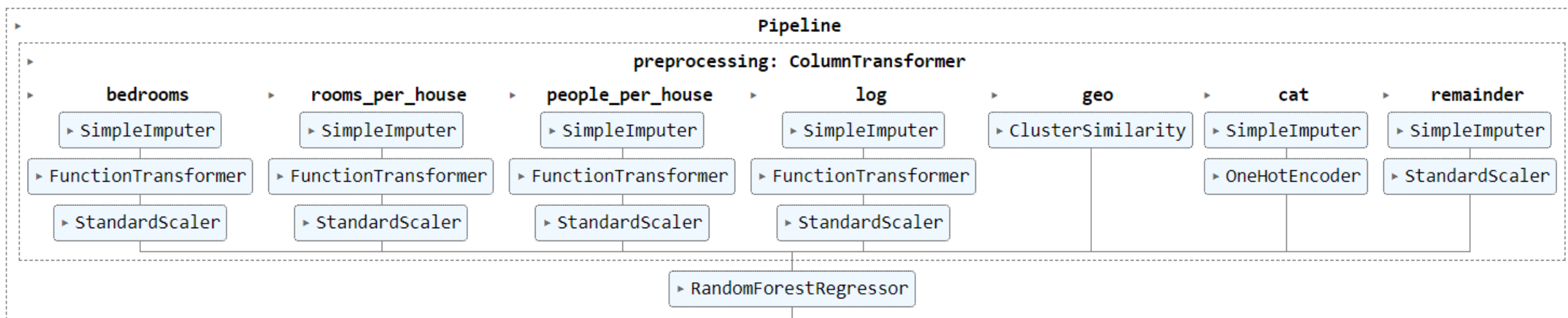
```
grid_search.best_params_
```

```
{'preprocessing__geo__n_clusters': 15, 'random_forest__max_features': 6}
```

| Grid Search

- You can access the best estimator using `grid_search.best_estimator_`.

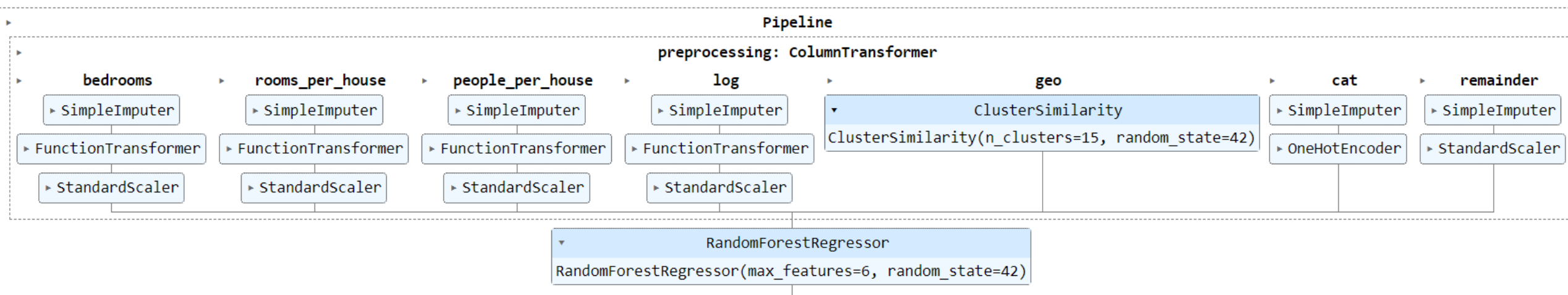
```
grid_search.best_estimator_
```



| Grid Search

- You can access the best estimator using `grid_search.best_estimator_`.

```
grid_search.best_estimator_
```



| Grid Search

- If GridSearchCV is initialized with **refit=True** (which is the default), then once it finds the best estimator using cross-validation, it retrains it on the whole training set. This is usually a good idea, since feeding it more data will likely improve its performance

| Grid Search

- The evaluation scores are available using `grid_search.cv_results_`

```
cv_res = pd.DataFrame(grid_search.cv_results_)
cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)

# extra code - these few lines of code just make the DataFrame look nicer
cv_res = cv_res[["param_preprocessing_geo__n_clusters",
                 "param_random_forest__max_features", "split0_test_score",
                 "split1_test_score", "split2_test_score", "mean_test_score"]]
score_cols = ["split0", "split1", "split2", "mean_test_rmse"]
cv_res.columns = ["n_clusters", "max_features"] + score_cols
cv_res[score_cols] = -cv_res[score_cols].round().astype(np.int64)

cv_res.head()
```

	n_clusters	max_features	split0	split1	split2	mean_test_rmse
12	15	6	43460	43919	44748	44042
13	15	8	44132	44075	45010	44406
14	15	10	44374	44286	45316	44659
7	10	6	44683	44655	45657	44999
9	10	6	44683	44655	45657	44999

| Grid Search

- The mean test RMSE score for the best model is 44,042, which is better than the score you got earlier using the default hyperparameter values (which was 47,019). Congratulations, you have successfully fine-tuned your best model