Kaggle Challenge - Fine Tune Parameters

WE'LL COVER THE FOLLOWING

- 5. Fine-Tune Your Model
  - Grid Search
    - Evaluate Using the Fine-Tuned Model
  - Some More Ways to Perform Fine-Tuning
    - 1. Randomized Search
    - 2. Ensemble Methods
  - Jupyter Notebook

5. Fine-Tune Your Model

Say our best performing model was the RandomForestRegressor. This is a model that has many input hyperparameters that can be tweaked for improving performance. For example, we could have a forest with 100 or 1000 trees, or we could use 10 or 50 features during random selection. What are the best values for these hyperparameters to pass as input to the model for training?

Grid Search

Should we fiddle with all the possible values manually and then compare results to find the best combination of hyperparameters? This would be really tedious work, and we would end up exploring only a few possible combinations.

Luckily, we can use Scikit-learn's `GridSearchCV` to do this tedious search work for us. All we need to do is tell it which hyperparameters we would like to explore and which values to try out, and it will evaluate all the possible combinations of hyperparameter values, using cross-validation.

For example, let’s see how to search for the best combination of hyperparameter values for the RandomForestRegressor:

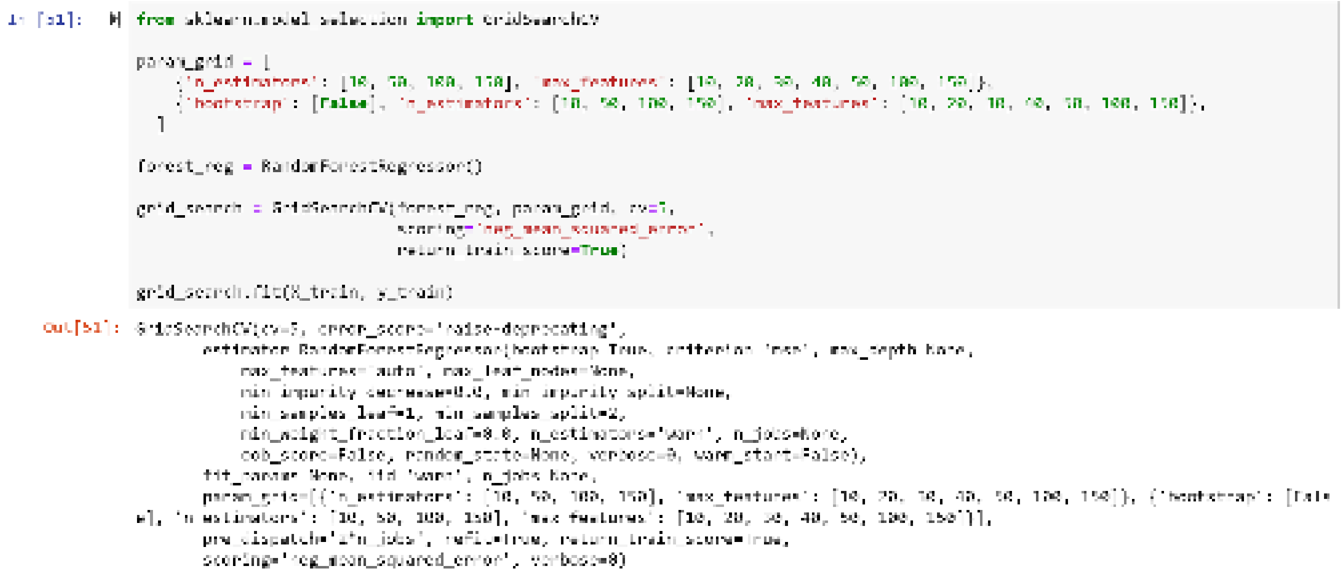
```
from sklearn.model_selection import GridSearchCV

# Define the parameters for exploration
param_grid = [
    {'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 40, 50, 100, 150]},
    {'bootstrap': [False], 'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 4
  ]

# The model for which we are finding params values
forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(X_train, y_train)
```



In [51]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 40, 50, 100, 150]},
    {'bootstrap': [False], 'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 40, 50, 100, 150]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(X_train, y_train)
```

Out[51]:

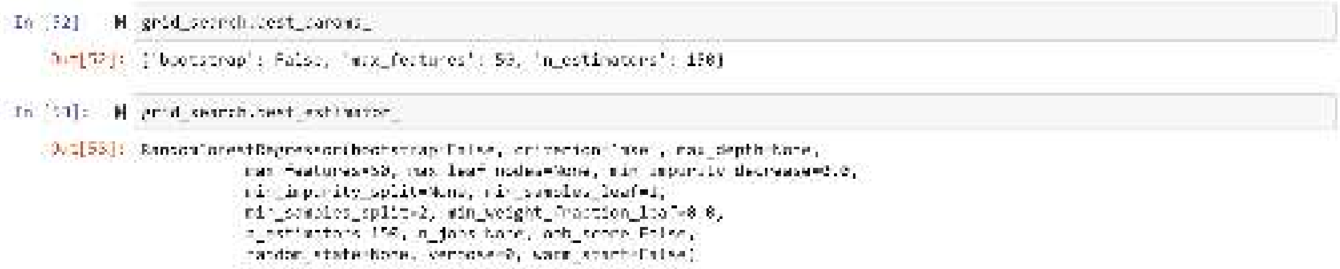
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 40, 50, 100, 150]}, {'bootstrap': [Fals
e], 'n_estimators': [10, 50, 100, 150], 'max_features': [10, 20, 30, 40, 50, 100, 150]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

> **Note**: If we are clueless about which value a hyperparameter should have, a simple approach is to pass consecutive powers of 10, or a smaller number if you want a more fine-grained search to GridSearch.

We can use `best_params_` to visualize the best values for the passed hyperparameters, and `best_estimator_` to get the fine-tuned model:

```
# Best values
grid_search.best_params_

# Model with best values
# Model with best values
grid_search.best_estimator_
```



```
In [52]:  grid_search.best_params_
Out[52]: {'bootstrap': False, 'max_features': 50, 'n_estimators': 150}

In [53]:  grid_search.best_estimator_
Out[53]: RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
                    max_features=50, max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=150, n_jobs=None, oob_score=False,
                    random_state=None, verbose=0, warm_start=False)
```

Evaluate Using the Fine-Tuned Model

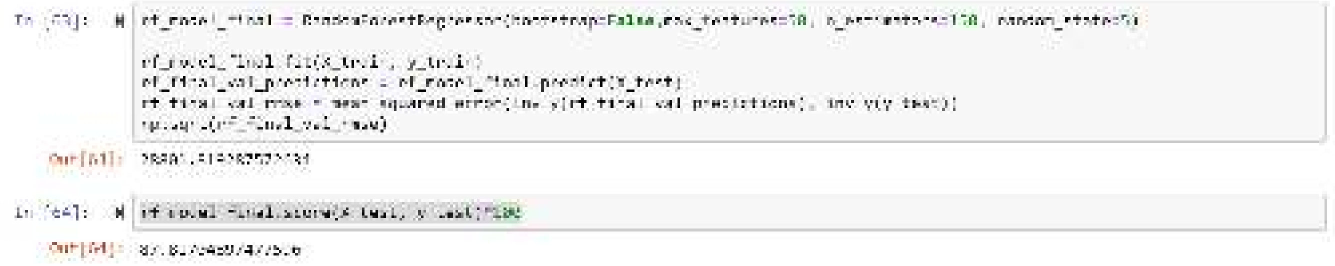
Now that we know the optimal values for the hyperparameters (*'bootstrap': False, 'max_features': 50, 'n_estimators': 150*), let's plug them in and see if our Random Forest model has improved compared to the vanilla Random Forest model that we trained earlier when we trained multiple models at once:

```
rf_model_final = RandomForestRegressor(bootstrap=False,max_features=50, n_estimators=150, rar

rf_model_final.fit(X_train, y_train)
rf_final_val_predictions = rf_model_final.predict(X_test)

# Get RMSE
rf_final_val_rmse = mean_squared_error(inv_y(rf_final_val_predictions), inv_y(y_test))
np.sqrt(rf_final_val_rmse)

# Get Accuracy
rf_model_final.score(X_test, y_test)*100
```



```
In [63]:  rf_model_final = RandomForestRegressor(bootstrap=False,max_features=50, n_estimators=150, random_state=5)

          rf_model_final.fit(X_train, y_train)
          rf_final_val_predictions = rf_model_final.predict(X_test)
          rf_final_val_rmse = mean_squared_error(inv_y(rf_final_val_predictions), inv_y(y_test))
          np.sqrt(rf_final_val_rmse)
Out[63]: 28801.514282572134

In [64]:  rf_model_final.score(X_test, y_test)*100
Out[64]: 87.81754897477536
```

Wow! Our accuracy has gone up from about 84.8 to 87.8 while the RMSE has decreased from 31491 to 28801. This is a significant improvement!

In this example, we obtained a much better solution by setting the max_features hyperparameter to 50, and the n_estimators hyperparameter to 150. However, notice that LinearRegression is still performing better so we are going to choose it as our final model.

Some More Ways to Perform Fine-Tuning

There are many ways to perform fine-tuning. Among which two are discussed

There are many ways to perform fine-tuning. Among which two are discussed here:

1. Randomized Search

The grid search approach is acceptable when we are exploring relatively few combinations, but when the number of combinations of the hyperparameters is large, it is often preferable to use `RandomizedSearchCV`. This is similar to GridSearchCV class, but instead of trying out all possible combinations, it evaluates a given number of random combinations at every iteration.

2. Ensemble Methods

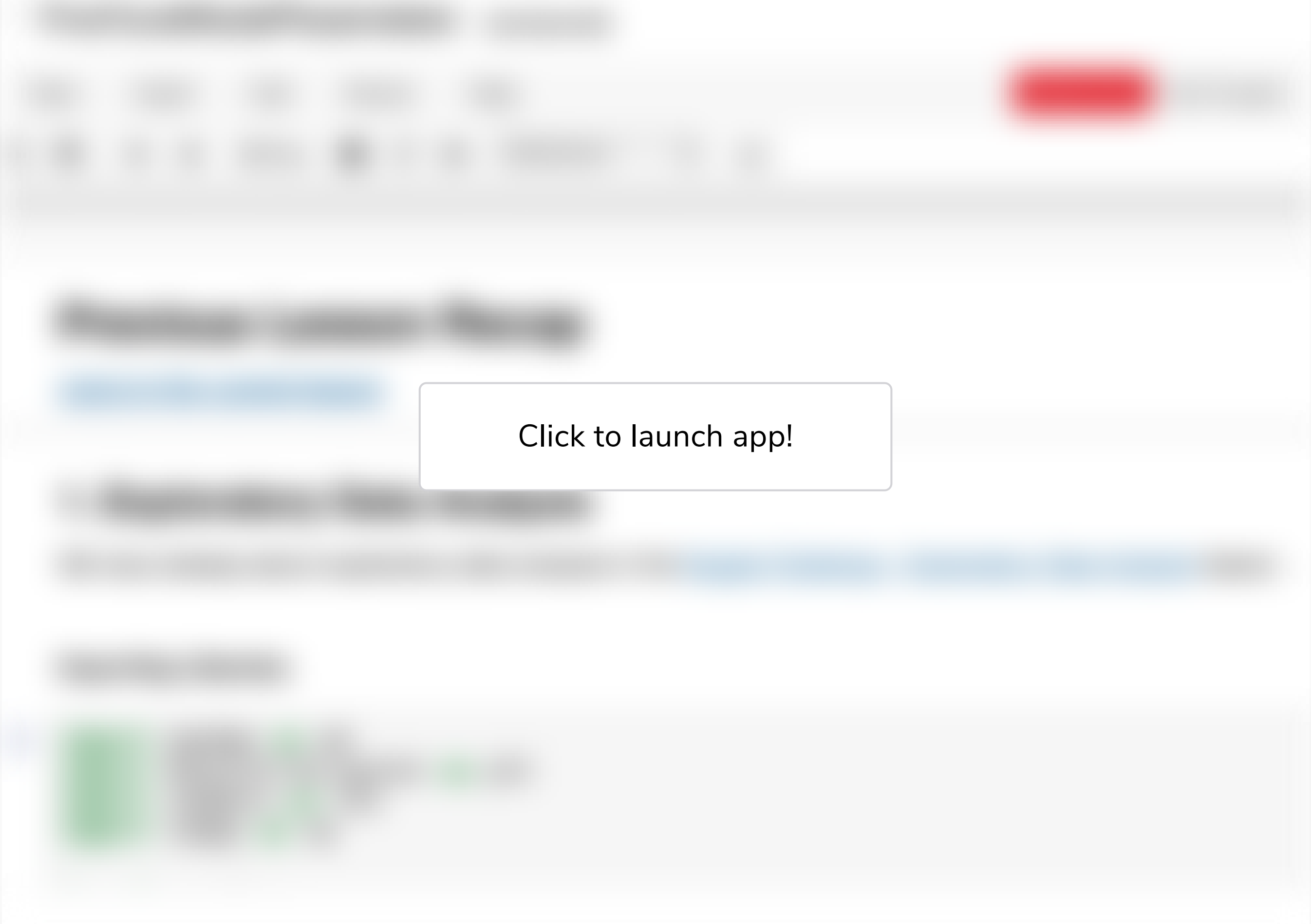
Another way to fine-tune is to try to combine the models that perform best. The group, or ensemble, will often perform better than the best individual model, just like Random Forests perform better than the individual Decision Trees they rely on, especially if the individual models make very different types of errors.

Jupyter Notebook

You can see the instructions running in the Jupyter Notebook below:

> 📝 **How to Use a Jupyter NoteBook?**
>
> - Click on **“Click to Launch”** 🚀 button to work and see the code running live in the notebook.
> - You can click ↗ to open the **Jupyter Notebook in a new tab**.
> - Go to files and click *Download as* and then choose the format of the file to **download** 📥. You can choose Notebook(.ipynb) to download the file and work locally or on your personal Jupyter Notebook.
> - ⚠ The notebook **session expires after 30 minutes of inactivity**. It will reset if there is no interaction with the notebook for 30 consecutive minutes.

Your app can be found at: https://1dgnrwwoynk5m-live-app.educative.run/notebooks/FineTuneModelParameters.ipynb ↗



Click to launch app!