Open the dataset ames housing no missing.csv with the following command:

```
import pandas as pd

ames_housing = pd.read_csv(
    "../datasets/ames_housing_no_missing.csv",
    na_filter=False, # required for pandas>2.0
)

target_name = "SalePrice"

data = ames_housing.drop(columns=target_name)

target = ames_housing[target_name]
```

ames_housing is a pandas dataframe. The column "SalePrice" contains the target variable.

To simplify this exercise, we will only used the numerical features defined below:

```
numerical_features = [
    "LotFrontage", "LotArea", "MasVnrArea", "BsmtFinSF1", "BsmtFinSF2",
    "BsmtUnfSF", "TotalBsmtSF", "1stFlrSF", "2ndFlrSF", "LowQualFinSF",
    "GrLivArea", "BedroomAbvGr", "KitchenAbvGr", "TotRmsAbvGrd",
"Fireplaces",
    "GarageCars", "GarageArea", "WoodDeckSF", "OpenPorchSF",
"EnclosedPorch",
    "3SsnPorch", "ScreenPorch", "PoolArea", "MiscVal",
]

data_numerical = data[numerical_features]
```

We will compare the generalization performance of a decision tree and a linear regression. For this purpose, we will create two separate predictive models and evaluate them by 10-fold cross-validation.

Thus,

use sklearn.linear_model.LinearRegression and sklearn.tree.DecisionTr eeRegressor to create the models. Use the default parameters for the linear regression and set random state=0 for the decision tree.

Be aware that a linear model requires to scale numerical features. Please use sklearn.preprocessing.StandardScaler so that your linear regression model behaves the same way as the quiz author intended;)

Question 1

(1 point possible)

By comparing the cross-validation test scores for both models fold-to-fold, count the number of times the linear model has a better test score than the decision tree model. Select the range which this number belongs to:

a) [0, 3]: the linear model is substantially worse than the decision tree b) [4, 6]: both models are almost equivalent b) [4, 6]: both models are almost equivalent - incorrect c) [7, 10]: the linear model is substantially better than the decision tree

EXPLANATION

solution: c)

The code to get an estimate of the generalization performance of a linear

regression is shown below:

```
from sklearn.model_selection import cross_validate

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
```

```
linear_regression = make_pipeline(StandardScaler(),

LinearRegression())

cv_results_linear_regression = cross_validate(

linear_regression, data_numerical, target, cv=10,

return_estimator=True,

n_jobs=2
)

scores_lr = cv_results_linear_regression["test_score"]

scores_lr
```

We need to preprocess the data with a StandardScaler to scale before to train the linear regressor.

When dealing with decision tree, it is unnecessary to scale the data. So we can directly train our decision tree regressor. The code below shows how to evaluate a decision tree regressor on our regression problem.

```
from sklearn.tree import DecisionTreeRegressor

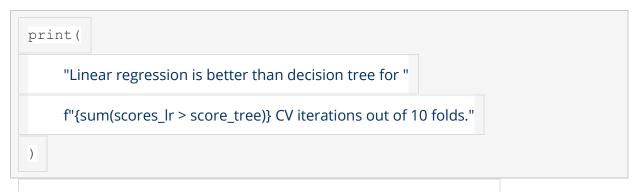
tree = DecisionTreeRegressor(random_state=0)

cv_results_tree = cross_validate(

    tree, data_numerical, target, cv=10, n_jobs=2
)

score_tree = cv_results_tree["test_score"]
```

We can then compare the individual CV scores.



And the linear regression model is better than the tree model.

Linear regression is better than decision tree for 9 CV iterations out of 10 folds.

Hide Answer

You have used 1 of 1 submissions

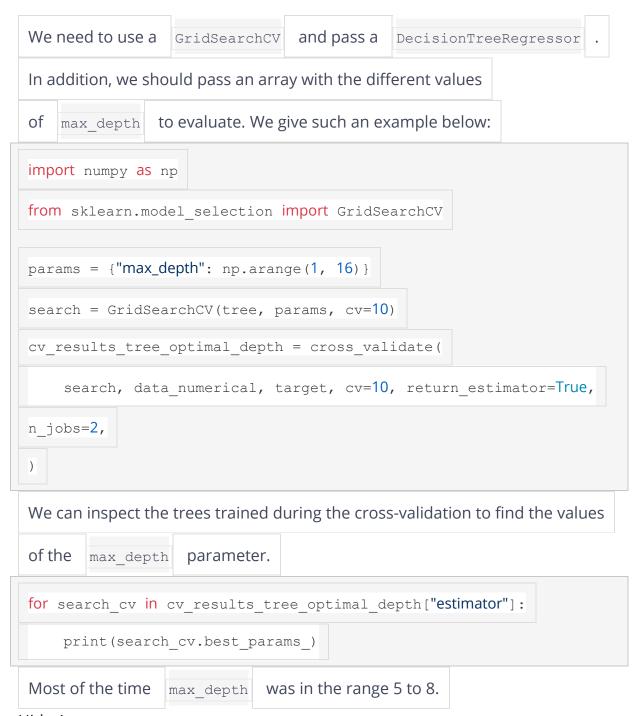
Question 2

(1/1 point)

Instead of using the default parameters for the decision tree regressor, we can optimize the max_depth of the tree. Vary the max_depth from 1 level up to 15
levels. Use nested cross-validation to evaluate a grid-search
(sklearn.model_selection.GridSearchCV). Set cv=10 for both the inner and outer cross-validations, then answer the questions below
What is the optimal tree depth for the current problem?

a) The optimal depth is ranging from 3 to 5 b) The optimal depth is ranging from 5 to 8 b) The optimal depth is ranging from 5 to 8 - correct c) The optimal depth is ranging from 8 to 11 d) The optimal depth is ranging from 11 to 15

EXPLANATION solution: b)



Hide Answer

You have used 1 of 1 submissions

Question 3

(1/1 point)

Now, we want to evaluate the generalization performance of the decision tree while taking into account the fact that we tune the depth for this specific dataset.

Use the grid-search as an estimator inside a $\cross_validate$ to automatically tune the $\cross_validate$ parameter on each cross-validation fold. A tree with tuned depth

a) is always worse than the linear models on all CV folds b) is often but not always worse than the linear model b) is often but not always worse than the linear model - correct c) is often but not always better than the linear model d) is always better than the linear models on all CV folds

Note: Try to set the random_state of the decision tree to different values e.g. random_state=1 or random_state=2 and re-run the nested cross-validation to check that your answer is stable enough.

EXPLANATION

solution: b)

```
search = GridSearchCV(tree, params, cv=10)

cv_results_tree_optimal_depth = cross_validate(

    search, data_numerical, target, cv=10, return_estimator=True,

n_jobs=2,
)

cv_results_tree_optimal_depth["test_score"].mean()
```

We can then check the generalization performance of the model.

```
"A tree with an optimized depth is better than linear regression for "

f"{sum(cv_results_tree_optimal_depth['test_score'] > scores_lr)} CV "

"iterations out of 10 folds."
```

A tree with an optimized depth is better than linear regression for 2 CV

iterations out of 10 folds.

Hide Answer

You have used 1 of 1 submissions

Question 4

(1/1 point)

Instead of using only the numerical features you can now use the entire dataset available in the variable data.

Create a preprocessor by dealing separately with the numerical and categorical columns. For the sake of simplicity, we assume the following:

- categorical columns can be selected if they have an object data type;
- use an OrdinalEncoder to encode the categorical columns;
- numerical columns should correspond to the numerical_features as defined above.

In addition, set the <code>max_depth</code> of the decision tree to [7] (fixed, no need to tune it with a grid-search).

Evaluate this model using <code>cross_validate</code> as in the previous questions.

A tree model trained with both numerical and categorical features

a) is most often worse than the tree model using only the numerical features b) is most often better than the tree model using only the numerical features b) is most often better than the tree model using only the numerical features - correct

Note: Try to set the random_state of the decision tree to different values e.g. random_state=1 or random_state=2 and re-run the (this time single) cross-validation to check that your answer is stable enough.

EXPLANATION

solution: b)

The code to create the predictive model to handle both categorical and

numerical columns is the following:

```
from sklearn.compose import make_column_transformer

from sklearn.compose import make_column_selector as selector

from sklearn.preprocessing import OrdinalEncoder

categorical_processor = OrdinalEncoder(
    handle_unknown="use_encoded_value", unknown_value=-1
)

preprocessor = make_column_transformer(
    (categorical_processor, selector(dtype_include=object)),
    ("passthrough", numerical_features)
)

tree = make_pipeline(preprocessor,

DecisionTreeRegressor(max_depth=7, random_state=0))
```

Then, we can evaluate this tree using cross-validation:

```
cv_results = cross_validate(
    tree, data, target, cv=10, return_estimator=True, n_jobs=2
)

cv_results["test_score"].mean()
import matplotlib.pyplot as plt
```

```
test_score_num = cv_results_tree_optimal_depth["test_score"]
test score all = cv results["test_score"]
indices = np.arange(len(test score num))
plt.scatter(
    indices, test score num, color="tab:blue", label="numerical
features only"
plt.scatter(
    indices,
    test_score_all,
    color="tab:red",
    label="all features",
plt.ylim((0, 1))
plt.xlabel("Cross-validation iteration")
plt.ylabel("R2 score")
 = plt.legend(loc="lower right")
print(
    "A tree model using both numerical and categorical features is better than a "
    "tree with optimal depth using only numerical features for "
    f"{sum(cv_results['test_score'] > cv_results_tree_optimal_depth['test_score'])} CV
```

"iterations out of 10 folds."

)

A tree model using both numerical and categorical features is better than a tree with optimal depth using only numerical features for 7 CV iterations out of 10 folds.