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")

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]
```

Start by fitting a ridge regressor (`sklearn.linear_model.Ridge`) fixing the penalty `alpha` to 0 to not regularize the model. Use a 10-fold cross-validation and pass the
argument `return_estimator=True` in `sklearn.model_selection.cross_valida`

`te` to access all fitted estimators fitted on each fold. As discussed in the previous notebooks, use an instance of `sklearn.preprocessing.StandardScaler` to scale the data before passing it to the regressor.

Question 1

(1/1 point)
How large is the largest absolute value of the weight (coefficient) in this trained model?

a) Lower than 1.0 (1e0) b) Between 1.0 (1e0) and 100,000.0 (1e5) c) Larger than 100,000.0 (1e5) c) Larger than 100,000.0 (1e5) - correct
Hint: Note that the estimator fitted in each fold of the cross-validation procedure is a pipeline object. To access the coefficients of the `Ridge` model at the last position in a pipeline object, you can use the expression `pipeline[-1].coef_` for each pipeline object fitted in the cross-validation procedure. The `-1` notation is a negative index meaning "last position".

EXPLANATION

solution: c)

The following code creates a predictive pipeline using a linear regression as a predictor. It is then evaluated using cross-validation evaluation. The coefficients can be found by inspecting the last step of each fitted pipeline stored in the key `"estimator"` from the dictionary returned by `cross_validate` .

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate
```

```
model = make_pipeline(StandardScaler(), Ridge(alpha=0))
cv_results = cross_validate(
    model, data_numerical, target, cv=10, return_estimator=True
)
coefs = [pipeline[-1].coef_ for pipeline in cv_results["estimator"]]
coefs = pd.DataFrame(coefs, columns=numerical_features)
coefs.describe().loc[["min", "max"]]
```

Here, we use `coefs.describe()` to compute the minimum and maximum but notice that this is not the only possible solution.

Sometimes using models without regularization (e.g. `Ridge` with `alpha=0` or `LinearRegression`) can be problematic. The problem to be solved can be ill-conditioned and the coefficients can be very large (e.g. ~1e18). This is due to some numerical errors and we should not use this model in practice. A model adding some regularization should be used instead.

For visually inspecting the coefficients, you could use a boxplot:

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_context("talk")
# Define the style of the box style
boxplot_property = {
```

```
    "vert": True,
    "whis": 100,
    "patch_artist": True,
    "widths": 0.5,
    "rot": 90,
    "boxprops": dict(linewidth=3, color="black", alpha=0.9),
    "medianprops": dict(linewidth=2.5, color="black", alpha=0.9),
    "whiskerprops": dict(linewidth=3, color="black", alpha=0.9),
    "capprops": dict(linewidth=3, color="black", alpha=0.9),
}

_, ax = plt.subplots(figsize=(15, 10))
_ = coefs.plot.box(**boxplot_property, ax=ax)
```

This plot gives us the information that some coefficients are really large (outlier dots on the plot). Notice that the scale of the y axis is huge (1e19) and the "bodies" of the boxplots are shrunk seemingly close to zero and we can only visualize the numerically unstable outliers.

Hide Answer

You have used 1 of 1 submissions

Question 2

(1/1 point)

Repeat the same experiment by fitting a ridge regressor (`sklearn.linear_model.Ridge`) with the default parameter (i.e. `alpha=1.0`). How large is the largest absolute value of the weight (coefficient) in this trained model?

a) Lower than 1.0 b) Between 1.0 and 100,000.0 b) Between 1.0 and 100,000.0 - correct c) Larger than 100,000.0

EXPLANATION

solution: b)

We can repeat the experiment using a ridge regressor. The coefficients can be inspected in the same manner.

```
from sklearn.linear_model import Ridge

model = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
cv_results = cross_validate(
    model, data_numerical, target, cv=10, return_estimator=True
)

coefs = [pipeline[-1].coef_ for pipeline in cv_results["estimator"]]
coefs = pd.DataFrame(coefs, columns=numerical_features)
coefs.describe().loc[["min", "max"]]
```

And we can also make a plot of the variability of the coefficients with a boxplot:

```
_, ax = plt.subplots(figsize=(15, 10))
_ = coefs.abs().plot.box(**boxplot_property, ax=ax)
```

The regularization shrinks the coefficients towards zero. It avoids the numerical errors. In this case, the extremum is around 20,000.

Hide Answer

You have used 1 of 1 submissions

Question 3

(1 point possible)

What are the two most important features used by the ridge regressor? You can make a box-plot of the coefficients across all folds to get a good insight.

a) `"MiscVal"` and `"BsmtFinSF1"` a) <code>"MiscVal"</code> and <code>"BsmtFinSF1"</code> - incorrect b) `"GarageCars"` and `"GrLivArea"` c) `"TotalBsmtSF"` and `"GarageCars"`

EXPLANATION

solution: b)

We can reuse the previous boxplot to answer the question:

```
_, ax = plt.subplots(figsize=(15, 10))
_ = coefs.abs().plot.box(**boxplot_property, ax=ax)
```

Hide Answer

You have used 1 of 1 submissions

Question 4

(1 point possible)

Remove the feature `"GarageArea"` from the dataset and repeat the previous experiment.

What is the impact on the weights of removing `"GarageArea"` from the dataset?

a) None b) Completely changes the order of the most important features c) Decreases the standard deviation (across CV folds) of the `"GarageCars"` coefficient

b) Completely changes the order of the most important features, - incorrect

Select all answers that apply

EXPLANATION

solution: c)

Indeed, we should look at the variability of the `"GarageCars"` coefficient during the experiment. In the previous plot, we could see that the coefficients related to this feature were varying from one fold to another. We can check the standard deviation of the coefficients and check the evolution.

```
coefs["GarageCars"].std()
```

Let's drop the column `"GarageArea"` .

```
column_to_drop = "GarageArea"
data_numerical = data_numerical.drop(columns=column_to_drop)

cv_results = cross_validate(
    model, data_numerical, target, cv=10, return_estimator=True
)
coefs = [pipeline[-1].coef_ for pipeline in cv_results["estimator"]]
coefs = pd.DataFrame(coefs, columns=data_numerical.columns)
coefs["GarageCars"].std()
```

The standard deviation decreases a lot.

Hide Answer

You have used 2 of 2 submissions

Question 5

(1 point possible)

What is the main reason for observing the previous impact on the most important weight(s)?

a) Both garage features are correlated and are carrying similar information b) Removing the "GarageArea" feature reduces the noise in the dataset b) Removing the "GarageArea" feature reduces the noise in the dataset - incorrect c) Just some random effects

EXPLANATION

solution: a)

The number of cars that can fit in the garage is indeed strongly dependent on the area of the garage. This could be checked by computing a correlation coefficient (e.g. the Pearson, Spearman or Kendall correlation coefficients) between the two columns.

Correlated features typically cause unstable estimation of the the matching linear model coefficients, even with some level of regularization. As a result we can expect comparatively larger standard deviations of their coefficients when the two correlated features are included in the linear model.

There is no reason that the measurement of the garage area would be more noisy than most other features.

One way to check the above analysis holds would be to drop the "GarageCars" feature instead of "GarageArea" and check that the coefficient is of "GarageArea" gets to the most important in magnitude along with a small standard deviation.

Hide Answer

You have used 1 of 1 submissions

Question 6

(1 point possible)

Now, we search for the regularization strength that maximizes the generalization performance of our predictive model. Fit a `sklearn.linear_model.RidgeCV` instead of a `Ridge` regressor on the numerical data without the `"GarageArea"` column.

Pass `alphas=np.logspace(-3, 3, num=101)` to explore the effect of changing the regularization strength.

What is the effect of tuning `alpha` on the variability of the weights of the feature `"GarageCars"`? Remember that the variability can be assessed by computing the standard deviation.

a) The variability does not change after tuning `alpha` b) The variability decreased after tuning `alpha` c) The variability increased after tuning `alpha` c) The variability increased after tuning <code>alpha</code> - incorrect

EXPLANATION

solution: b)

We only need to repeat the previous experiment by changing the final regressor.

```
import numpy as np
from sklearn.linear_model import RidgeCV

alphas = np.logspace(-3, 3, num=101)
model = make_pipeline(StandardScaler(), RidgeCV(alphas=alphas))
cv_results_num_only = cross_validate(
    model, data_numerical, target, cv=10, return_estimator=True
```

```
)
test_score_num_only = cv_results_num_only["test_score"]  # save it for
later

coefs = [pipeline[-1].coef_ for pipeline in
cv_results_num_only["estimator"]]
coefs = pd.DataFrame(coefs, columns=data_numerical.columns)
coefs["GarageCars"].std()
```

We see that the magnitude of the standard deviation decreased slightly; it could mean that our model chose a stronger regularization parameter than the default value in `Ridge` .

Hide Answer

You have used 1 of 1 submissions

Question 7

(1 point possible)

Check the parameter `alpha_` (the regularization strength) for the different ridge regressors obtained on each fold.

In which range does `alpha_` fall into for most folds?

a) between 0.1 and 1 b) between 1 and 10 b) between 1 and 10 - incorrect c) between 10 and 100 d) between 100 and 1000

EXPLANATION

solution: d)

To find the answer, we need to check the value of `alpha_` for the different folds.

```
tuned_alphas = [pipeline[-1].alpha_ for pipeline in
cv_results_num_only["estimator"]]
tuned_alphas
```

In this case, we see that most `alpha_` are larger than 100 and more precisely they are contained in the range 250-400, except for a single fold leading to a particularly low value for the optimal `alpha_` .

Hide Answer

You have used 1 of 1 submissions

So far we only used the list of `numerical_features` to build the predictive model. Now create a preprocessor to deal separately with the numerical and categorical columns:

- categorical features can be selected if they have an `object` data type;
- use an `OneHotEncoder` to encode the categorical features;
- numerical features should correspond to the `numerical_features` as defined above. This is a subset of the features that are not an `object` data type;
- use an `StandardScaler` to scale the numerical features.

The last step of the pipeline should be a `RidgeCV` with the same set of `alphas` to evaluate as previously.

Question 8

(1 point possible)

By comparing the cross-validation test scores fold-to-fold for the model with `numerical_features` only and the model with both `numerical_features` and `categorical_features`, count the number of times the simple model has a better test score than the model with all features. Select the range which this number belongs to:

a) [0, 3]: the simple model is consistently worse than the model with all features b) [4, 6]: both models are almost equivalent c) [7, 10]: the simple model is consistently better than the model with all features c) [7, 10]: the simple model is consistently better than the model with all features - incorrect

EXPLANATION

solution: a)

To find the answer, we need to check the test scores of both model for the different folds.

```
from sklearn.compose import make_column_selector as selector
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

categorical_features = selector(dtype_include=object)(data)
numerical_features.remove("GarageArea")

preprocessor = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore"), categorical_features),
    (StandardScaler(), numerical_features),
)
model = make_pipeline(preprocessor, RidgeCV(alphas=alphas))
cv_results_num_and_cat = cross_validate(
    model, data, target, cv=10, n_jobs=2
)
test_score_num_and_cat = cv_results_num_and_cat["test_score"]

indices = np.arange(len(test_score_num_only))
plt.scatter(
    indices,
    test_score_num_only,
```

```
    color="tab:blue",
    label="numerical features only"
)
plt.scatter(
    indices,
    test_score_num_and_cat,
    color="tab:red",
    label="all features",
)
plt.ylim((0, 1))
plt.xlabel("Cross-validation iteration")
plt.ylabel("R2 score")
_ = plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")

print(
    "A model using both numerical and categorical features is better than a"
    " model using only numerical features for"
    f" {sum(test_score_num_and_cat > test_score_num_only)} CV iterations out of
10."
)
```

Hide Answer

You have used 1 of 1 submissions

In this Module we saw that non-linear feature engineering may yield a more predictive pipeline, as long as we take care of adjusting the regularization to avoid overfitting.

Try this approach by building a new pipeline similar to the previous one but replacing the `StandardScaler` by a `SplineTransformer` (with default hyperparameter values) to better model the non-linear influence of the numerical features.

Furthermore, let the new pipeline model feature interactions by adding a new `Nystroem` step between the preprocessor and the `RidgeCV` estimator. Set `kernel="poly"`, `degree=2` and `n_components=300` for this new feature engineering step.

Question 9

(1 point possible)
By comparing the cross-validation test scores fold-to-fold for the model with both `numerical_features` and `categorical_features`, and the model that performs non-linear feature engineering; count the number of times the non-linear pipeline has a better test score than the model with simpler preprocessing. Select the range which this number belongs to:
a) [0, 3]: the new non-linear pipeline is consistently worse than the previous pipeline a) [0, 3]: the new non-linear pipeline is consistently worse than the previous pipeline - incorrect b) [4, 6]: both models are almost equivalent c) [7, 10]: the new non-linear pipeline is consistently better than the previous pipeline

EXPLANATION

solution: c)

To find the answer we can do something similar as for the last question:

```
from sklearn.kernel_approximation import Nystroem
from sklearn.preprocessing import SplineTransformer

preprocessor = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore"), categorical_features),
    (SplineTransformer(), numerical_features),
```

```
)

model_with_interaction = make_pipeline(
    preprocessor,
    Nystroem(kernel="poly", degree=2, n_components=300),
    RidgeCV(alphas=alphas)
)
cv_results_interactions = cross_validate(
    model_with_interaction,
    data,
    target,
    cv=10,
    n_jobs=2,
)
test_score_interactions = cv_results_interactions["test_score"]

plt.scatter(
    indices,
    test_score_num_only,
    color="tab:blue",
    label="numerical features only"
)
plt.scatter(
    indices,
    test_score_num_and_cat,
```

```
    color="tab:red",
    label="all features",
)
plt.scatter(
    indices,
    test_score_interactions,
    color="black",
    label="all features and interactions",
)
plt.ylim((0, 1))
plt.xlabel("Cross-validation iteration")
plt.ylabel("R2 Score")
_ = plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")

print(
    "A model using all features with non-linear feature engineering is better"
    "  than the previous pipeline for"
    f" {sum(test_score_interactions > test_score_num_and_cat)} CV iterations"
    " out of 10."
)
```

Notice that `Nystroem` is a randomized preprocessing step. Therefore the results might vary a bit when rerunning the cross-validation of the pipeline if the `random_state` is not specified. However, this should not change the

solution to the quiz question because the model with interactions is indeed significantly better. Confirm this by reloading the previous cells several times.

Alternatively we could have used an explicit polynomial expansion such as `PolynomialFeatures(degree=2, interactions_only=True)` to model interactions between features instead of using the `Nystroem(kernel="poly", degree=2, n_components=300)` approximation above.

However the `PolynomialFeatures` would generate significantly more intermediate features than the fixed 300 features of `Nystroem` for this dataset: this would make the pipeline really slow to execute, use much more memory and possibly overfit more in the end.