Intermediate Machine Learning Home Page (https://www.kaggle.com/learn/intermediate-machine-learning)

This exercise involves you writing code, and we check it automatically to tell you if it's right. We're having a temporary problem with out checking infrastructure, causing a bar that says `None` *in some cases when you have the right answer. We're sorry. We're fixing it. In the meantime, if you see a bar saying* `None` *that means you've done something good.*

In this exercise, you will use your new knowledge to train a model with **gradient boosting**.

Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

In [1]:
```
# Set up code checking
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv", "../inpu
t/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv", "../input/
test.csv")
from learntools.core import binder
binder.bind(globals())
from learntools.ml_intermediate.ex6 import *
print("Setup Complete")
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarn
ing: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[04:08:38] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Setup Complete
```

Exercise: XGBoost

Python notebook using data from Housing Prices Competition for Kaggle Learn Users · 0 views · 3m ago · Edit tags

0 Access Edit

Ver
1
fork

data-for-ml-course) dataset from the previous exercise.

Run the next code cell without changes to load the training and validation sets in `X_train`, `X_valid`, `y_train`, and `y_valid`. The test set is loaded in `X_test`.

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X.SalePrice
X.drop(['SalePrice'], axis=1, inplace=True)

# Break off validation set from training data
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X, y,
train_size=0.8, test_size=0.2,
                                                                random
_state=0)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenien
t but arbitrary)
low_cardinality_cols = [cname for cname in X_train_full.columns if X_t
rain_full[cname].nunique() < 10 and
                        X_train_full[cname].dtype == "object"]

# Select numeric columns
numeric_cols = [cname for cname in X_train_full.columns if X_train_ful
l[cname].dtype in ['int64', 'float64']]

# Keep selected columns only
my_cols = low_cardinality_cols + numeric_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
X_test = X_test_full[my_cols].copy()

# One-hot encode the data (to shorten the code, we use pandas)
X_train = pd.get_dummies(X_train)
X_valid = pd.get_dummies(X_valid)
X_test = pd.get_dummies(X_test)
```

Notebook Data Comments

Step 1: Build model

In this step, you'll build and train your first model with gradient boosting.

- Begin by setting `my_model_1` to an XGBoost model. Use the XGBRegressor (https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBRegressor) class, and set the random seed to 0 (`random_state=0`). **Leave all other parameters as default.**
- Then, fit the model to the training data in `X_train` and `y_train`.

In [3]:
```
from xgboost import XGBRegressor

# Define the model
```

```
my_model_1 = XGBRegressor(random_state=0)

# Fit the model
my_model_1.fit(X_train, y_train) # Your code here

# Check your answer
step_1.a.check()
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarn
ing: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \


[04:08:40] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
```

Correct

In [4]:
```
# Lines below will give you a hint or solution code
#step_1.a.hint()
#step_1.a.solution()
```

Set `predictions_1` to the model's predictions for the validation data. Recall that the validation features are stored in `X_valid`.

In [5]:
```
from sklearn.metrics import mean_absolute_error

# Get predictions
predictions_1 = my_model_1.predict(X_valid) # Your code here

# Check your answer
step_1.b.check()
```

Correct

In [6]:
```
# Lines below will give you a hint or solution code
#step_1.b.hint()
#step_1.b.solution()
```

Finally, use the `mean_absolute_error()` function to calculate the mean absolute error (MAE) corresponding to the predictions for the validation set. Recall that the labels for the validation data are stored in `y_valid`.

In [7]:

```
# Calculate MAE
mae_1 = mean_absolute_error(y_valid, predictions_1) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_1)

# Check your answer
step_1.c.check()
```

```
Mean Absolute Error: 16803.434690710616
```

Correct

In [8]:

```
# Lines below will give you a hint or solution code
#step_1.c.hint()
#step_1.c.solution()
```

Step 2: Improve the model

Now that you've trained a default model as baseline, it's time to tinker with the parameters, to see if you can get better performance!

- Begin by setting `my_model_2` to an XGBoost model, using the XGBRegressor (https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBRegressor) class. Use what you learned in the previous tutorial to figure out how to change the default parameters (like `n_estimators` and `learning_rate`) to get better results.
- Then, fit the model to the training data in `X_train` and `y_train`.
- Set `predictions_2` to the model's predictions for the validation data. Recall that the validation features are stored in `X_valid`.
- Finally, use the `mean_absolute_error()` function to calculate the mean absolute error (MAE) corresponding to the predictions on the validation set. Recall that the labels for the validation data are stored in `y_valid`.

In order for this step to be marked correct, your model in `my_model_2` must attain lower MAE than the model in `my_model_1`.

In [9]:

```
# Define the model
my_model_2 =XGBRegressor(n_estimators=1000, learning_rate=0.05)

# Fit the model
my_model_2.fit(X_train, y_train) # Your code here

# Get predictions
predictions_2 = my_model_2.predict(X_valid) # Your code here

# Calculate MAE
mae_2 = mean_absolute_error(y_valid, predictions_2) # Your code here
```

```
mae_2 = mean_absolute_error(y_valid, predictions_2) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_2)

# Check your answer
step_2.check()
```

```
[04:08:41] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.


/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarn
ing: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \


Mean Absolute Error: 16084.123354559075
```

Correct

In [10]:
```
# Lines below will give you a hint or solution code
#step_2.hint()
#step_2.solution()
```

Step 3: Break the model

In this step, you will create a model that performs worse than the original model in Step 1. This will help you to develop your intuition for how to set parameters. You might even find that you accidentally get better performance, which is ultimately a nice problem to have and a valuable learning experience!

- Begin by setting `my_model_3` to an XGBoost model, using the XGBRegressor (https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBRegressor) class. Use what you learned in the previous tutorial to figure out how to change the default parameters (like `n_estimators` and `learning_rate`) to design a model to get high MAE.
- Then, fit the model to the training data in `X_train` and `y_train`.
- Set `predictions_3` to the model's predictions for the validation data. Recall that the validation features are stored in `X_valid`.
- Finally, use the `mean_absolute_error()` function to calculate the mean absolute error (MAE) corresponding to the predictions on the validation set. Recall that the labels for the validation data are stored in `y_valid`.

In order for this step to be marked correct, your model in `my_model_3` must attain higher MAE than the model in `my_model_1`.

In [11]:
```
# Define the model
my_model_3 = XGBRegressor(n_estimators=1)
```

```
# Fit the model
my_model_3.fit(X_train, y_train) # Your code here

# Get predictions
predictions_3 = my_model_3.predict(X_valid)

# Calculate MAE
mae_3 = mean_absolute_error(predictions_3, y_valid)

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_3)

# Check your answer
step_3.check()
```

```
[04:08:49] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Mean Absolute Error: 163382.59044574058


/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarn
ing: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

Correct

In [12]:
```
# Lines below will give you a hint or solution code
#step_3.hint()
step_3.solution()
```

Solution:

```
# Define the model
my_model_3 = XGBRegressor(n_estimators=1)

# Fit the model
my_model_3.fit(X_train, y_train)

# Get predictions
predictions_3 = my_model_3.predict(X_valid)

# Calculate MAE
mae_3 = mean_absolute_error(predictions_3, y_valid)
print("Mean Absolute Error:" , mae_3)
```

Did you find this Kernel useful?

0

Show your appreciation with an upvote

Data

Data Sources

Housing Prices Competition for Kaggle Learn Users

sample_submission.csv	1459 x 2
test.csv	1459 x 80
train.csv	1460 x 81

Housing Prices Competition for Kaggle Learn Users

Apply what you learned in the Machine Learning course on Kaggle Learn alongside others in the course.

Last Updated: a year ago

About this Competition

File descriptions

- **train.csv** - the training set
- **test.csv** - the test set
- **data_description.txt** - full description of each column, originally prepared by Dean De Cock but lightly edited to match the column names used here
- **sample_submission.csv** - a benchmark submission from a linear regression on year and month of sale, lot square footage, and number of bedrooms

Data fields

Here's a brief version of what you'll find in the data description file.

- **SalePrice** - the property's sale price in dollars. This is the target variable that you're trying to predict.
- **MSSubClass**: The building class
- **MSZoning**: The general zoning classification
- **LotFrontage**: Linear feet of street connected to property
- **LotArea**: Lot size in square feet
- **Street**: Type of road access
- **Alley**: Type of alley access
- **LotShape**: General shape of property
- **LandContour**: Flatness of the property

Comments (0)

Click here to comment...