Untitled

June 17, 2020

1 Extreme Gradient Boosting with XGBoost

Do you know the basics of supervised learning and want to use state-of-the-art models on real-world datasets? Gradient boosting is currently one of the most popular techniques for efficient modeling of tabular datasets of all sizes. XGboost is a very fast, scalable implementation of gradient boosting, with models using XGBoost regularly winning online data science competitions and being used at scale across different industries. Here I'll use this powerful library alongside pandas and scikit-learn to build and tune supervised learning models. I'll work with real-world datasets to solve classification and regression problems.

Classification with XGBoost

Here I will introduce the fundamental idea behind XGBoost—boosted learners. I'll apply it to solve a common classification problem found in industry: predicting whether a customer will stop being a customer at some point in the future.

In order to understand XGBoost we need to have some handle on the broader topics of supervised classification, decision trees, and boosting. Supervised learning is a kind of learning problems that XGBoost can be applied to,relies on labeled data.That is we have some understanding of the past behavior of the problem we are trying to solve or what we are trying to predict. For example, assessing whether a specific image contains a person's face is a classification problem. In this case, the training data are images converted into vectors of pixel values, and the labels are either 1 when the image contains a face or 0 when the image doesn't contain a face. Given this, there are two kinds of supervised learning problems that account for the vast majority of use-cases: classification problems and regression problems. Classification problems involve predicting either binary or multi-class outcomes. For example, predicting whether a person will purchase an insurance package given some quote is a binary supervised learning problem and predicting whether a picture contains one of several species of birds is a multi-class supervised learning problems.

When dealing with binary supervised learning problems, the AUC or Area Under the Receive Operating Characteristic Curve is the most versatile and common evaluation metric used to judge the quality of a binary classification model. It is simply the probability that a randomly chosen positive data point will have a higher rank than a randomly chosen negative data point for your learning problem. So, a higher AUC means a more sensitive better performing model.

When dealing with multi-class classification problems it is common to use the accuracy score (higher is better) and to look at the overall confusion matrix to evaluate the quality of a model.

Some common algorithms for classification problems include logistic regression, and decision trees. All supervised learning problems including classification problems require that the data is structured as a table of feature vectors, where the features themselves (also called attributes or predictors) are

either numeric or categorical. Furthermore, it is usually the case that numeric features are scaled to aid in either feature interpretation or to ensure that the model can be trained properly for example, numerical feature scaling is essential to ensure properly trained support vector machine models. Categorical features are also almost always encoded before applying supervised learning algorithms, most commonly using one-hot encoding. Some other kinds of supervised learning problems include: ranking problems that involve predicting an ordering on a set of choices like google search suggestions. Recommendations problems involve recommending an item or a set of items to a user based on his/her consumption history and profile like Netflix.

Differentiating a classification problem

a - Given past performance of stocks and various other financial data, predicting the exact price of a given stock (Google) tomorrow.

This is an example of a regression problem, because we are predicting a continuous quantity.

b - Given a large dataset of user behaviors on a website, generating an informative segmentation of the users based on their behaviors.

There's nothing to predict here, this is an unsupervised (clustering) problem.

c - Predicting whether a given user will click on an ad given the ad content and metadata associated with the user.

This is a classification problem.

d - Given a user's past behavior on a video platform, presenting him/her with a series of recommended videos to watch next.

This problem involves ranking entities and returning the highest ranked ones (in order) to the user.

Which of these is a binary classification problem?

A classification problem involves predicting the category a given data point belongs to out of a finite set of possible categories. Depending on how many possible categories there are to predict, a classification problem can be either binary or multi-class

a - Predicting whether a given image contains a cat.

This is a binary problem. A binary classification problem involves picking between 2 choices.

b - Predicting the emotional valence of a sentence (Valence can be positive, negative, or neutral).

This is not a binary problem as there are 3 categories to choose from here.

c- Recommending the most tax-efficient strategy for tax filing in an automated accounting system.

This smells like a recommendation problem, not a classification problem.

d - Given a list of symptoms, generating a rank-ordered list of most likely diseases.

This is also a recommendation problem.

Introducing XGBoost

XGBoost is the hottest library in supervised machine learning. It was designed originally as a C++ command line application. Bindings or functions that tapped into the core C++ code started

appearing in a variety of other languages including Python, R, Scala, Java, and Julia. XGBoost is popular for its speed and performance.

The core XGBoost algorightm is parallelizable. It can harness all of the processing power of modern multi-core computers. It is also parallelizable onto GPU's and across networks of computers making it feasible to train models on very large datasets on the order of hundreds of millions of training examples.

What makes XGBoost so popular is that it consistently outperforms almost all other single algorithms methods in machine learning competitions. We can use XGBoost using a classification problem.

When we create X and y, we split the entire dataset into a matrix of samples by features called X by convention and a vector of target values called y by convention.

Applying XGBoost on Churn_data to classify the customers who will churn and who will remain with the company.

[28]:
```
import pandas as pd
file_path = '/Users/MuhammadBilal/Desktop/Data Camp/Extreme Gradient Boosting␣
 ↪with XGBoost/Data/churn_data.csv'
churn_data = pd.read_csv(file_path)
```

[23]:
```
import pandas as pd
from sklearn_pandas import DataFrameMapper
from sklearn.impute import SimpleImputer
from sklearn_pandas import CategoricalImputer
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.feature_extraction import DictVectorizer
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
import numpy as np
```

[16]:
```
# Replacing 'no' with 0 and 'yes' with 1 in 'Vmail_Plan'
churn_data['Vmail_Plan'] = churn_data['Vmail_Plan'].replace({'no':0, 'yes':1})

# Replacing 'no' with 0 and 'yes' with 1 in 'Churn'
churn_data['Churn'] = churn_data['Churn'].replace({'no':0, 'yes':1})

# Replace 'no' with 0 and 'yes' with 1 in 'Intl_Plan'
churn_data['Intl_Plan'] = churn_data['Intl_Plan'].replace({'no':0, 'yes':1})
```

[17]:
```
# Performing one hot encoding on 'State'
churn_state = pd.get_dummies(churn_data['State'])

# Printing the head of telco_state
print(churn_state.head())
```

```
   AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL  …  SD  TN  TX  UT  VA  VT  WA  \
```

```
0   0   0   0   0   0   0   0   0   0   0  …   0   0   0   0   0   0   0
1   0   0   0   0   0   0   0   0   0   0  …   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0  …   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0  …   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0  …   0   0   0   0   0   0   0

   WI  WV  WY
0   0   0   0
1   0   0   0
2   0   0   0
3   0   0   0
4   0   0   0

[5 rows x 51 columns]
```

[21]:
```
X = churn_data[['Account_Length', 'Vmail_Message', 'Day_Mins',␣
 ↪'Eve_Mins','Night_Mins', 'Intl_Mins', 'CustServ_Calls', 'Intl_Plan',␣
 ↪'Vmail_Plan', 'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge',␣
 ↪'Night_Calls', 'Night_Charge', 'Intl_Calls', 'Intl_Charge']]
y = churn_data['Churn']
```

[31]:
```
# Create arrays for the features and the target: X, y
# X, y = churn_data.iloc[:,:-1], churn_data.iloc[:,-1]

# Create the training and test sets
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123, stratify = y)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Instantiate the XGBClassifier: xg_cl
xg_cl = xgb.XGBClassifier(objective='binary:logistic',
n_estimators=10, seed=123)

# Fit the classifier to the training set
xg_cl.fit(X_train, y_train)

# Predict the labels of the test set: preds
preds = xg_cl.predict(X_test)

# Compute the accuracy: accuracy
accuracy = float(np.sum(preds==y_test))/y_test.shape[0]
print("accuracy: %f" % (accuracy))
```

```
accuracy: 0.922039
```

With XGBoost we get a reasonably high accuracy.

Applying the same model without XGBoost on the same data.

[32]:
```
# Create the training and test sets
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123, stratify = y)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Import LogisticRegression
from sklearn.linear_model import LogisticRegression
# Instantiate the classifier
clf = LogisticRegression(random_state = 0)
# Fitting the classifier
clf.fit(X_train, y_train)

# Predicting the Test set results
y_pred = clf.predict(X_test)

# Compute accuracy
print(clf.score(X_test, y_test))
```

```
0.8620689655172413

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

The model without XGBoost gives a lower accuracy.

What is a decision tree?

XGBoost is usually used with trees as base learners. In Decision Trees a single question is asked at each decision node and only 2 possible choices. At the very bottom of each decision tree, there is a single possible decision.

For example, a decision tree for whether to purchase a vehicle, the first question one may ask is whether it has been road-tested. If it hasn't, one can immediately decide not to buy, otherwise, one continues asking such as what the vehicles mileage is and if its age is old or recent. At bottom, every possible decision will eventually lead to a choice, some taking many fewer questions to get to those choices than others.

For now we can think of any individual learning algorithm in an ensemble algorithm as a base learner. This is important as XGBoost itself is an ensemble learning method and it uses the

outputs of many models for a final prediction. A decision tree is a learning method that involves a tree-like graph to model either a continuous or categorical choice given some data. it is composed of a series of binary decision (yes/no or true/false questions) that when answered in succession ultimately yield a prediction about the data at hand (these predictions happen at the leaves of the tree). Decision trees are constructed iteratively (that is, one binary decision at a time) until some stopping criteria is met. The depth of the tree reaches some pre-defined maximum value. During construction the tree is built one split at a time and the way that a split is selected (that is what features to split on and where in the feature's range of values to split) can vary but involves choosing a split point that segregates the target values better, that is, puts each target category into buckets that are increasingly dominated by just one category until all or nearly all values within a given split are exclusively of one category or another. Using this process each leaf of the decision tree will have a single category in the majority or should be exclusively of one category.

Individual decision trees in general are low-bias, high-variance learning models. That is, they are very good at learning relationships within any data you train them on but they tend to overfit the data you use to train them on and usually generalize to new data poorly.

XGBoost uses a slightly different kind of a decision tree called a classification and regression tree or CART. Whereas for the decision trees, the leaf nodes always contain decision values, CART trees contain a real-valued score in each leaf regardless of whether they are used for classification or regression. The real-valued scores can then be thresholded to convert into categories for classification problems if necessary.

[ ]:
```
###### Decision trees

Here I will make a simple decision tree using scikit-learn's␣
↪DecisionTreeClassifier on the breast cancer dataset that comes pre-loaded␣
↪with scikit-learn.

This dataset contains numeric measurements of various dimensions of individual␣
↪tumors (such as perimeter and texture) from breast biopsies and a single␣
↪outcome value (the tumor is either malignant, or benign).

I will upload dataset of samples (measurements) into X and the target values␣
↪per tumor into y. I will split the complete dataset into training and␣
↪testing sets, and then train a DecisionTreeClassifier. I'll specify a␣
↪parameter called max_depth
```

[36]:
```
# https://necromuralist.github.io/data_science/posts/predicting-cancer/ (source)
from sklearn.datasets import load_breast_cancer
```

[37]:
```
cancer = load_breast_cancer()
```

[39]:
```
cancer
```

[39]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
```

```
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
         8.758e-02],
        …,
        [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 569\n\n    :Number of
Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute
Information:\n        - radius (mean of distances from center to points on the
perimeter)\n        - texture (standard deviation of gray-scale values)\n
- perimeter\n        - area\n        - smoothness (local variation in radius
lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity
(severity of concave portions of the contour)\n        - concave points (number
```

of concave portions of the contour)\n - symmetry \n - fractal dimension ("coastline approximation" - 1)\n\n The mean, standard error, and "worst" or largest (mean of the three\n largest values) of these features were computed for each image,\n resulting in 30 features. For instance, field 3 is Mean Radius, field\n 13 is Radius SE, field 23 is Worst Radius.\n\n - class:\n - WDBC-Malignant\n - WDBC-Benign\n\n :Summary Statistics:\n\n
=================================== ====== ======\n
Min Max\n
=================================== ====== ======\n
radius (mean): 6.981 28.11\n
texture (mean): 9.71 39.28\n
perimeter (mean): 43.79 188.5\n
area (mean): 143.5 2501.0\n
smoothness (mean): 0.053 0.163\n
compactness (mean): 0.019 0.345\n
concavity (mean): 0.0 0.427\n
concave points (mean): 0.0 0.201\n
symmetry (mean): 0.106 0.304\n
fractal dimension (mean): 0.05 0.097\n
radius (standard error): 0.112 2.873\n
texture (standard error): 0.36 4.885\n
perimeter (standard error): 0.757 21.98\n
area (standard error): 6.802 542.2\n
smoothness (standard error): 0.002 0.031\n
compactness (standard error): 0.002 0.135\n
concavity (standard error): 0.0 0.396\n
concave points (standard error): 0.0 0.053\n
symmetry (standard error): 0.008 0.079\n
fractal dimension (standard error): 0.001 0.03\n
radius (worst): 7.93 36.04\n
texture (worst): 12.02 49.54\n
perimeter (worst): 50.41 251.2\n
area (worst): 185.2 4254.0\n
smoothness (worst): 0.071 0.223\n
compactness (worst): 0.027 1.058\n
concavity (worst): 0.0 1.252\n
concave points (worst): 0.0 0.291\n
symmetry (worst): 0.156 0.664\n
fractal dimension (worst): 0.055 0.208\n
=================================== ====== ======\n\n :Missing Attribute Values: None\n\n :Class Distribution: 212 - Malignant, 357 - Benign\n\n :Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n :Donor: Nick Street\n\n :Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a breast mass. They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to construct a decision tree. Relevant features\nwere selected using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is

```
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N.
Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction \n     for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n
San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast
cancer diagnosis and \n     prognosis via linear programming. Operations
Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg,
W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose
breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n
163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean
area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': '/opt/anaconda3/lib/python3.7/site-
packages/sklearn/datasets/data/breast_cancer.csv'}
```

[40]:
```
print(cancer.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names',
'filename'])
```

[48]:
```
def answer_one():
    """converts the sklearn 'cancer' bunch

    Returns:
     pandas.DataFrame: cancer data
    """
    data = numpy.c_[cancer.data, cancer.target]
    columns = numpy.append(cancer.feature_names, ["target"])
    return pandas.DataFrame(data, columns=columns)
```

[51]:
```
import pandas
frame = answer_one()
assert frame.shape == (len(cancer.target), 31)
```

[52]:
```
def answer_three():
    """splits the data into data and labels
```

```
    Returns:
     (pandas.DataFrame, pandas.Series): data, labels
    """
    cancerdf = answer_one()
    X = cancerdf[cancerdf.columns[:-1]]
    y = cancerdf.target
    return X, y
```

[53]:
```
# Import the necessary modules
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Create the training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123)

# Instantiate the classifier: dt_clf_4
dt_clf_4 = DecisionTreeClassifier(max_depth=4)

# Fit the classifier to the training set
dt_clf_4.fit(X_train, y_train)

# Predict the labels of the test set: y_pred_4
y_pred_4 = dt_clf_4.predict(X_test)

# Compute the accuracy of the predictions: accuracy
accuracy = float(np.sum(y_pred_4==y_test))/y_test.shape[0]
print("accuracy:", accuracy)
```

```
accuracy: 0.9070464767616192
```

It's now time to learn about what gives XGBoost its state-of-the-art performance: Boosting.

What is Boosting

The core concept that gives XGBoost its state of the art performance is boosting. Boosting isn't really a specific machine learning algorithm but a concept that can be applied to a set of machine learning models. It's a meta algorithm. Specifically it is an ensemble meta-algorithm primarily used to reduce any given single learner's variance and to convert many weak learners into an arbitrarily strong learner. A weak learner is any machine learning algorithm that is just slightly better than chance. So a decision tree that can predict some outcome slightly more frequently than pure randomness would be considered a weak learner. The principal insight that allows XGBoost to work is the fact that we can use boosting to convert a collection of weak learners into a strong learner. Where a strong learner is any algorithm that can be tuned to achieve arbitrarily good performance for some supervised learning problem. It is done by iteratively learning a set of weak models on subsets of the data we have at hand, and weighting each of their predictions according to each weak learner's performance. We then combine all of the weak learners' predictions multiplied by their weights to obtain a single final weighted predictions that is much better than any of the individual predictions themselves.

In case of two decision trees, each tree gives a different prediction score depending on the data it sees. The prediction scores for each possibility are summed across trees and the prediction is simply the sum of the scores across both trees.

I will be working with XGBoost's learning API for model evaluation and it will be a good idea to briefly describe an example that shows how model evaluation using cross-validation works with XGBoost's learing API (which is different from the scikit-learn compatible API) as it has cross-validation capabilities baked in.

Cross validation is a robust method for estimating the expected performance of a machine learning model on unseen data by generating many non-overlapping train/test splits on the training data and reporting the average test set performance across all data splits.

Measuring accuracy

I'll now practice using XGBoost's learning API through its baked in cross-validation capabilities. XGBoost gets its lauded performance and efficiency gains by utilizing its own optimized data structure for datasets called a DMatrix.

When we use the xgboost cv object, we have to first explicitly convert the data into a DMatrix. So, that's what I will do here before running cross-validation on churn_data.

[56]:
```
# Replacing 'no' with 0 and 'yes' with 1 in 'Vmail_Plan'
churn_data['Vmail_Plan'] = churn_data['Vmail_Plan'].replace({'no':0, 'yes':1})

# Replacing 'no' with 0 and 'yes' with 1 in 'Churn'
churn_data['Churn'] = churn_data['Churn'].replace({'no':0, 'yes':1})

# Replacing 'no' with 0 and 'yes' with 1 in 'Intl_Plan'
churn_data['Intl_Plan'] = churn_data['Intl_Plan'].replace({'no':0, 'yes':1})
```

[57]:
```
# Creating arrays for the features and the target: X, y
# X, y = churn_data.iloc[:,:-1], churn_data.iloc[:,-1]
X = churn_data[['Account_Length', 'Vmail_Message', 'Day_Mins',␣
 ↪'Eve_Mins','Night_Mins', 'Intl_Mins', 'CustServ_Calls', 'Intl_Plan',␣
 ↪'Vmail_Plan', 'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge',␣
 ↪'Night_Calls', 'Night_Charge', 'Intl_Calls', 'Intl_Charge']]
y = churn_data['Churn']

# Creating the DMatrix from X and y: churn_dmatrix
churn_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary: params
params = {"objective":"reg:logistic", "max_depth":3}

# Performing cross-validation: cv_results
cv_results = xgb.cv(dtrain=churn_dmatrix, params=params,
                    nfold=3, num_boost_round=5,
                    metrics="error", as_pandas=True, seed=123)
```

```
# Printing cv_results
print(cv_results)

# Printing the accuracy
print(((1-cv_results["test-error-mean"]).iloc[-1]))
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

	train-error-mean	train-error-std	test-error-mean	test-error-std
0	0.091809	0.003729	0.100810	0.009031
1	0.091209	0.003414	0.102310	0.010505
2	0.085059	0.000735	0.096310	0.013471
3	0.079358	0.004682	0.093609	0.012143
4	0.076208	0.004893	0.092109	0.016006

```
0.9078906666666666
```

Measuring AUC

Now that I've used cross-validation to compute average out-of-sample accuracy (after converting from an error), it's very easy to compute any other metric we might be interested in. All we have to do is pass it (or a list of metrics) in as an argument to the metrics parameter of xgb.cv().

Here I will compute another common metric used in binary classification - the area under the curve ("auc").

[58]:
```
# Performing cross_validation: cv_results
cv_results = xgb.cv(dtrain=churn_dmatrix, params=params,
                    nfold=3, num_boost_round=5,
                    metrics="auc", as_pandas=True, seed=123)

# Printing cv_results
print(cv_results)

# Printing the AUC
print((cv_results["test-auc-mean"]).iloc[-1])
```

	train-auc-mean	train-auc-std	test-auc-mean	test-auc-std
0	0.836970	0.004261	0.823101	0.022818
1	0.852943	0.015413	0.842077	0.016330
2	0.884122	0.009758	0.872564	0.028379
3	0.898340	0.001999	0.886186	0.017050
4	0.903081	0.001136	0.891238	0.017271

```
0.8912383333333334
```

Fantastic! An AUC of 0.89 is quite strong. As we have seen, XGBoost's learning API makes it very

easy to compute any metric we may be interested in. Below I will work on techniques to fine-tune the XGBoost models to improve their performance even further. For now, it's time to learn a little about exactly when to use XGBoost.

When should and shouldn't we use XGBoost

XGBoost can be used for any supervised machine learning task that fits the following criteria.

A dataset that has few features and at least 1000 examples. However, in general as long as the number of features in the training set is smaller than the number of examples there are, it should be fine. XGBoost also tends to do well when there is a mixture of categorical and numeric features, or when there are only numeric features.

When should we not use XGBoost

The most important kinds of problems where XGBoost is a suboptimal choice involve either those that have found success using other state-of-the-art algorithms or those that suffer from dataset size issues. Specifically, XGBoost is not ideally suited for image recognition, computer vision, or natural language processing and understanding problems as those kinds of problems can be much better tackled using deep learning approaches. In terms of dataset size problems, XGBoost is not suitable when you have very small training sets (less than 100 training examples) or when the number of training examples is significantly smaller than the number of features being used for training.

Using XGBoost

XGBoost is a powerful library that scales very well to many samples and works for a variety of supervised learning problems. That said, we shouldn't always pick it as a default machine learning library when starting a new project, since there are some situations in which it is not the best option. Below is an example where XGBoost can work well.

Predicting the likelihood that a given user will click an ad from a very large clickstream log with millions of users and their web interactions.

Regression Review

Regression problems involve predicting continuous or real values. For example, if we are attempting to predict the height in centimeters a given person will be at 30 given some of their physical attributes at birth, we are solving a regression problem.

Evaluating the quality of a regression model involves using a different set of metrics than those I used in classification problems above. To evaluate a regression problem we use RMSE or the Mean Absolute Error (MAE) to evaluate the quality of a regression model. RMSE is computed by taking the difference between the actual and the predicted values for what we are trying to predict, squaring those differences, computing their mean and taking that value's square root. This allows us to treat negative and positive differences equally but tends to punish larger differences between predicted and actual values much more than smaller ones. MAE on the other hand simply sums the absolute differences between predicted and actual values across all of the samples we build our model on. Although MAE isn't affected by large differences as much as RMSE, it lacks some nice mathematical properties that make it much less frequently used as an evaluation metric.

Some common algorithms that are used for regression problems include linear regression and decision trees. Decision trees can be used for regression as well as classification tasks which is one

of their prime properties that makes them prime candidates to be building blocks for XGBoost models.

Which of these is a regression problem?

Here are 4 potential machine learning problems you might encounter in the wild. Pick the one that is a clear example of a regression problem.

1 - Recommending a restaurant to a user given their past history of restaurant visits and reviews for a dining aggregator app.

This is a recommendation problem.

2 - Predicting which of several thousand diseases a given person is most likely to have given their symptoms.

This is a multi-class classification problem.

3 - Tagging an email as spam/not spam based on its content and metadata (sender, time sent, etc.).

This is a binary classification problem.

4 - Predicting the expected payout of an auto insurance claim given claim properties (car, accident type, driver prior history, etc.).

This is indeed an example of a regression problem.

Objective (loss) functions and base learners

Objective functions and base learners are critical to understand in order for us to be able to grasp why XGBoost is such a powerful approach to building supervised regression models. An objective or loss function quantifies how far off our prediction is from the actual result for a given data point. It maps the difference between the prediction and the target to a real number. When we construct any machine learning model we hope that it minimizes the loss function across all of the data points we pass in. The ultimate goal is smallest possible loss. Loss functions have specific naming conventions in XGBoost. For regression models, the most common loss function used is called reg linear. For binary classification models, the most common loss functions used are reg logistic, when we simply want the category of the target, and binary logistic when we want the actual predicted probability of the positive class. Above I implicitly used the reg logistic loss function when building the classification models in XGBoost.

XGBoost is an ensemble learning method is composed of many individual models that are added together to generate a single prediction. Each of the individual models that are trained and combined are called base learners. The goal of XGBoost is to have base learners that is slightly better than random guessing on certain subsets of training examples, and uniformly bad at the remainder, so that when all of the predictions combined, uniformly bad predictions cancel out and those slightly better than chance combine into a single very good prediction.

Below are examples using trees and linear base learners in XGBoost. Below is an example of how to train an XGBoost regression model with trees as base learners using XGBoost's scikit-learn compatible API.

The libraries that we need have already been imported and data is loaded. Then we convert the data in X matrix and y vector and split the data in training and test set. Next we create XGBoost regressor object, making sure we use the reg linear objective function, fit it to the training data

and generate our prediction on the test set. Finally we compute the RMSE and print the result to screen.

[77]:
```
boston_data = pd.read_csv('boston.csv')

X, y = boston_data.iloc[:,:-1], boston_data.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123)

xg_reg = xgb.XGBRegressor(objective = 'reg:linear', n_estimators = 10,
                                                    seed = 123)

xg_reg.fit(X_train, y_train)

preds = xg_reg.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, preds))

print('RMSE: %f' % (rmse))
```

```
[07:59:49] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
RMSE: 9.749041
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

To use linear base learners we have to use the learning API in XGBoost. In the below example we do the same in first 3 lines that we did above. In line 4 and 5 we convert training and testing sets into DMatrix objects as is required by the learning API. Next we create a parameter dictionary explicitly specifying the base learner we want as a gblinear and the gb objective function we want to use. Next we train our model on the training set and generate our predictions on the test set. Next we compute our RMSE and print to screen as before.

[71]:
```
import xgboost as xgb
from sklearn import metrics

boston_data = pd.read_csv('boston.csv')

X, y = boston_data.iloc[:,:-1], boston_data.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123)

DM_train = xgb.DMatrix(data = X_train, label =y_train)
DM_test = xgb.DMatrix(data = X_test, label =y_test)
```

```
params = {'booster':'gblinear', 'objective':'reg:linear'}
xg_reg = xgb.train(params = params, dtrain = DM_train, num_boost_round=10)

preds = xg_reg.predict(DM_test)
```

```
[01:26:45] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

[74]:
```
from sklearn.metrics import mean_squared_error

rmse = np.sqrt(mean_squared_error(y_test, preds))

print('RMSE: %f' % (rmse))
```

```
RMSE: 9.749041
```

[ ]:
```
###### Decision trees as base learners

It's now time to build an XGBoost model to predict house prices - not in␣
↪Boston, Massachusetts, but in Ames, Iowa! This dataset of housing prices has␣
↪been pre-loaded into a DataFrame called df. If we explore it , we'll see␣
↪that there are a variety of features about the house and its location in the␣
↪city.

Here my goal is to use trees as base learners. By default, XGBoost uses trees␣
↪as base learners, so we don't have to specify that we want to use trees here␣
↪with booster="gbtree".

xgboost has been imported as xgb and the arrays for the features and the target␣
↪are available in X and y, respectively.
```

[78]:
```
df = pd.read_csv('ames_preprocessed.csv')
```

[79]:
```
df.head()
```

[79]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
0	60	65	8450	7	5	2003	
1	20	80	9600	6	8	1976	
2	60	68	11250	7	5	2001	
3	70	60	9550	7	5	1915	
4	60	84	14260	8	5	2000	

	Remodeled	GrLivArea	BsmtFullBath	BsmtHalfBath	…	HouseStyle_1.5Unf	\
0	0	1710	1	0	…	0	
1	0	1262	0	1	…	0	
2	1	1786	1	0	…	0	
3	1	1717	1	0	…	0	

```
4          0       2198             1             0  …                   0

   HouseStyle_1Story  HouseStyle_2.5Fin  HouseStyle_2.5Unf  HouseStyle_2Story  \
0                  0                  0                  0                  1
1                  1                  0                  0                  0
2                  0                  0                  0                  1
3                  0                  0                  0                  1
4                  0                  0                  0                  1

   HouseStyle_SFoyer  HouseStyle_SLvl  PavedDrive_P  PavedDrive_Y  SalePrice
0                  0                0             0             1     208500
1                  0                0             0             1     181500
2                  0                0             0             1     223500
3                  0                0             0             1     140000
4                  0                0             0             1     250000

[5 rows x 57 columns]
```

[80]:
```
X, y = df.iloc[:,:-1], df.iloc[:,-1]
```

[81]:
```
# Creating the training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123)

# Instantiating the XGBRegressor: xg_reg
xg_reg = xgb.XGBRegressor(objective='reg:linear', n_estimators=10, seed=123)

# Fitting the regressor to the training set
xg_reg.fit(X_train, y_train)

# Predicting the labels of the test set: preds
preds = xg_reg.predict(X_test)

# Computing the rmse: rmse
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
[08:26:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
RMSE: 78847.401758

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

Next, I'll train an XGBoost model using linear base learners and XGBoost's learning API. Will it perform better or worse?

Linear base learners

Now that I've used trees as base models in XGBoost, let's use the other kind of base model that can be used with XGBoost - a linear learner. This model, although not as commonly used in XGBoost, allows us to create a regularized linear regression using XGBoost's powerful learning API. However, because it's uncommon, I will have to use XGBoost's own non-scikit-learn compatible functions to build the model, such as xgb.train().

In order to do this we must create the parameter dictionary that describes the kind of booster we want to use (similarly to how I created the dictionary above when I used xgb.cv()). The key-value pair that defines the booster type (base model) we need is "booster":"gblinear".

Once I've created the model, I can use the .train() and .predict() methods of the model just like I've done before.

Here, I will split the data into training and testing sets, next I will dive right into creating the DMatrix objects required by the XGBoost learning API.

[82]:
```
# Converting the training and testing sets into DMatrixes: DM_train, DM_test
DM_train = xgb.DMatrix(data=X_train, label=y_train)
DM_test =  xgb.DMatrix(data=X_test, label=y_test)

# Creating the parameter dictionary: params
params = {"booster":"gblinear", "objective":"reg:linear"}

# Training the model: xg_reg
xg_reg = xgb.train(params = params, dtrain=DM_train, num_boost_round=5)

# Predicting the labels of the test set: preds
preds = xg_reg.predict(DM_test)

# Computing and print the RMSE
rmse = np.sqrt(mean_squared_error(y_test,preds))
print("RMSE: %f" % (rmse))
```

```
[08:33:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
RMSE: 44331.645061
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

Interesting - it looks like linear base learners performed better!

Evaluating model quality

It's now time to begin evaluating model quality.

Here, I will compare the RMSE and MAE of a cross-validated XGBoost model on the Ames housing data.

[84]:
```
# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)
```

```
# Creating the parameter dictionary: params
params = {"objective":"reg:linear", "max_depth":4}

# Performing cross-validation: cv_results
cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=4,␣
 ↪num_boost_round=5, metrics='rmse', as_pandas=True, seed=123)

# Printing cv_results
print(cv_results)

# Extracting and print final boosting round metric
print((cv_results["test-rmse-mean"]).tail(1))
```

```
[08:40:21] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:40:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:40:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:40:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
   train-rmse-mean  train-rmse-std  test-rmse-mean  test-rmse-std
0    141767.488281      429.449371   142980.464844    1193.806011
1    102832.562500      322.503447   104891.398438    1223.161012
2     75872.621094      266.493573    79478.947265    1601.341377
3     57245.657226      273.633063    62411.919922    2220.151162
4     44401.291992      316.426590    51348.276367    2963.378029
4    51348.276367
Name: test-rmse-mean, dtype: float64
```

[83]:
```
# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary: params
params = {"objective":"reg:linear", "max_depth":4}

# Performing cross-validation: cv_results
cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=4,␣
 ↪num_boost_round=5, metrics='mae', as_pandas=True, seed=123)

# Printing cv_results
print(cv_results)

# Extracting and print final boosting round metric
print((cv_results["test-mae-mean"]).tail(1))
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \

[08:39:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:39:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:39:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[08:39:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
   train-mae-mean  train-mae-std  test-mae-mean  test-mae-std
0   127343.595703     668.167771  127634.185547   2404.009753
1    89770.031250     456.980559   90122.505860   2107.916842
2    63580.782226     263.442189   64278.558594   1887.552548
3    45633.181640     151.849960   46819.175781   1459.821980
4    33587.097656      87.003217   35670.655274   1140.613227
4    35670.655274
Name: test-mae-mean, dtype: float64
```

Regularization and base learners in XGBoost

Loss functions in XGBoost don't just take into account how close a model's predictions are to the actual values but also takes into account how complex the model is. This idea of penalizing models as they become more complex is called regularization. So, loss functions in XGBoost are used to find models that are both accurate and as simple as they can possibly be. There are several parameters that can be tweaked in XGBoost to limit model complexity by altering the loss function. Gamma is a parameter for tree base learners that controls whether a given node on a base learner will split based on the expected reduction in the loss that would occur after performing the split, so that higher values lead to fewer splits. Alpha is another name for L1 regularization. However, this regularization term is a penalty on leaf weights rather than on feature weights, as is the case in linear or logistic regression. Higher alpha values lead to stronger L1 regularization, which causes many leaf weights in the base learners to go to 0. Lambda is another name for L2 regularization. L2 is a much smoother penalty than L1 and causes leaf weights to smoothly decrease, instead of enforcing strong sparsity constraints on the leaf weights as in L1.

Let's see how we can tune of these regularization parameters using XGBoost.

[90]:
```
X,y = boston_data.iloc[:,:-1],boston_data.iloc[:,-1]
housing_dmatrix = xgb.DMatrix(data=X, label=y)
params = {"objective":"reg:linear","max_depth":4}
l1_params = [1, 10, 100]
rmses_l1 = []
for reg in l1_params:
```

```
        params["alpha"] = reg

        cv_results_rmse = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=4,
                                 num_boost_round=5, metrics="rmse",␣
 ↪as_pandas=True, seed=123)

        rmses_l1.append(cv_results_rmse["test-rmse-mean"].tail(1).values[0])

print("Best rmse as a function of l1:")
print(pd.DataFrame(list(zip(l1_params, rmses_l1)), columns=["l1","rmse"]))
```

```
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:22:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best rmse as a function of l1:
    l1      rmse
0    1  5.601136
1   10  6.003651
2  100  6.950627
```

After creating X and y vectors we convert our X matrix and y vector into a single optimized DMatrix object. Next we create our parameter dictionary that defines some required parameters for our learner. Specifically we provide the loss function necessary for regression and the maximum depth each tree base learner can have. Next we create a list of 3 different l1 or alpha values that we will try, and then we initialize an empty list that will store our final root mean square error for each of these l1 or alpha values. Next is actually a multi-line for loop where we iterate over each entry in our l1 params list and do the following:

First we create a new key-value pair in our parameter dictionary that holds our current alpha value. We then run our XGBoost cross validation by passing in our DMatrix object, updated parameter dictionary, number of folds we want to cross-validate, number of trees we want as num_boost_round. The metric we want to compute, which is rmse, and that we want to output the results as a pandas DataFrame.

Next we simply look at the final RMSE as a function of l1 regularization strength.

I will now compare the two kind of base learners that exist in XGBoost.

The linear base learner is simply a sum of linear terms exactly as you would find in a linear or logistic regression model. When you combine many of these base models into an ensemble you get a weighted sum of linear models which is itself linear. Since you don't get any nonlinear combination of features in the final mode, this approach is rarely used, as you can get identical performance from a regularized linear model. The tree base learners uses decision trees as base models. When the decision trees are all combined into an ensemble, their combination becomes a nonlinear function of each individual tree, which itself is nonlinear.

After the results are computed, datafrmaes are created. Zip and the list functions are used, one inside of the other, to convert multiple equal length lists into a single object that we can convert into a pandas dataframe. Zip is a function that allows you to take multiple equal length lists and iterate over them in parallel, side by side. In python 3, zip creates a generator, or an object that doesn't have to be completely instantiated at runtime. In order for the entire zipped pair of lists to be instantiated we have to cast the zip generator object into a list directly. After casting, we can convert this object directly into a dataframe.

Using regularization in XGBoost

I'll now vary the l2 regularization penalty - also known as "lambda" - and see its effect on overall model performance on the Ames housing dataset.

[94]:
```
# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

reg_params = [1, 10, 100]

# Creating the initial parameter dictionary for varying l2 strength: params
params = {"objective":"reg:linear","max_depth":3}

# Create an empty list for storing rmses as a function of l2 complexity
rmses_l2 = []

# Iterating over reg_params
for reg in reg_params:

    # Updating l2 strength
    params["lambda"] = reg

    # Passing this updated param dictionary into cv
```

```
    cv_results_rmse = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=2,␣
↪num_boost_round=5, metrics="rmse", as_pandas=True, seed=123)

    # Appending best rmse (final round) to rmses_l2
    rmses_l2.append(cv_results_rmse["test-rmse-mean"].tail(1).values[0])

# Looking at best rmse per l2 param
print("Best rmse as a function of l2:")
print(pd.DataFrame(list(zip(reg_params, rmses_l2)), columns=["l2","rmse"]))
```

```
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[12:44:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best rmse as a function of l2:
    l2       rmse
0    1   6.022222
1   10   7.201520
2  100  10.692149
```

It looks like as the value of 'lambda' increases, so does the RMSE.

Visualizing individual XGBoost trees

Now that I've used XGBoost to both build and evaluate regression as well as classification models, I'll try to get a handle on how to visually explore the models. Here, I will visualize individual trees from the fully boosted model that XGBoost creates using the entire housing dataset.

XGBoost has a plot_tree() function that makes this type of visualization easy. Once we train a model using the XGBoost learning API, we can pass it to the plot_tree() function along with the number of trees we want to plot using the num_trees argument.

[ ]:
```
# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary: params
params = {"objective":"reg:linear", "max_depth":2}

# Training the model: xg_reg
xg_reg = xgb.train(params=params, dtrain=housing_dmatrix, num_boost_round=10)
```

```
# Plotting the first tree
xgb.plot_tree(xg_reg, num_trees = 0)
plt.show()

# Plotting the fifth tree
xgb.plot_tree(xg_reg, num_trees = 4)
plt.show()

# Plotting the last tree sideways
xgb.plot_tree(xg_reg, num_trees = 9, rankdir='LR')
plt.show()

# The above code only works after installing graphviz
```

Have a look at each of the plots. They provide insight into how the model arrived at its final decisions and what splits it made to arrive at those decisions. This allows us to identify which features are the most important in determining house price. Next I'll work on another way of visualizing feature importances.

Visualizing feature importances: What features are most important in my dataset

Another way to visualize the XGBoost models is to examine the importance of each feature column in the original dataset within the model.

One simple way of doing this involves counting the number of times each feature is split on across all boosting rounds (trees) in the model, and then visualizing the result as a bar graph, with the features ordered according to how many times they appear. XGBoost has a plot_importance() function that allows you to do exactly this, and we'll get a chance to use it.

[ ]:
```
import matplotlib.pyplot as plt

# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary: params
params = {"objective":"reg:linear", "max_depth":4}

# Training the model: xg_reg
xg_reg = xgb.train(params=params, dtrain=housing_dmatrix, num_boost_round=10)

# Plotting the feature importances
xgb.plot_importance(xg_reg)
plt.show()
```

It looks like RM is the most important feature.

Why tune your model?

To motivate myself on tuning the model, I will work on two models, one tuned and other untuned. Here I will try to make XGBoost models as performant as possible. Here I will work on the variety of

parameters that can be adjusted to alter the behavior of XGBoost and how to tune them efficiently so that we can supercharge the performance of the models.

[126]:
```
X,y = df[df.columns.tolist()[:-1]], df[df.columns.tolist()[-1]]

housing_dmatrix = xgb.DMatrix(data=X, label=y)
untuned_params = {'objective':'reg:linear'}
untuned_cv_results_rmse = xgb.cv(dtrain=housing_dmatrix,
                  params = untuned_params, nfold=4,
                  metrics = 'rmse', as_pandas=True, seed=123)
print('Untuned rmse: %f' %((untuned_cv_results_rmse['test-rmse-mean']).tail(1)))
```

```
[14:23:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[14:23:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[14:23:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[14:23:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Untuned rmse: 34624.230957
```

Now I will compare the above model with a tuned model.

[127]:
```
X,y = df[df.columns.tolist()[:-1]], df[df.columns.tolist()[-1]]

housing_dmatrix = xgb.DMatrix(data=X, label=y)

tuned_params = {'objective':'reg:linear', 'colsample_bytree':0.3,
    'learning_rate':0.1, 'max_depth':5}

tuned_cv_results_rmse = xgb.cv(dtrain=housing_dmatrix,
                  params = tuned_params, nfold=4, num_boost_round=200,
                  metrics = 'rmse', as_pandas=True, seed=123)
print('Tuned rmse: %f' %((tuned_cv_results_rmse['test-rmse-mean']).tail(1)))
```

```
[16:40:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:40:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:40:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:40:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Tuned rmse: 30370.555176
```

When is tuning your model a bad idea?

Now that you've seen the effect that tuning has on the overall performance of your XGBoost model, let's turn the question on its head and see if you can figure out when tuning your model might not

be the best idea. Given that model tuning can be time-intensive and complicated, which of the following scenarios would NOT call for careful tuning of your model?

i - You have lots of examples from some dataset and very many features at your disposal.

If you have lots of examples and features, tuning your model is a great idea.

ii - You are very short on time before you must push an initial model to production and have little data to train your model on. You cannot tune if you do not have time!

iii - You have access to a multi-core (64 cores) server with lots of memory (200GB RAM) and no time constraints. This is a perfect example of when you should definitely tune your model.

iv - You must squeeze out every last bit of performance out of your xgboost model. If you are trying to squeeze out performance, you are definitely going to tune your model. ‘

Tuning the number of boosting rounds

Let’s start with parameter tuning by seeing how the number of boosting rounds (number of trees we build) impacts the out-of-sample performance of the XGBoost model. I’ll use xgb.cv() inside a for loop and build one model per num_boost_round parameter.

Here, I’ll continue working with the Ames housing dataset.

[128]:
```
# Creating the DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary for each tree: params
params = {"objective":"reg:linear", "max_depth":3}

# Creating list of number of boosting rounds
num_rounds = [5, 10, 15]

# Empty list to store final round rmse per XGBoost model
final_rmse_per_round = []

# Iterating over num_rounds and build one model per num_boost_round parameter
for curr_num_rounds in num_rounds:

    # Performing cross-validation: cv_results
    cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=3,␣
 ↪num_boost_round=curr_num_rounds, metrics="rmse", as_pandas=True, seed=123)

    # Appending final round RMSE
    final_rmse_per_round.append(cv_results["test-rmse-mean"].tail().values[-1])

# Printing the resultant DataFrame
num_rounds_rmses = list(zip(num_rounds, final_rmse_per_round))
print(pd.DataFrame(num_rounds_rmses,columns=["num_boosting_rounds","rmse"]))
```

```
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:47:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
   num_boosting_rounds          rmse
0                    5  50903.299479
1                   10  34774.191406
2                   15  32895.098307
```

As you can see, increasing the number of boosting rounds decreases the RMSE.

Automated boosting round selection using early_stopping

Now, instead of attempting to cherry pick the best possible number of boosting rounds, we can very easily have XGBoost automatically select the number of boosting rounds within xgb.cv(). This is done using a technique called early stopping.

Early stopping works by testing the XGBoost model after every boosting round against a hold-out dataset and stopping the creation of additional boosting rounds (thereby finishing training of the model early) if the hold-out metric ("rmse" in our case) does not improve for a given number of rounds. Here I will use the early_stopping_rounds parameter in xgb.cv() with a large possible number of boosting rounds (50). Bear in mind that if the holdout metric continuously improves up through when num_boost_rounds is reached, then early stopping does not occur.

Here, the DMatrix and parameter dictionary have been created you. The task is to use cross-validation with early stopping.

[129]:
```python
# Creating your housing DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary for each tree: params
params = {"objective":"reg:linear", "max_depth":4}

# Performing cross-validation with early stopping: cv_results
cv_results = xgb.cv(dtrain=housing_dmatrix, params=params,␣
 ↪nfold=3,early_stopping_rounds=10, num_boost_round=50, metrics='rmse',␣
 ↪as_pandas=True, seed=123)
```

```
# Printing cv_results
print(cv_results)
```

```
[16:48:53] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:48:53] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:48:53] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
    train-rmse-mean  train-rmse-std  test-rmse-mean  test-rmse-std
0     141871.630208      403.632409   142640.630208     705.552907
1     103057.033854       73.787612   104907.677083     111.124997
2      75975.958333      253.705643    79262.057292     563.761707
3      57420.515625      521.666323    61620.138021    1087.681933
4      44552.960938      544.168971    50437.558594    1846.450522
5      35763.942708      681.796885    43035.660156    2034.476339
6      29861.469401      769.567549    38600.881511    2169.803563
7      25994.679036      756.524834    36071.816407    2109.801581
8      23306.832031      759.237670    34383.183594    1934.542189
9      21459.772786      745.623841    33509.141927    1887.374589
10     20148.728516      749.612756    32916.806641    1850.890045
11     19215.382162      641.387202    32197.834635    1734.459068
12     18627.391276      716.256399    31770.848958    1802.156167
13     17960.697265      557.046469    31482.781901    1779.126300
14     17559.733724      631.413289    31389.990234    1892.321401
15     17205.712891      590.168517    31302.885417    1955.164927
16     16876.571615      703.636538    31234.060547    1880.707358
17     16597.666992      703.677646    31318.347656    1828.860164
18     16330.460612      607.275030    31323.636719    1775.911103
19     16005.972331      520.472435    31204.138021    1739.073743
20     15814.299479      518.603218    31089.865885    1756.024090
21     15493.405924      505.617405    31047.996094    1624.672630
22     15270.733724      502.021346    31056.920573    1668.036788
23     15086.381836      503.910642    31024.981120    1548.988924
24     14917.606445      486.208398    30983.680990    1663.131129
25     14709.591797      449.666844    30989.479818    1686.664414
26     14457.285156      376.785590    30952.116536    1613.170520
27     14185.567708      383.100492    31066.899088    1648.531897
28     13934.065104      473.464919    31095.643880    1709.226491
29     13749.646485      473.671156    31103.885417    1778.882817
30     13549.837891      454.900755    30976.083984    1744.514903
31     13413.480469      399.601066    30938.469401    1746.051298
32     13275.916341      415.404898    30931.000651    1772.471473
33     13085.878906      493.793750    30929.056640    1765.541487
34     12947.182292      517.789542    30890.625651    1786.510889
35     12846.026367      547.731831    30884.489583    1769.731829
36     12702.380534      505.522036    30833.541667    1690.999881
```

37	12532.243815	508.298122	30856.692709	1771.447014
38	12384.056641	536.224879	30818.013672	1782.783623
39	12198.445312	545.165866	30839.394531	1847.325690
40	12054.582682	508.840691	30776.964844	1912.779519
41	11897.033528	477.177882	30794.703776	1919.677255
42	11756.221354	502.993261	30780.961589	1906.820582
43	11618.846029	519.835813	30783.754557	1951.258396
44	11484.081380	578.429092	30776.734375	1953.449992
45	11356.550781	565.367451	30758.544271	1947.456794
46	11193.557292	552.298192	30729.973307	1985.701585
47	11071.317383	604.088404	30732.662760	1966.997355
48	10950.777018	574.864279	30712.243490	1957.751584
49	10824.865885	576.664748	30720.852214	1950.513825

Overview of XGBoost's hyperparameters

Overview of XGBoost's hyperparameters

I will now go over the differences in what parameters can be tuned for each kind of base model in XGBoost. The parameters that can be tuned are significantly different for each base learner.

For the tree base learner which is the one we should use in almost every single case, the most frequently tuned parameters are outlined below.

- The learning rate affects how quickly the model fits the residual error using additional base learners. A low learning rate will require more boosting rounds to achieve the same reduction in residual error as an XGBoost model with a high learning rate.
- Gamma, alpha and lambda all have effect on how strongly regularized the trained model will be.
- Max_depth must be a positive integer value and affects how deeply each tree is allowed to grow during any given boosting round.
- Subsample must be a value between 0 and 1 and is the fraction of the total training set that can be used for any given boosting round. If the value is low, then the fraction of the training data used per boosting round would be low and we may run into under fitting problems. A value that is very high can lead to overfitting as well.
- Colsample_bytree is the fraction of features we can select from during any given boosting round and must also be a value between 0 and 1. A large value means that almost all features can be used to build a tree during a given boosting round, whereas a small value means that the fraction of features that can be selected from is very small. In general, smaller colsample_bytree values can be thought of as providing additional regularization to the model. Whereas using all columns may in certain cases overfit a trained model.

For the linear base learner the number of tunable parameters is significantly smaller.

- We only have access to l1 and l2 regularization on the weights associated with any given feature, and then another regularization term that can be applied to the model's bias.

Finally it is important to mention that the number of boosting rounds (that is, either the number of trees we build or the number of linear base learners we construct) is itself a tunable parameter.

Tuning eta

It's time to practice tuning other XGBoost hyperparameters in earnest and observing their effect on model performance! I'll begin by tuning the "eta", also known as the learning rate.

The learning rate in XGBoost is a parameter that can range between 0 and 1, with higher values of "eta" penalizing feature weights more strongly, causing much stronger regularization.

[130]:
```
# Creating your housing DMatrix: housing_dmatrix
housing_dmatrix = xgb.DMatrix(data=X, label=y)

# Creating the parameter dictionary for each tree (boosting round)
params = {"objective":"reg:linear", "max_depth":3}

# Creating list of eta values and empty list to store final round rmse per␣
↪xgboost model
eta_vals = [0.001, 0.01, 0.1]
best_rmse = []

# Systematically vary the eta
for curr_val in eta_vals:

    params["eta"] = curr_val

    # Performing cross-validation: cv_results
    cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=3,
                        num_boost_round=10, early_stopping_rounds=5,
                        metrics="rmse", as_pandas=True, seed=123)

    # Appending the final round rmse to best_rmse
    best_rmse.append(cv_results["test-rmse-mean"].tail().values[-1])

# Printing the resultant DataFrame
print(pd.DataFrame(list(zip(eta_vals, best_rmse)), columns=["eta","best_rmse"]))
```

```
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[16:51:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
```

```
deprecated in favor of reg:squarederror.
[16:51:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
     eta      best_rmse
0  0.001  195736.406250
1  0.010  179932.161458
2  0.100   79759.401041
```

Tuning max_depth

Here I will tune max_depth, which is the parameter that dictates the maximum depth that each tree in a boosting round can grow to. Smaller values will lead to shallower trees, and larger values to deeper trees.

[132]:
```
# Creating your housing DMatrix
housing_dmatrix = xgb.DMatrix(data=X,label=y)

# Creating the parameter dictionary
params = {"objective":"reg:linear", "max_depth":20}

# Creating list of max_depth values
max_depths = [2, 5, 10, 20]
best_rmse = []

# Systematically vary the max_depth
for curr_val in max_depths:

    params["max_depth"] = curr_val

    # Performing cross-validation
    cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=2,
                        num_boost_round=10, early_stopping_rounds=5,
                        metrics="rmse", as_pandas=True, seed=123)

    # Appending the final round rmse to best_rmse
    best_rmse.append(cv_results["test-rmse-mean"].tail().values[-1])

# Printing the resultant DataFrame
print(pd.DataFrame(list(zip(max_depths,␣
 ↪best_rmse)),columns=["max_depth","best_rmse"]))
```

```
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:30:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
   max_depth     best_rmse
0          2  37957.476562
1          5  35596.599610
2         10  36065.537110
3         20  36739.574219
```

[ ]:
```
###### Tuning colsample_bytree

Now, it's time to tune "colsample_bytree". I've already seen this if you've␣
↪ever worked with scikit-learn's RandomForestClassifier or␣
↪RandomForestRegressor, where it just was called max_features. In both␣
↪xgboost and sklearn, this parameter (although named differently) simply␣
↪specifies the fraction of features to choose from at every split in a given␣
↪tree. In xgboost, colsample_bytree must be specified as a float between 0␣
↪and 1.
```

[133]:
```
# Create your housing DMatrix
housing_dmatrix = xgb.DMatrix(data=X,label=y)

# Create the parameter dictionary
params={"objective":"reg:linear","max_depth":3}

# Create list of hyperparameter values
colsample_bytree_vals = [0.1, 0.5, 0.8, 1]
best_rmse = []

# Systematically vary the hyperparameter value
for curr_val in colsample_bytree_vals:

    params["colsample_bytree"] = curr_val

    # Perform cross-validation
    cv_results = xgb.cv(dtrain=housing_dmatrix, params=params, nfold=2,
                        num_boost_round=10, early_stopping_rounds=5,
                        metrics="rmse", as_pandas=True, seed=123)

    # Append the final round rmse to best_rmse
    best_rmse.append(cv_results["test-rmse-mean"].tail().values[-1])
```

```python
# Print the resultant DataFrame
print(pd.DataFrame(list(zip(colsample_bytree_vals, best_rmse)),␣
 ↪columns=["colsample_bytree","best_rmse"]))
```

```
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[23:33:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
   colsample_bytree     best_rmse
0               0.1  51386.587890
1               0.5  36585.345703
2               0.8  36093.660157
3               1.0  35836.042968
```

There are several other individual parameters that we can tune, such as “subsample”, which dictates the fraction of the training data that is used during any given boosting round. Next up: Grid Search and Random Search to tune XGBoost hyperparameters more efficiently!

Review of Grid Search and Random Search

How do we find the optimal values for several hyperparameters simultaneously leading to the lowest loss possible when their values interact in non-obvious, non-linear ways?

Two common strategies for choosing several hyperparameters values simultaneously are Grid Search and Random Search.

Grid Search is a method of exhaustively searching through a collection of possible parameters values. For example, if we have 2 hyperparameters we would like to use and 4 possible values for each hyperparameter, then a grid search over that parameter space would try all 16 possible parameter configurations. In a grid search, we try every parameter configuration, evaluate some metric for that configuration and pick the parameter configuration that gave the best value for the metric we were using which in our case will be root mean squared error.

Let’s go over an example of how to grid search over several hyperparameters using XGBoost and scikit learn. After importing the libraries and loading the data and converting it into a DMatrix. In next line we create our grid of hyperparameters we want to search over. If we select 4 different learning rates (or eta values), 3 different subsample values, and a single number of trees. The total number of distinct hyperparameter configurations is 12, so 12 different models will be built. Next

we create our regressor and then we pass the xgbregressor object, parameter grid, evaluation metric, and number of cross validation folds to GridSearchCV and then immediately fit that gridsearch object just like every other scikit learn estimator object. Next, having fit the gridsearch object, we can extract the best parameters the grid search found and print them to the screen. In next line we get the RMSE that corresponds to the best parameters found and see its value.

[138]:
```
# df is the ames housing dataset

from sklearn.model_selection import GridSearchCV

X,y = df[df.columns.tolist()[:-1]], df[df.columns.tolist()[-1]]

housing_dmatrix = xgb.DMatrix(data=X, label=y)

gbm_param_grid = {'learning_rate': [0.01, 0.1, 0.5, 0.9],
                  'n_estimators':[200],
                  'subsample':[0.3,0.5,0.9]}
gbm = xgb.XGBRegressor()
grid_mse = GridSearchCV(estimator=gbm, param_grid = gbm_param_grid,
           scoring = 'neg_mean_squared_error', cv=4, verbose=1)
grid_mse.fit(X,y)
print('Best parameters found:', grid_mse.best_params_)
print('Lowest RMSE found:',np.sqrt(np.abs(grid_mse.best_score_)))
```

```
Fitting 4 folds for each of 12 candidates, totalling 48 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[01:31:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:07] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:07] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:14] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:15] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:16] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:16] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:18] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:21] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:30] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:33] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:35] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:36] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:37] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[01:31:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:   40.1s finished
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \

[01:31:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best parameters found: {'learning_rate': 0.1, 'n_estimators': 200, 'subsample':
0.5}
Lowest RMSE found: 28410.039476552454
```

[ ]:
```
Lowest RMSE found is 28410.039476552454.
```

Random search is significantly different from Grid Search in that the number of models that we are required to iterate over doesn't grow as expand the overall hyperparameter space. In random search we get to decide how many models or iterations we want to try out before stopping. Random search simply involves drawing a random combination of possible hyperparameter values from the range of allowable hyperparameters a set number of times. Each time we train a model with the selected hyperparameters, evaluate the performance of the model and then rinse and repeat. When we have created the number of models that we had specified initially, we simply pick the best one.

Let's work on a full random search example. After data preprocessing we create our parameter grid, this time generating a large number of learning_rate values and subsample values using np.dot.arange and subsample values using np.dot.arange. There are 20 values for learning_rate (or eta) and 20 values for subsample, which would be 400 models to try if we were to tun a grid search (which we aren't doing here). Next we create our xgbregressor object and next we crate our RandomizedSearchCV object passing in the xgbregressor and parameter grid we had just created. We also set the number of iterations we want the random search to proceed to 25, so we know it will not be able to try all 400 possible parameter configurations. We also specify the evaluation metric we want to use, and that we want to run 4-fold cross-validation on each iteration. Next we fit our randomizedsearch object which can take a bit of time. Finally we print the best model parameters found and the corresponding best RMSE.

[142]:
```
# Importing the library
from sklearn.model_selection import RandomizedSearchCV
X,y = df[df.columns.tolist()[:-1]], df[df.columns.tolist()[-1]]

housing_dmatrix = xgb.DMatrix(data=X, label=y)
gbm_param_grid = {'learning_rate': np.arange(0.05, 1.05, .05) ,
                                    'n_estimators':[200],
                                     'subsample': np.arange(0.05, 1.05, .05)}
gbm = xgb.XGBRegressor()
randomized_mse = RandomizedSearchCV(estimator=gbm,␣
 ↪param_distributions=gbm_param_grid,
                         n_iter =25, scoring ='neg_mean_squared_error', cv=4,␣
 ↪verbose=1)
randomized_mse.fit(X,y)
print('Best parameters found:', randomized_mse.best_params_)
print('Lowest RMSE found:',np.sqrt(np.abs(randomized_mse.best_score_)))
```

```
Fitting 4 folds for each of 25 candidates, totalling 100 fits
[02:00:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
```

```
  if getattr(data, 'base', None) is not None and \

[02:00:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:14] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:16] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:18] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:21] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:30] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:32] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:33] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:35] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:35] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:36] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:37] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:44] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:45] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:45] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:46] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:47] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:48] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:48] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:49] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:50] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:50] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:51] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:53] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:54] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:55] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:55] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:56] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:57] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:00:59] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:00] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:06] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:07] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:08] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:08] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:14] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:15] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:15] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:16] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:18] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:21] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:21] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:01:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  1.3min finished
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \

[02:01:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best parameters found: {'subsample': 0.35000000000000003, 'n_estimators': 200,
'learning_rate': 0.15000000000000002}
Lowest RMSE found: 27959.816747233395
```

Grid search with XGBoost

Now that I've tried how to tune parameters individually with XGBoost, I will take the parameter tuning to the next level by using scikit-learn's GridSearch and RandomizedSearch capabilities with internal cross-validation using the GridSearchCV and RandomizedSearchCV functions. I will use these to find the best model exhaustively from a collection of possible parameter values across multiple parameters simultaneously. Let's get to work, starting with GridSearchCV!

[143]:
```
# Creating the parameter grid: gbm_param_grid
gbm_param_grid = {
    'colsample_bytree': [0.3, 0.7],
    'n_estimators': [(50)],
    'max_depth': [2, 5]
}

# Instantiating the regressor: gbm
gbm = xgb.XGBRegressor()

# Performing grid search: grid_mse
grid_mse = GridSearchCV(param_grid = gbm_param_grid, estimator = gbm, scoring =␣
 ↪'neg_mean_squared_error', cv=4, verbose=1)


# Fitting grid_mse to the data
grid_mse.fit(X,y)

# Printing the best parameters and lowest RMSE
print("Best parameters found: ", grid_mse.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(grid_mse.best_score_)))
```

```
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[02:10:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:10:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:10:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    3.5s finished
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[02:10:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best parameters found:  {'colsample_bytree': 0.7, 'max_depth': 5,
'n_estimators': 50}
Lowest RMSE found:  30540.19922467927
```

Random search with XGBoost

Often, GridSearchCV can be really time consuming, so in practice, we may want to use RandomizedSearchCV instead, as I will do now. The good news is we only have to make a few modifications to the GridSearchCV code to do RandomizedSearchCV. The key difference is we have to specify a param_distributions parameter instead of a param_grid parameter.

[145]:
```
# Creating the parameter grid: gbm_param_grid
gbm_param_grid = {
    'n_estimators': [25],
    'max_depth': range(2, 12)
}

# Instantiating the regressor: gbm
gbm = xgb.XGBRegressor(n_estimators=10)

# Performing random search: grid_mse
randomized_mse = RandomizedSearchCV(estimator=gbm,␣
 ↪param_distributions=gbm_param_grid,
                                    n_iter=5, scoring='neg_mean_squared_error',␣
 ↪cv=4, verbose=1)
randomized_mse.fit(X, y)

# Printing the best parameters and lowest RMSE
print("Best parameters found: ",randomized_mse.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(randomized_mse.best_score_)))
```

```
Fitting 4 folds for each of 5 candidates, totalling 20 fits
[02:13:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:13:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:13:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[02:13:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[02:13:06] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[02:13:06] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
Best parameters found:  {'n_estimators': 25, 'max_depth': 4}
Lowest RMSE found:  37447.36831128452

[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    5.2s finished
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

Limits of grid search and random search

Grid search and random search each suffer from distinct limitations. As long as the number of hyperparameters and distinct values per hyperparameter we search over is kept small, grid search will give an answer in a reasonable amount of time. However, as the number of hyperparameter grows the time it takes to complete a full grid search increases exponentially.

For random search the problem is a bit different. Since we can specify how many iterations a random search should be run the time it takes to finish the random search wont explode as we add more and more hyperparameters to search through. The problem really is that as we add new hyperparameters to search over the size of the hyperparameters space explodes as it did in the grid search case and so we are left with hoping that one of the random parameter configurations that the search chooses is a good one. We can always increase the number of iterations we want the random search to run but then finding an optimal configuration becomes a combination of waiting randomly finding a good set of hyperparameters.

Both approaches have significant limitations.

When should we use grid search and random search?

Now that you've seen some of the drawbacks of grid search and random search, which of the following most accurately describes why both random search and grid search are non-ideal search hyperparameter tuning strategies in all scenarios?

a - Grid Search and Random Search both take a very long time to perform, regardless of the number of parameters you want to tune.

This is not always the case.

b - Grid Search and Random Search both scale exponentially in the number of hyperparameters you want to tune.

Only grid-search search time grows exponentially as you add parameters, random-search does not have to grow in this way.

c- The search space size can be massive for Grid Search in certain cases, whereas for Random Search the number of hyperparameters has a significant effect on how long it takes to run.

This is why random search and grid search should not always be used.

d - Grid Search and Random Search require that you have some idea of where the ideal values for hyperparameters reside.

Although this is true, it is true of all hyperparameter search strategies, not just grid and random search.

Using XGBoost in pipelines

Taking my XGBoost skills to the next level by incorporating your models into two end-to-end machine learning pipelines. I will tune the most important XGBoost hyperparameters efficiently within a pipeline, and will also use some more advanced preprocessing techniques.

Review of pipelines using sklearn

Pipelines in sklearn are objects that take a list of named tuples as input. The named tuples must always contain a string name as the first element in each tuple and any scikit-learn compatible transformer or estimator object as the second element. Each named tuple in the pipeline is called a step and the list of transformations that are contained in the list are executed in order once some data is passed through the pipeline. This is done using the standard fit/predict paradigm that is standard in scikit-learn. Finally, where pipelines are really useful is that they can be used as input estimator objects into other scikit-learn objects themselves, the most useful of which are the cross_val_score method, which allows for efficient cross-validation and out of sample metric calculation, and the grid search and random search approaches for tuning hyperparameters.

I will work on the Ames unprocessed housing data which is much more complex and bigger than Boston housing data. Secondly, I will work on kidney data.

[172]:
```
df = pd.read_csv('ames_oroginal.csv')
```

Exploratory data analysis

Before diving into the nitty gritty of pipelines and preprocessing, let's do some exploratory analysis of the original, unprocessed Ames housing dataset. When I worked with this data above, it was already processed so that I could focus on the core XGBoost concepts. In this chapter, I'll do the preprocessing as well!

The DataFrame has 21 columns and 1460 rows.

The mean of the LotArea column is 10516.828082.

The DataFrame has missing values.

The LotFrontage column has no missing values and its entries are of type float64.

The standard deviation of SalePrice is 79442.502883.

[173]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 21 columns):
MSSubClass      1460 non-null int64
MSZoning        1460 non-null object
LotFrontage     1201 non-null float64
```

```
LotArea         1460 non-null int64
Neighborhood    1460 non-null object
BldgType        1460 non-null object
HouseStyle      1460 non-null object
OverallQual     1460 non-null int64
OverallCond     1460 non-null int64
YearBuilt       1460 non-null int64
Remodeled       1460 non-null int64
GrLivArea       1460 non-null int64
BsmtFullBath    1460 non-null int64
BsmtHalfBath    1460 non-null int64
FullBath        1460 non-null int64
HalfBath        1460 non-null int64
BedroomAbvGr    1460 non-null int64
Fireplaces      1460 non-null int64
GarageArea      1460 non-null int64
PavedDrive      1460 non-null object
SalePrice       1460 non-null int64
dtypes: float64(1), int64(15), object(5)
memory usage: 239.7+ KB
```

[174]:
```
df['LotArea'].mean()
```

[174]:
```
10516.828082191782
```

[175]:
```
# Counting missing values in data
print(df.isnull().sum())
```

```
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
Neighborhood      0
BldgType          0
HouseStyle        0
OverallQual       0
OverallCond       0
YearBuilt         0
Remodeled         0
GrLivArea         0
BsmtFullBath      0
BsmtHalfBath      0
FullBath          0
HalfBath          0
BedroomAbvGr      0
Fireplaces        0
GarageArea        0
PavedDrive        0
```

```
SalePrice         0
dtype: int64
```

[176]:
```
df.SalePrice.std()
```

[176]:
```
79442.50288288663
```

The LotFrontage column actually does have missing values: 259, to be precise. Additionally, notice how columns such as MSZoning, PavedDrive, and HouseStyle are categorical. These need to be encoded numerically before we can use XGBoost. This is what I'll do in the coming exercises.

Encoding categorical columns I: LabelEncoder

Now that we've seen what will need to be done to get the housing data ready for XGBoost, let's go through the process step-by-step.

First, we will need to fill in missing values - as we saw previously, the column LotFrontage has many missing values. Then, we will need to encode any categorical columns in the dataset using one-hot encoding so that they are encoded numerically.

The data has five categorical columns: MSZoning, PavedDrive, Neighborhood, BldgType, and HouseStyle. Scikit-learn has a LabelEncoder function that converts the values in each categorical column into integers. I'll practice using this here.

[177]:
```
# Importing LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Filling missing values with 0
df.LotFrontage = df.LotFrontage.fillna(0)

# Creating a boolean mask for categorical columns
categorical_mask = (df.dtypes == object)

# Getting list of categorical column names
categorical_columns = df.columns[categorical_mask].tolist()

# Printing the head of the categorical columns
print(df[categorical_columns].head())

# Creating LabelEncoder object: le
le = LabelEncoder()

# Applying LabelEncoder to categorical columns
df[categorical_columns] = df[categorical_columns].apply(lambda x: le.
 ↪fit_transform(x))

# Printing the head of the LabelEncoded categorical columns
print(df[categorical_columns].head())
```

```
  MSZoning Neighborhood BldgType HouseStyle PavedDrive
```

```
0       RL      CollgCr     1Fam     2Story          Y
1       RL      Veenker     1Fam     1Story          Y
2       RL      CollgCr     1Fam     2Story          Y
3       RL      Crawfor     1Fam     2Story          Y
4       RL      NoRidge     1Fam     2Story          Y
   MSZoning  Neighborhood  BldgType  HouseStyle  PavedDrive
0         3             5         0           5           2
1         3            24         0           2           2
2         3             5         0           5           2
3         3             6         0           5           2
4         3            15         0           5           2
```

[179]:
```
df.dtypes
```

[179]:
```
MSSubClass        int64
MSZoning          int64
LotFrontage     float64
LotArea           int64
Neighborhood      int64
BldgType          int64
HouseStyle        int64
OverallQual       int64
OverallCond       int64
YearBuilt         int64
Remodeled         int64
GrLivArea         int64
BsmtFullBath      int64
BsmtHalfBath      int64
FullBath          int64
HalfBath          int64
BedroomAbvGr      int64
Fireplaces        int64
GarageArea        int64
PavedDrive        int64
SalePrice         int64
dtype: object
```

All the categorical featurs are converted now using label encoder.

Notice how the entries in each categorical column are now encoded numerically. A BldgTpe of 1Fam is encoded as 0, while a HouseStyle of 2Story is encoded as 5.

Encoding categorical columns II: OneHotEncoder

Okay - so we have your categorical columns encoded numerically. Can we now move onto using pipelines and XGBoost? Not yet! In the categorical columns of this dataset, there is no natural ordering between the entries. As an example: Using LabelEncoder, the CollgCr Neighborhood was encoded as 5, while the Veenker Neighborhood was encoded as 24, and Crawfor as 6. Is Veenker "greater" than Crawfor and CollgCr? No - and allowing the model to assume this natural ordering

may result in poor performance.

As a result, there is another step needed: we have to apply a one-hot encoding to create binary, or "dummy" variables. We can do this using scikit-learn's OneHotEncoder.

[180]:
```
# Importing OneHotEncoder
from sklearn.preprocessing import OneHotEncoder

# Creating OneHotEncoder: ohe
ohe = OneHotEncoder(categorical_features=categorical_mask, sparse=False)

# Applying OneHotEncoder to categorical columns - output is no longer a␣
 ↪dataframe: df_encoded
df_encoded = ohe.fit_transform(df)

# Printing first 5 rows of the resulting dataset - again, this will no longer␣
 ↪be a pandas dataframe
print(df_encoded[:5, :])

# Printing the shape of the original DataFrame
print(df.shape)

# Printing the shape of the transformed array
print(df_encoded.shape)
```

```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/_encoders.py:415: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/_encoders.py:451: DeprecationWarning: The
'categorical_features' keyword is deprecated in version 0.20 and will be removed
in 0.22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

```
[[0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 6.000e+01 6.500e+01 8.450e+03
  7.000e+00 5.000e+00 2.003e+03 0.000e+00 1.710e+03 1.000e+00 0.000e+00
```

```
  2.000e+00 1.000e+00 3.000e+00 0.000e+00 5.480e+02 2.085e+05]
 [0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 1.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 2.000e+01 8.000e+01 9.600e+03
  6.000e+00 8.000e+00 1.976e+03 0.000e+00 1.262e+03 0.000e+00 1.000e+00
  2.000e+00 0.000e+00 3.000e+00 1.000e+00 4.600e+02 1.815e+05]
 [0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 6.000e+01 6.800e+01 1.125e+04
  7.000e+00 5.000e+00 2.001e+03 1.000e+00 1.786e+03 1.000e+00 0.000e+00
  2.000e+00 1.000e+00 3.000e+00 1.000e+00 6.080e+02 2.235e+05]
 [0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 7.000e+01 6.000e+01 9.550e+03
  7.000e+00 5.000e+00 1.915e+03 1.000e+00 1.717e+03 1.000e+00 0.000e+00
  1.000e+00 0.000e+00 3.000e+00 1.000e+00 6.420e+02 1.400e+05]
 [0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 6.000e+01 8.400e+01 1.426e+04
  8.000e+00 5.000e+00 2.000e+03 0.000e+00 2.198e+03 1.000e+00 0.000e+00
  2.000e+00 1.000e+00 4.000e+00 1.000e+00 8.360e+02 2.500e+05]]
(1460, 21)
(1460, 62)
```

As we can see, after one hot encoding, which creates binary variables out of the categorical variables, there are now 62 columns.

Encoding categorical columns III: DictVectorizer

Alright, one final trick before we dive into pipelines. The two step process we just went through - LabelEncoder followed by OneHotEncoder - can be simplified by using a DictVectorizer.

Using a DictVectorizer on a DataFrame that has been converted to a dictionary allows us to get label encoding as well as one-hot encoding in one go.

Here the task is to work through this strategy!

[181]:
```
# Importing DictVectorizer
from sklearn.feature_extraction import DictVectorizer

# Converting df into a dictionary: df_dict
df_dict = df.to_dict("records")

# Creating the DictVectorizer object: dv
dv = DictVectorizer(sparse=False)

# Applying dv on df: df_encoded
df_encoded = dv.fit_transform(df_dict)

# Printing the resulting first five rows
print(df_encoded[:5,:])

# Printing the vocabulary
print(dv.vocabulary_)
```

```
[[3.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 2.000e+00 5.480e+02
  1.710e+03 1.000e+00 5.000e+00 8.450e+03 6.500e+01 6.000e+01 3.000e+00
  5.000e+00 5.000e+00 7.000e+00 2.000e+00 0.000e+00 2.085e+05 2.003e+03]
 [3.000e+00 0.000e+00 0.000e+00 1.000e+00 1.000e+00 2.000e+00 4.600e+02
  1.262e+03 0.000e+00 2.000e+00 9.600e+03 8.000e+01 2.000e+01 3.000e+00
  2.400e+01 8.000e+00 6.000e+00 2.000e+00 0.000e+00 1.815e+05 1.976e+03]
 [3.000e+00 0.000e+00 1.000e+00 0.000e+00 1.000e+00 2.000e+00 6.080e+02
  1.786e+03 1.000e+00 5.000e+00 1.125e+04 6.800e+01 6.000e+01 3.000e+00
  5.000e+00 5.000e+00 7.000e+00 2.000e+00 1.000e+00 2.235e+05 2.001e+03]
 [3.000e+00 0.000e+00 1.000e+00 0.000e+00 1.000e+00 1.000e+00 6.420e+02
  1.717e+03 0.000e+00 5.000e+00 9.550e+03 6.000e+01 7.000e+01 3.000e+00
  6.000e+00 5.000e+00 7.000e+00 2.000e+00 1.000e+00 1.400e+05 1.915e+03]
 [4.000e+00 0.000e+00 1.000e+00 0.000e+00 1.000e+00 2.000e+00 8.360e+02
  2.198e+03 1.000e+00 5.000e+00 1.426e+04 8.400e+01 6.000e+01 3.000e+00
  1.500e+01 5.000e+00 8.000e+00 2.000e+00 0.000e+00 2.500e+05 2.000e+03]]
{'MSSubClass': 12, 'MSZoning': 13, 'LotFrontage': 11, 'LotArea': 10,
'Neighborhood': 14, 'BldgType': 1, 'HouseStyle': 9, 'OverallQual': 16,
'OverallCond': 15, 'YearBuilt': 20, 'Remodeled': 18, 'GrLivArea': 7,
'BsmtFullBath': 2, 'BsmtHalfBath': 3, 'FullBath': 5, 'HalfBath': 8,
'BedroomAbvGr': 0, 'Fireplaces': 4, 'GarageArea': 6, 'PavedDrive': 17,
'SalePrice': 19}
```

Preprocessing within a pipeline

Now that we've seen what steps need to be taken individually to properly process the Ames housing data, let's use the much cleaner and more succinct DictVectorizer approach and put it alongside an XGBoostRegressor inside of a scikit-learn pipeline.

[186]:
```
from sklearn.model_selection import train_test_split

# Creating arrays for the features and the target: X, y
X, y = df.iloc[:,:-1], df.iloc[:,-1]
```

[187]:
```
# Importing necessary modules
from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import Pipeline

# Filling LotFrontage missing values with 0
X.LotFrontage = X.LotFrontage.fillna(0)

# Seting up the pipeline steps: steps
steps = [("ohe_onestep", DictVectorizer(sparse=False)),
         ("xgb_model", xgb.XGBRegressor())]

# Creating the pipeline: xgb_pipeline
xgb_pipeline = Pipeline(steps)

# Fitting the pipeline
xgb_pipeline.fit(X.to_dict("records"), y)
```

```
[15:40:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

[187]:
```
Pipeline(memory=None,
         steps=[('ohe_onestep',
                 DictVectorizer(dtype=<class 'numpy.float64'>, separator='=',
                                sort=True, sparse=False)),
                ('xgb_model',
                 XGBRegressor(base_score=0.5, booster='gbtree',
                              colsample_bylevel=1, colsample_bynode=1,
                              colsample_bytree=1, gamma=0,
                              importance_type='gain', learning_rate=0.1,
                              max_delta_step=0, max_depth=3, min_child_weight=1,
                              missing=None, n_estimators=100, n_jobs=1,
                              nthread=None, objective='reg:linear',
                              random_state=0, reg_alpha=0, reg_lambda=1,
                              scale_pos_weight=1, seed=None, silent=None,
                              subsample=1, verbosity=1))],
         verbose=False)
```

[188]:
```
X
```

[188]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	BldgType	\
0	60	3	65.0	8450	5	0	
1	20	3	80.0	9600	24	0	

```
2             60         3         68.0    11250             5         0
3             70         3         60.0     9550             6         0
4             60         3         84.0    14260            15         0
…            …        …           …       …              …        …
1455          60         3         62.0     7917             8         0
1456          20         3         85.0    13175            14         0
1457          70         3         66.0     9042             6         0
1458          20         3         68.0     9717            12         0
1459          20         3         75.0     9937             7         0

      HouseStyle  OverallQual  OverallCond  YearBuilt  Remodeled  GrLivArea  \
0              5            7            5       2003          0       1710
1              2            6            8       1976          0       1262
2              5            7            5       2001          1       1786
3              5            7            5       1915          1       1717
4              5            8            5       2000          0       2198
…            …          …          …         …        …        …
1455           5            6            5       1999          1       1647
1456           2            6            6       1978          1       2073
1457           5            7            9       1941          1       2340
1458           2            5            6       1950          1       1078
1459           2            5            6       1965          0       1256

      BsmtFullBath  BsmtHalfBath  FullBath  HalfBath  BedroomAbvGr  \
0                1             0         2         1             3
1                0             1         2         0             3
2                1             0         2         1             3
3                1             0         1         0             3
4                1             0         2         1             4
…              …           …       …       …           …
1455             0             0         2         1             3
1456             1             0         2         0             3
1457             0             0         2         0             4
1458             1             0         1         0             2
1459             1             0         1         1             3

      Fireplaces  GarageArea  PavedDrive
0              0         548           2
1              1         460           2
2              1         608           2
3              1         642           2
4              1         836           2
…            …         …       …
1455           1         460           2
1456           2         500           2
1457           2         252           2
1458           0         240           2
```

```
1459           0         276           2

[1460 rows x 20 columns]
```

[189]:
```
print(X.isnull().sum())
```

```
MSSubClass      0
MSZoning        0
LotFrontage     0
LotArea         0
Neighborhood    0
BldgType        0
HouseStyle      0
OverallQual     0
OverallCond     0
YearBuilt       0
Remodeled       0
GrLivArea       0
BsmtFullBath    0
BsmtHalfBath    0
FullBath        0
HalfBath        0
BedroomAbvGr    0
Fireplaces      0
GarageArea      0
PavedDrive      0
dtype: int64
```

No more missing values.

Cross-validating your XGBoost model

Here I'll go one step further by using the pipeline I've created to preprocess and cross-validate the model.

[193]:
```
# Importing necessary modules
from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

from sklearn.model_selection import train_test_split

# Creating arrays for the features and the target: X, y
X, y = df.iloc[:,:-1], df.iloc[:,-1]

# Filling LotFrontage missing values with 0
X.LotFrontage = X.LotFrontage.fillna(0)

# Seting up the pipeline steps: steps
```

```python
steps = [("ohe_onestep", DictVectorizer(sparse=False)),
         ("xgb_model", xgb.XGBRegressor(max_depth=2, objective="reg:linear"))]

# Creating the pipeline: xgb_pipeline
xgb_pipeline = Pipeline(steps)

# Cross-validating the model
cross_val_scores = cross_val_score(xgb_pipeline, X.to_dict("records"), y,␣
 ↪cv=10, scoring="neg_mean_squared_error")

# Printing the 10-fold RMSE
print("10-fold RMSE: ", np.mean(np.sqrt(np.abs(cross_val_scores))))
```

```
[16:06:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:27] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
```

```
  if getattr(data, 'base', None) is not None and \

[16:06:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.

/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \

[16:06:30] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
10-fold RMSE:  30343.486551766466
```

Kidney disease case study I: Categorical Imputer

I'll now continue the exploration of using pipelines with a dataset that requires significantly more wrangling. The chronic kidney disease dataset contains both categorical and numeric features, but contains lots of missing values. The goal here is to predict who has chronic kidney disease given various blood indicators as features.

Here I will work with a new library, sklearn_pandas, that allows us to chain many more processing steps inside of a pipeline than are currently supported in scikit-learn. Specifically, I'll be able to impute missing categorical values directly using the Categorical_Imputer() class in sklearn_pandas, and the DataFrameMapper() class to apply any arbitrary sklearn-compatible transformer on DataFrame columns, where the resulting output can be either a NumPy array or DataFrame.

I've also created a transformer called a Dictifier that encapsulates converting a DataFrame using .to_dict("records") without us having to do it explicitly (and so that it works in a pipeline). Finally, I've also provided the list of feature names in kidney_feature_names, the target name in kidney_target_name, the features in X, and the target in y.

Here the task is to apply the CategoricalImputer to impute all of the categorical columns in the dataset. Notice the keyword arguments input_df=True and df_out=True? This is so that we can work with DataFrames instead of arrays. By default, the transformers are passed a numpy array of the selected columns as input, and as a result, the output of the DataFrame mapper is also an array. Scikit-learn transformers have historically been designed to work with numpy arrays, not pandas DataFrames, even though their basic indexing interfaces are similar.

[389]:
```
# Import xgboost
import xgboost as xgb

kidney_feature_names = ['age',
                        'bp',
                        'sg',
                        'al',
                        'su',
                        'rbc',
                        'pc',
                        'pcc',
                        'ba',
                        'bgr',
                        'bu',
                        'sc',
                        'sod',
                        'pot',
                        'hemo',
                        'pcv',
                        'wc',
                        'rc',
                        'htn',
                        'dm',
                        'cad',
                        'appet',
                        'pe',
                        'ane',
                        'class']

df = pd.read_csv('kidney.csv', names=kidney_feature_names, na_values=["?"])
df
```

[389]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	\
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	
..	…	…	…	…	…	…	…	…	…	
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	

	bgr	…	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	121.0	…	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd

```
1      NaN  …  38.0  6000.0  NaN   no   no   no   good   no   no     ckd
2    423.0  …  31.0  7500.0  NaN   no  yes   no   poor   no  yes     ckd
3    117.0  …  32.0  6700.0  3.9  yes   no   no   poor  yes  yes     ckd
4    106.0  …  35.0  7300.0  4.6   no   no   no   good   no   no     ckd
..     …  …   …     …  …  …  …  …    …  …  …     …
395  140.0  …  47.0  6700.0  4.9   no   no   no   good   no   no  notckd
396   75.0  …  54.0  7800.0  6.2   no   no   no   good   no   no  notckd
397  100.0  …  49.0  6600.0  5.4   no   no   no   good   no   no  notckd
398  114.0  …  51.0  7200.0  5.9   no   no   no   good   no   no  notckd
399  131.0  …  53.0  6800.0  6.1   no   no   no   good   no   no  notckd

[400 rows x 25 columns]
```

[390]:
```
# Replacing label values with 0 (ckd) and 1
df['class'].replace({'ckd':0, 'notckd':1}, inplace=True)
df.head()
```

[390]:
```
    age    bp     sg   al   su     rbc        pc         pcc          ba  \
0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

     bgr  …   pcv      wc   rc  htn   dm  cad  appet   pe  ane class
0  121.0  …  44.0  7800.0  5.2  yes  yes   no   good   no   no     0
1    NaN  …  38.0  6000.0  NaN   no   no   no   good   no   no     0
2  423.0  …  31.0  7500.0  NaN   no  yes   no   poor   no  yes     0
3  117.0  …  32.0  6700.0  3.9  yes   no   no   poor  yes  yes     0
4  106.0  …  35.0  7300.0  4.6   no   no   no   good   no   no     0

[5 rows x 25 columns]
```

[391]:
```
X = df[['age','bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',␣
↪'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad', 'appet',␣
↪'pe', 'ane']]
y = df[['class']]
```

[392]:
```
# Counting missing values in data
print(X.isnull().sum())
```

```
age      9
bp      12
sg      47
al      46
su      49
```

```
rbc      152
pc        65
pcc        4
ba         4
bgr       44
bu        19
sc        17
sod       87
pot       88
hemo      52
pcv       71
wc       106
rc       131
htn        2
dm         2
cad        2
appet      1
pe         1
ane        1
dtype: int64
```

Kidney disease case study II: Feature Union

Having separately imputed numeric as well as categorical columns, the task is now to use scikit-learn's FeatureUnion to concatenate their results, which are contained in two separate transformer objects.

Here I will use featureunion. Just like with pipelines, we have to pass it a list of (string, transformer) tuples, where the first half of each tuple is the name of the transformer.

Kidney disease case study III: Full pipeline

It's time to piece together all of the transforms along with an XGBClassifier to build the full pipeline!

Besides the numeric_categorical_union that you created in the previous exercise, there are two other transforms needed: the Dictifier() transform which I have created, and the DictVectorizer().

After creating the pipeline, the task is to cross-validate it to see how well it performs.

Create the pipeline using the numeric_categorical_union, Dictifier(), and DictVectorizer(sort=False) transforms, and xgb.XGBClassifier() estimator with max_depth=3. Name the transforms "featureunion", "dictifier" "vectorizer", and the estimator "clf".

Performing 3-fold cross-validation on the pipeline using cross_val_score(). Pass it the pipeline, the features: X, the outcomes, y. Also set scoring to "roc_auc" and cv to 3.

[410]:
```
# Import modules
import pandas as pd
from sklearn_pandas import DataFrameMapper, CategoricalImputer
from sklearn.preprocessing import Imputer, StandardScaler
from sklearn.pipeline import FeatureUnion
```

```
from sklearn.model_selection import cross_val_score, RandomizedSearchCV
from sklearn.base import BaseEstimator, TransformerMixin
import xgboost as xgb

# Create list of column names for kidney data: kidney_cols
kidney_cols = ['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
               'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm',
               'cad', 'appet', 'pe', 'ane', 'label']

# Load dataset: df_kidney
df_kidney = pd.read_csv('kidney.csv', names=kidney_cols,
                        na_values='?')

# Replace label values with 0 (ckd) and 1
df_kidney['label'].replace({'ckd':0, 'notckd':1}, inplace=True)

# Define X and y: X, y
X, y = df_kidney.iloc[:, :-1], df_kidney['label'].values

# Define new column order for X: col_order
col_order = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot',
             'hemo', 'pcv', 'wc', 'rc', 'rbc', 'pc', 'pcc', 'ba', 'htn', 'dm',
             'cad', 'appet', 'pe', 'ane']

# Rearrange columns of X
X = X[col_order]

# Create a boolean mask for categorical columns
categorical_feature_mask = X.dtypes == object

# Get a list of categorical column names
categorical_columns = X.columns[categorical_feature_mask].tolist()

# Get a list of non-categorical column names
non_categorical_columns = X.columns[~categorical_feature_mask].tolist()

# Create empty list to hold column imputers: transformers
transformers = []

# Create numeric imputers and add to list of transformers
transformers.extend([([num_col], [Imputer(strategy='median'),
                                                  StandardScaler()]) for num_col
                     in non_categorical_columns])

# Create categorical imputers and add to list of transformers
transformers.extend([(cat_col, [CategoricalImputer()]) for cat_col in
                     categorical_columns])
```

```python
# Use list of transformers to create a DataFrameMapper object
numeric_categorical_union = DataFrameMapper(transformers, input_df=True,
                                            df_out=True)

# Define Dictifier class to turn df into dictionary as part of pipeline
class Dictifier(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.to_dict('records')

# Create full pipeline
pipeline = Pipeline([('featureunion', numeric_categorical_union),
                    ('dictifier', Dictifier()),
                    ('vectorizer', DictVectorizer(sort=False)),
                    ('clf', xgb.XGBClassifier(max_depth=3))])

# Perform cross-validation
cross_val_scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=3)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
```

```
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
```

```
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
```

```
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
```

```
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

[406]:
```
cross_val_scores
```

[406]:
```
array([0.99928571, 0.99686747, 0.99975904])
```

[407]:
```
print("3-fold AUC: ", np.mean(cross_val_scores))
```

```
3-fold AUC:  0.998637406769937
```

Bringing it all together

Alright, it's time to bring together everything I have done so far! In this final exercise I will combine the work from above into one end-to-end XGBoost pipeline to really cement the understanding of preprocessing and pipelines in XGBoost.

The work from the previous 3 exercises, where I preprocessed the data and set up the pipeline, has been pre-loaded. The task here is to perform a randomized search and identify the best hyperparameters.

[408]:
```
# Creating the parameter grid
gbm_param_grid = {
    'clf__learning_rate': np.arange(.05, 1, .05),
    'clf__max_depth': np.arange(3,10, 1),
    'clf__n_estimators': np.arange(50, 200, 50)
}

# Performing RandomizedSearchCV
randomized_roc_auc = RandomizedSearchCV(estimator=pipeline,
                                        param_distributions=gbm_param_grid,
                                        n_iter=2, scoring='roc_auc', cv=2,␣
 ↪verbose=1)

# Fitting the estimator
randomized_roc_auc.fit(X, y)
```

```python
# Computing metrics
print(randomized_roc_auc.best_score_)
print(randomized_roc_auc.best_estimator_)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
```

```
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
```

```
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
```

```
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
```

```
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
```

```
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
```

```
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
```

```
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.7s finished
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
```

```
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:66:
DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in
version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from
sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

```
0.9980266666666665
```

```
Pipeline(memory=None,
         steps=[('featureunion',
                 DataFrameMapper(default=False, df_out=True,
                                 features=[(['age'],
                                            [Imputer(axis=0, copy=True,
                                                     missing_values='NaN',
                                                     strategy='median',
                                                     verbose=0),
                                             StandardScaler(copy=True,
                                                            with_mean=True,
                                                            with_std=True)]),
                                           (['bp'],
                                            [Imputer(axis=0, copy=True,
                                                     missing_values='NaN',
                                                     strategy='median',
                                                     verbose=0),
                                             StandardScaler(copy=True,
                                                            wit…
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0, learning_rate=0.1,
                               max_delta_step=0, max_depth=9,
                               min_child_weight=1, missing=None,
                               n_estimators=50, n_jobs=1, nthread=None,
                               objective='binary:logistic', random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               seed=None, silent=None, subsample=1,
                               verbosity=1))],
         verbose=False)
```

2 Thanks a lot for your attention.

[ ]: