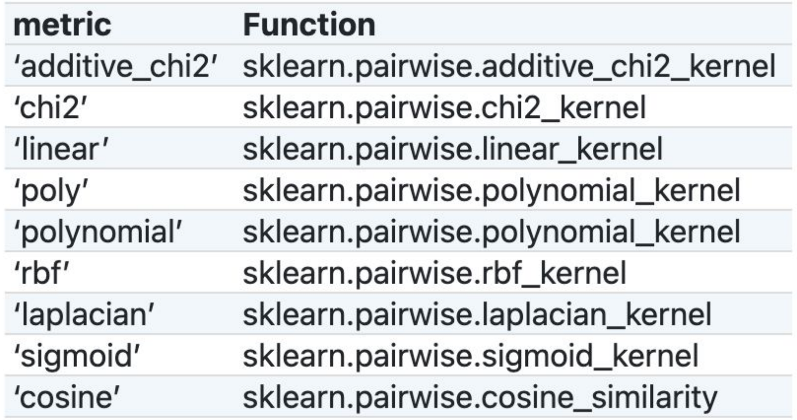
A Brief Intro to Decision Trees, Random Forest, and Ridge Regression, and How to Get Better Scores in General

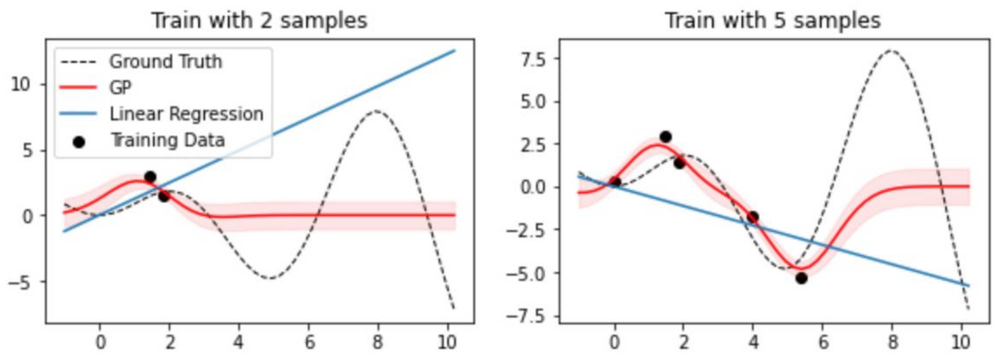
Kernel Ridge Regression

- Ridge regression is just multi-linear regression with some extra regularization terms
    - Basically just some math that helps prevent overfitting during linear regression
- The kernel method is some more complicated math
    - Maps input data to a higher dimensional feature space
    - This just means it can predict non-linearities well
        - Lots of non-linear data in hydrology (think of the complicated non-linear physics equations present in nature)
    - Lots of different kernel methods in machine learning
- Kernel ridge regression is thus ridge regression with the kernel method applied
    - Basically multi-linear regression with extra protection against overfitting and better prediction of non-linearities

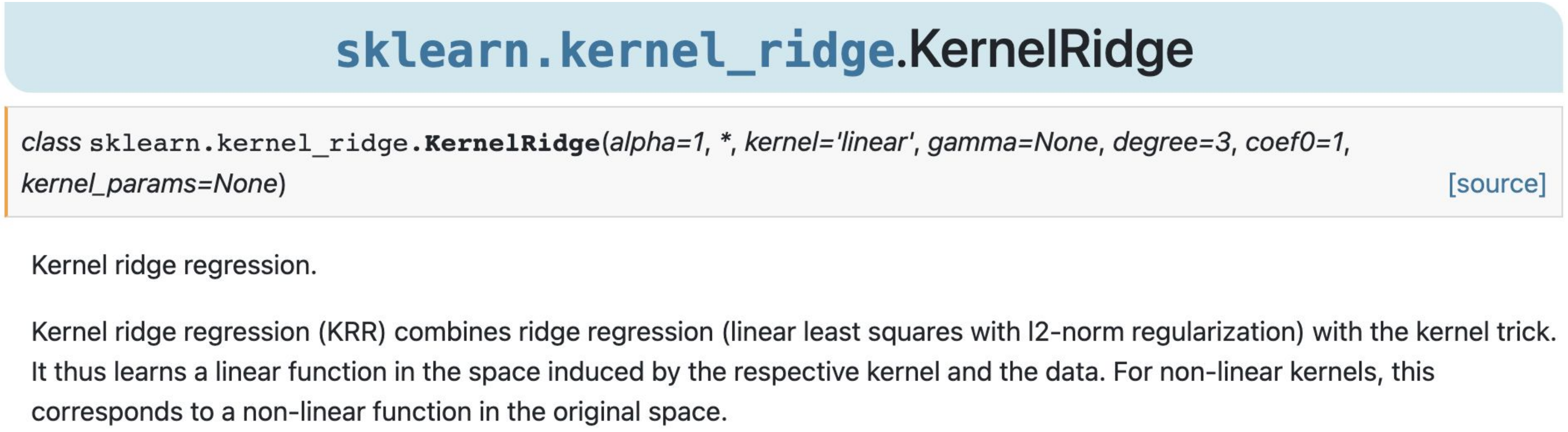
Kernel Ridge Regression



metric	Function
‘additive_chi2’	sklearn.pairwise.additive_chi2_kernel
‘chi2’	sklearn.pairwise.chi2_kernel
‘linear’	sklearn.pairwise.linear_kernel
‘poly’	sklearn.pairwise.polynomial_kernel
‘polynomial’	sklearn.pairwise.polynomial_kernel
‘rbf’	sklearn.pairwise.rbf_kernel
‘laplacian’	sklearn.pairwise.laplacian_kernel
‘sigmoid’	sklearn.pairwise.sigmoid_kernel
‘cosine’	sklearn.pairwise.cosine_similarity



https://pat.chormai.org/blog/2021-krr-gps



sklearn.kernel_ridge.KernelRidge

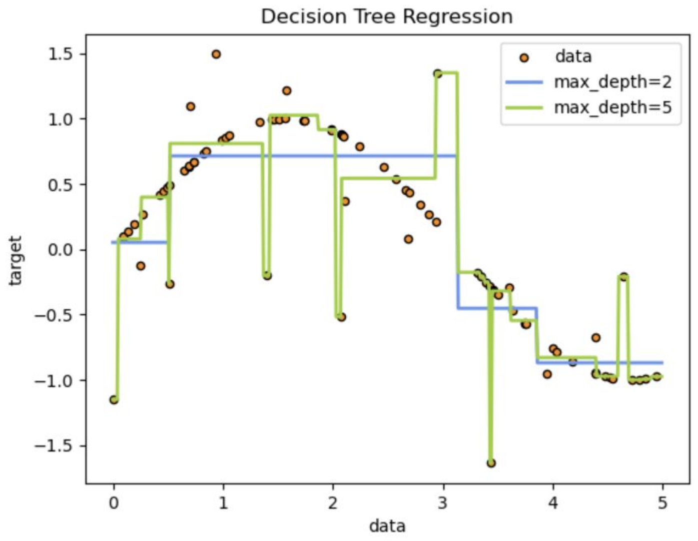
class `sklearn.kernel_ridge.`**`KernelRidge`**(*alpha=1*, ***, *kernel='linear'*, *gamma=None*, *degree=3*, *coef0=1*, *kernel_params=None*) [source]

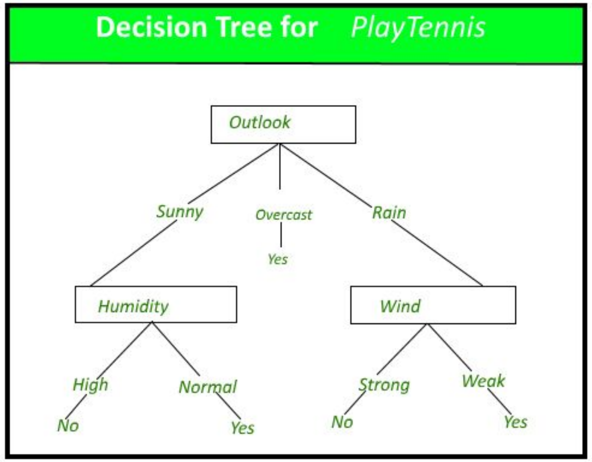
Kernel ridge regression.

Kernel ridge regression (KRR) combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space.

Decision Trees

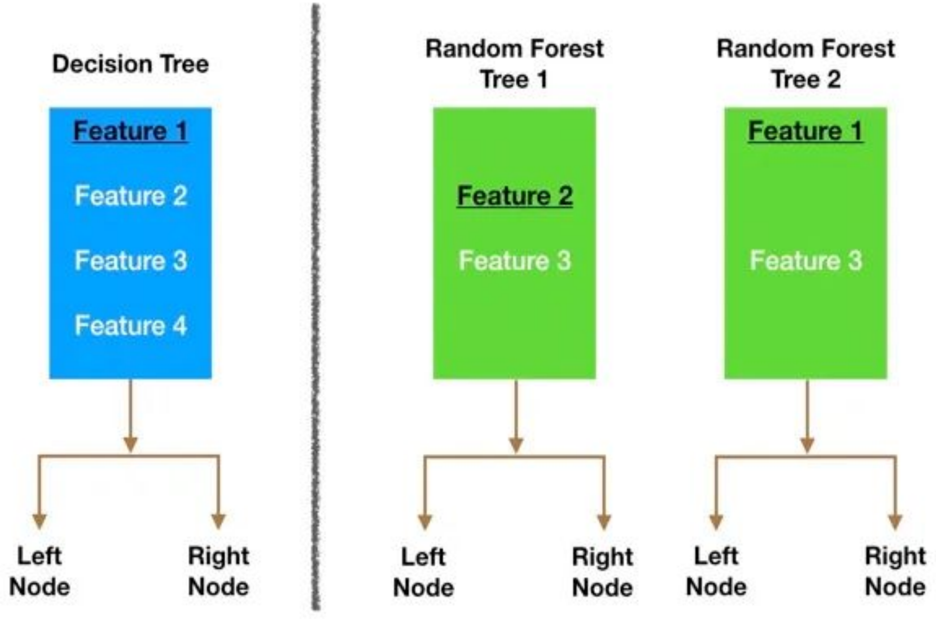
- Decision trees work through recursive partitioning
- Advantages
  - Simple and easy to understand
  - Fast (low computational cost)
  - Can accept numerical and categorical (string) inputs (although not sklearn)
- Disadvantages
  - Prone to overfitting
  - Predict through piecewise functions
    - Unseen and significantly different data is very hard to predict with a non-continuous function
  - Class imbalances lead to biased trees
    - Use feature engineering or generate synthetic data to address class imbalances





Random Forest

- Ensemble of decision trees
- Features are randomly sampled to create a number of decision trees (bagging)
  - If the model also includes boosting, this means the data is weighted based on classification/regression performance
- Each tree is trained and their scores are averaged to produce the final score
- Advantages
  - Reduces overfitting of decision trees
  - Great for datasets with lots of features, still good for small features
  - Can identify important variables
- Disadvantages
  - Loss of interpretability
  - Not great for regression (still basically piecewise)
  - Training with a large number of trees may take long



https://towardsdatascience.com/understanding-random-forest-58381e0602d2#:~:text=The%20random%20forest%20is%20a,that%20of%20any%20individual%20tree.

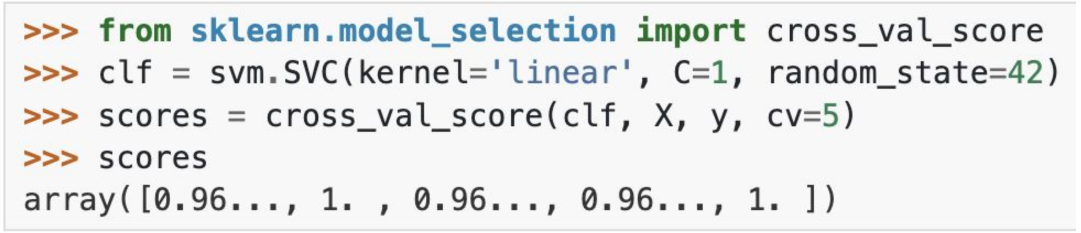
How to Get Better Model Scores

- Add more data
  - If you can't find more, try synthetic generation (repeat data with slight Gaussian noise)
- Clean data
  - Remove outliers (but only if you have a good reason)
  - Impute missing values
- Try different standardization and normalization methods
- **Try a different ML model (everyone loves XGBoost)**
  - **Play around with model hyperparameters**
  - **For lots of parameters, use gridsearchcv**
- Use feature engineering to transform data
  - Create a new feature from one or more old ones
    - Calculate monthly variables, combine weakly correlated variables to capture a different correlation
    - Bin one or more variables
    - Binarize a variable
  - Replace features
    - Combine features with high multicollinearity (or just remove one)
- **Ensemble methods (average predictions from multiple models)**
  - **Train the same data on different models, or train parts of the data on the same model**
- Cross-validation (may also give you lower scores, but more accurate than holdout)
- Project data into a new dimension via PCA, kernel methods, LDA, etc.
  - Using principal components as new features can also reduce scores
  - Complicated application to regression problems

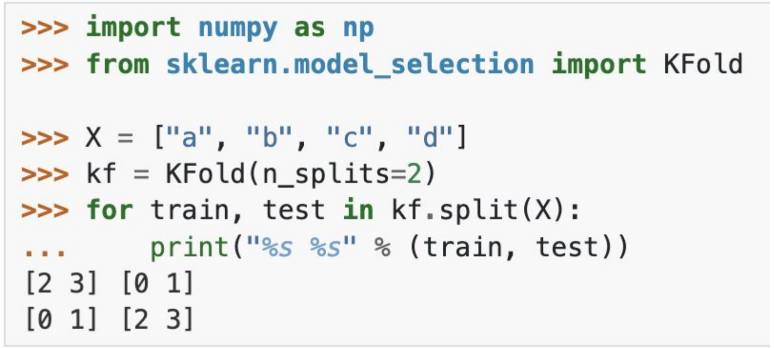
How to Get Better Model Scores

- Reframe the problem
  - Classification vs regression, supervised vs unsupervised, time series, specific site analysis
  - Just use different scores
- Try something outside of sklearn
  - Different machine learning frameworks will provide slightly different results for the same model
  - Keras, Tensorflow and PyTorch (deep learning with neural nets)
- How to test score improvement
  - Use cross-validation to get a baseline score (holdout is an unreliable method for determining actual improvements)
  - Add model improvements
  - Get new cross-validation score

Cross Validation and GridSearchCV in Sklearn

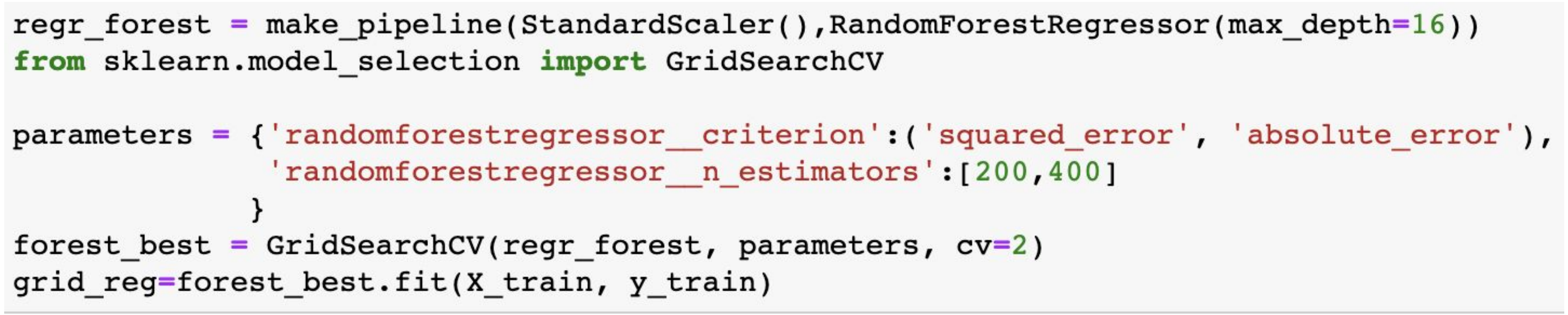


```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1, random_state=42)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores
array([0.96..., 1.  , 0.96..., 0.96..., 1. ])
```

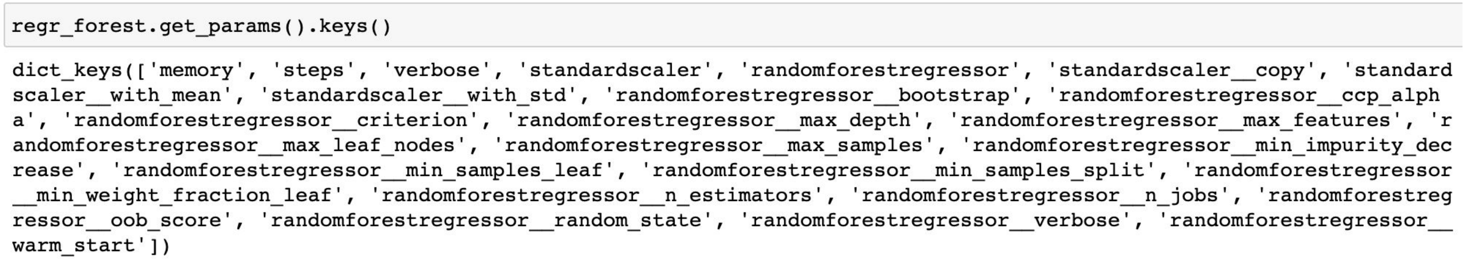


```
>>> import numpy as np
>>> from sklearn.model_selection import KFold

>>> X = ["a", "b", "c", "d"]
>>> kf = KFold(n_splits=2)
>>> for train, test in kf.split(X):
...     print("%s %s" % (train, test))
[2 3] [0 1]
[0 1] [2 3]
```

Grid Search



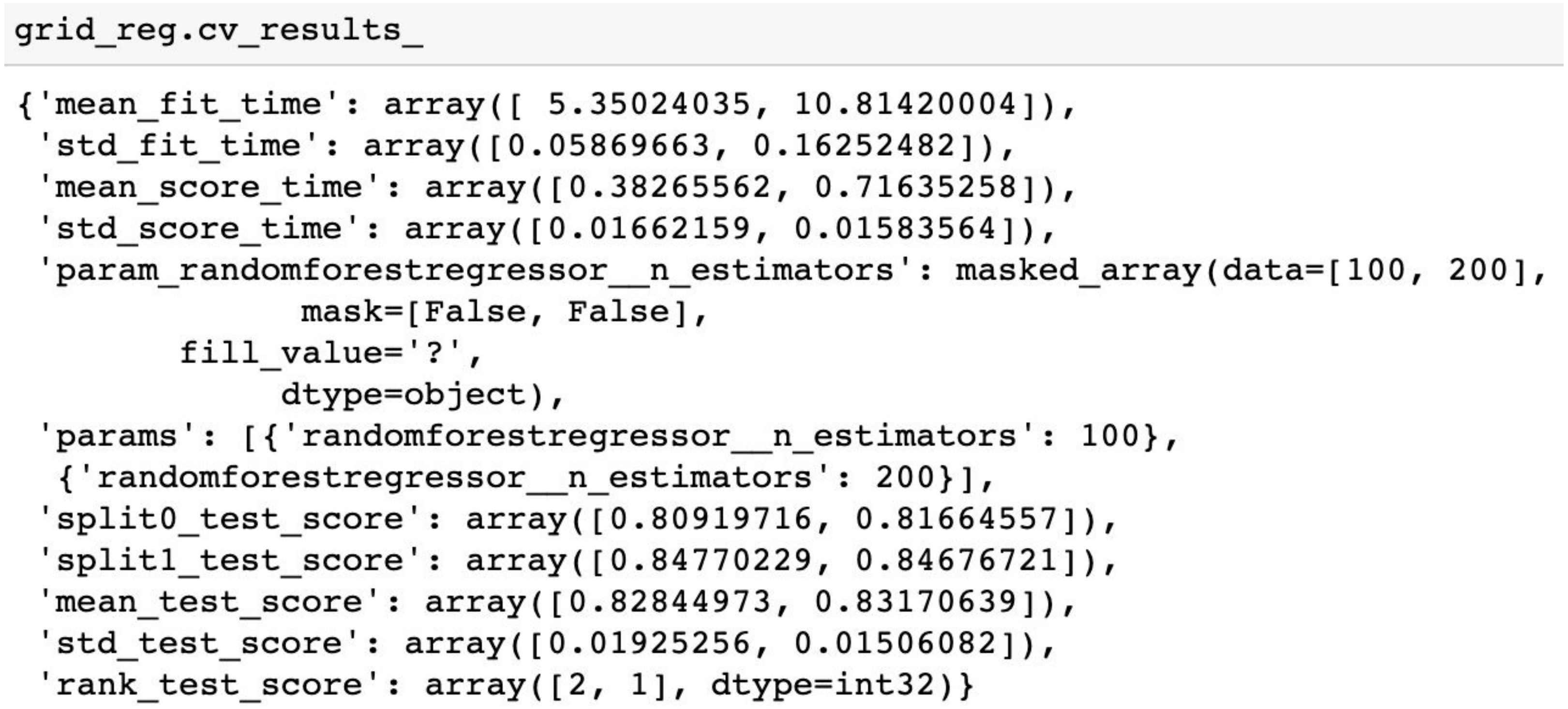
```
regr_forest = make_pipeline(StandardScaler(),RandomForestRegressor(max_depth=16))
from sklearn.model_selection import GridSearchCV

parameters = {'randomforestregressor__criterion':('squared_error', 'absolute_error'),
              'randomforestregressor__n_estimators':[200,400]
             }
forest_best = GridSearchCV(regr_forest, parameters, cv=2)
grid_reg=forest_best.fit(X_train, y_train)
```



```
regr_forest.get_params().keys()
```

```
dict_keys(['memory', 'steps', 'verbose', 'standardscaler', 'randomforestregressor', 'standardscaler__copy', 'standard
scaler__with_mean', 'standardscaler__with_std', 'randomforestregressor__bootstrap', 'randomforestregressor__ccp_alph
a', 'randomforestregressor__criterion', 'randomforestregressor__max_depth', 'randomforestregressor__max_features', 'r
andomforestregressor__max_leaf_nodes', 'randomforestregressor__max_samples', 'randomforestregressor__min_impurity_dec
rease', 'randomforestregressor__min_samples_leaf', 'randomforestregressor__min_samples_split', 'randomforestregressor
__min_weight_fraction_leaf', 'randomforestregressor__n_estimators', 'randomforestregressor__n_jobs', 'randomforestreg
ressor__oob_score', 'randomforestregressor__random_state', 'randomforestregressor__verbose', 'randomforestregressor__
warm_start'])
```

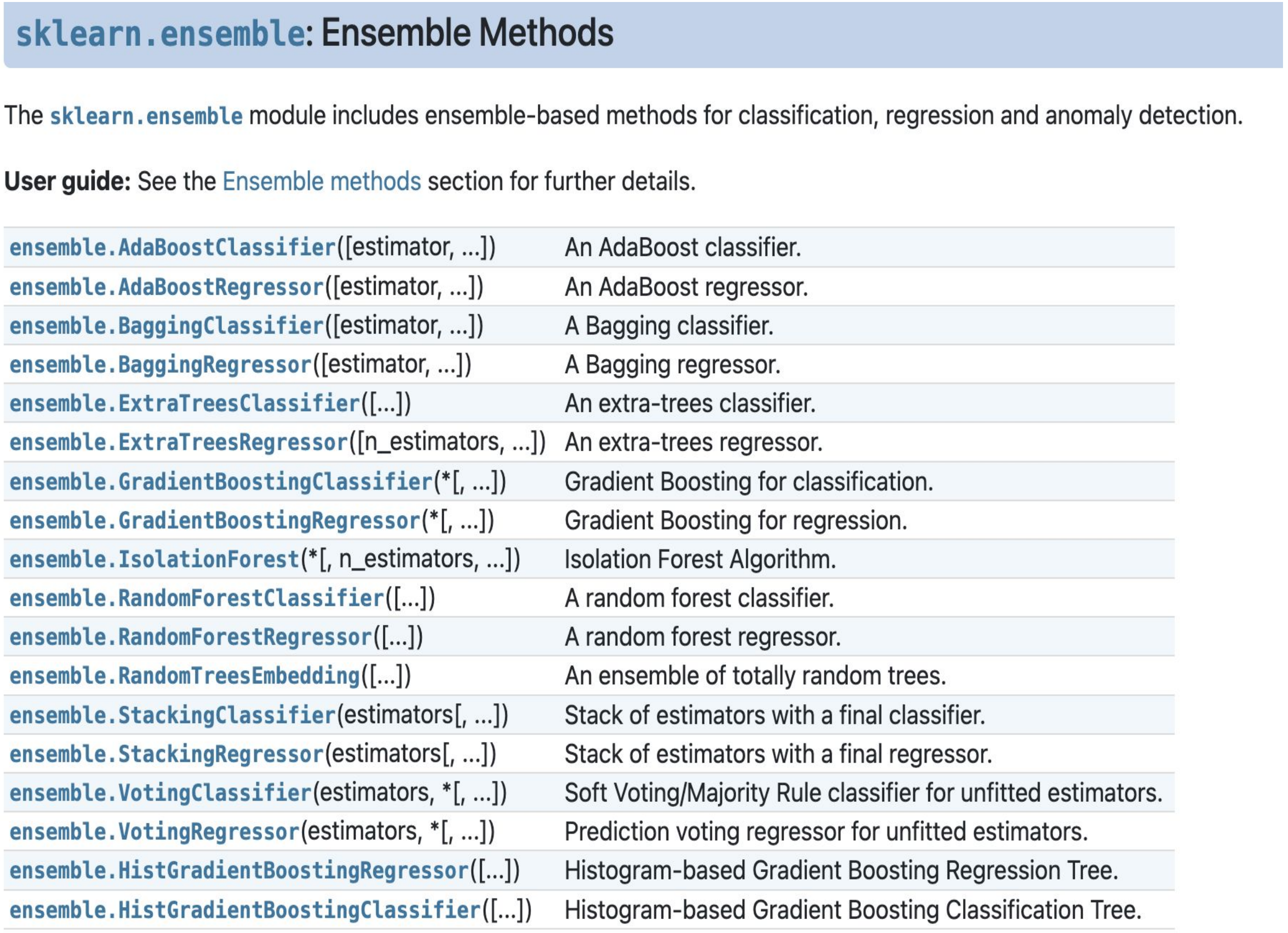
GridSearchCV Output



```
grid_reg.cv_results_
```

```
{'mean_fit_time': array([ 5.35024035, 10.81420004]),
 'std_fit_time': array([0.05869663, 0.16252482]),
 'mean_score_time': array([0.38265562, 0.71635258]),
 'std_score_time': array([0.01662159, 0.01583564]),
 'param_randomforestregressor__n_estimators': masked_array(data=[100, 200],
              mask=[False, False],
        fill_value='?',
             dtype=object),
 'params': [{'randomforestregressor__n_estimators': 100},
  {'randomforestregressor__n_estimators': 200}],
 'split0_test_score': array([0.80919716, 0.81664557]),
 'split1_test_score': array([0.84770229, 0.84676721]),
 'mean_test_score': array([0.82844973, 0.83170639]),
 'std_test_score': array([0.01925256, 0.01506082]),
 'rank_test_score': array([2, 1], dtype=int32)}
```

Ensemble Methods



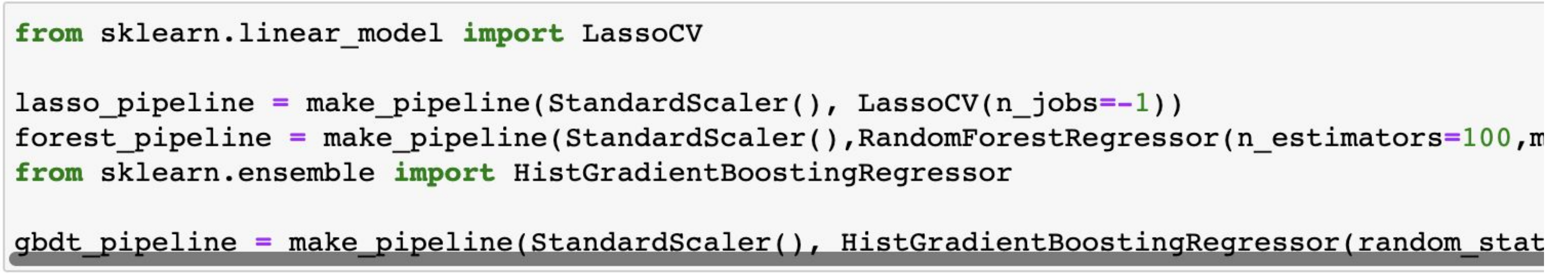
`sklearn.ensemble`: Ensemble Methods

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detection.

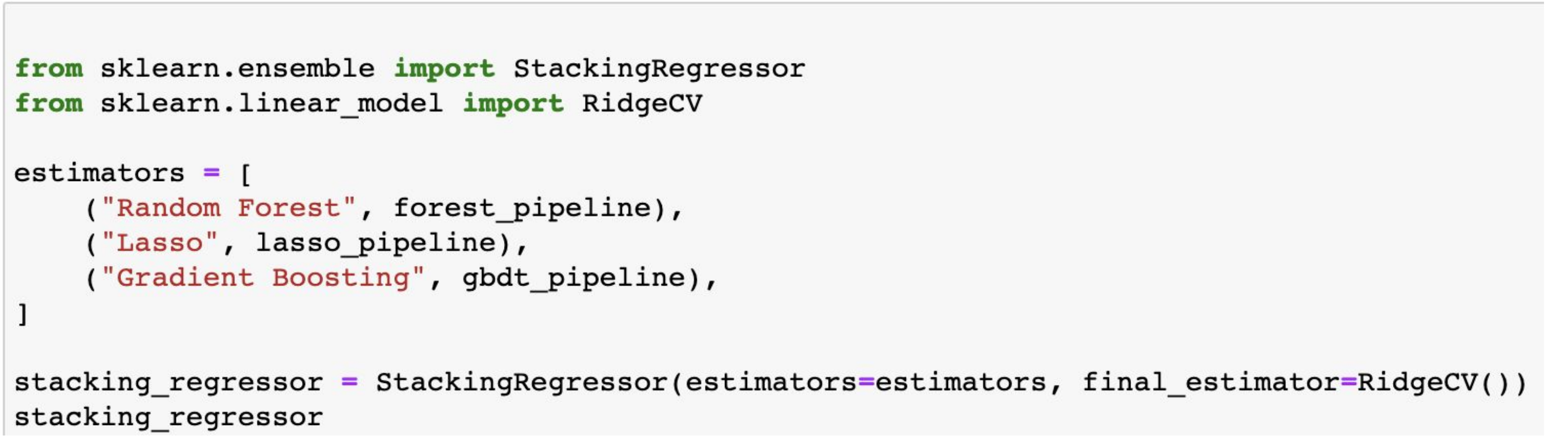
User guide: See the Ensemble methods section for further details.

`ensemble.AdaBoostClassifier`([estimator, ...])	An AdaBoost classifier.
`ensemble.AdaBoostRegressor`([estimator, ...])	An AdaBoost regressor.
`ensemble.BaggingClassifier`([estimator, ...])	A Bagging classifier.
`ensemble.BaggingRegressor`([estimator, ...])	A Bagging regressor.
`ensemble.ExtraTreesClassifier`([...])	An extra-trees classifier.
`ensemble.ExtraTreesRegressor`([n_estimators, ...])	An extra-trees regressor.
`ensemble.GradientBoostingClassifier`(*[, ...])	Gradient Boosting for classification.
`ensemble.GradientBoostingRegressor`(*[, ...])	Gradient Boosting for regression.
`ensemble.IsolationForest`(*[, n_estimators, ...])	Isolation Forest Algorithm.
`ensemble.RandomForestClassifier`([...])	A random forest classifier.
`ensemble.RandomForestRegressor`([...])	A random forest regressor.
`ensemble.RandomTreesEmbedding`([...])	An ensemble of totally random trees.
`ensemble.StackingClassifier`(estimators[, ...])	Stack of estimators with a final classifier.
`ensemble.StackingRegressor`(estimators[, ...])	Stack of estimators with a final regressor.
`ensemble.VotingClassifier`(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
`ensemble.VotingRegressor`(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.
`ensemble.HistGradientBoostingRegressor`([...])	Histogram-based Gradient Boosting Regression Tree.
`ensemble.HistGradientBoostingClassifier`([...])	Histogram-based Gradient Boosting Classification Tree.

Ensemble Methods (code)



```
from sklearn.linear_model import LassoCV

lasso_pipeline = make_pipeline(StandardScaler(), LassoCV(n_jobs=-1))
forest_pipeline = make_pipeline(StandardScaler(),RandomForestRegressor(n_estimators=100,m
from sklearn.ensemble import HistGradientBoostingRegressor

gbdt_pipeline = make_pipeline(StandardScaler(), HistGradientBoostingRegressor(random_stat
```



```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import RidgeCV

estimators = [
    ("Random Forest", forest_pipeline),
    ("Lasso", lasso_pipeline),
    ("Gradient Boosting", gbdt_pipeline),
]

stacking_regressor = StackingRegressor(estimators=estimators, final_estimator=RidgeCV())
stacking_regressor
```