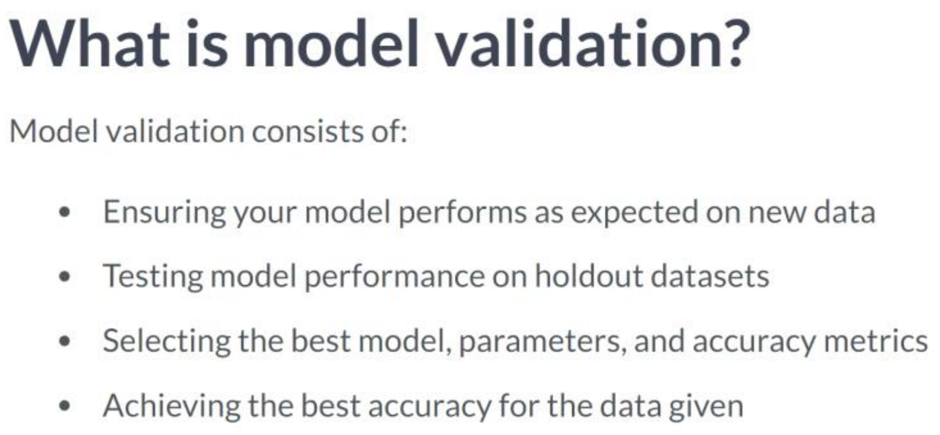
Model Validation in Python

Friday, 9 October 2020 12:44 AM

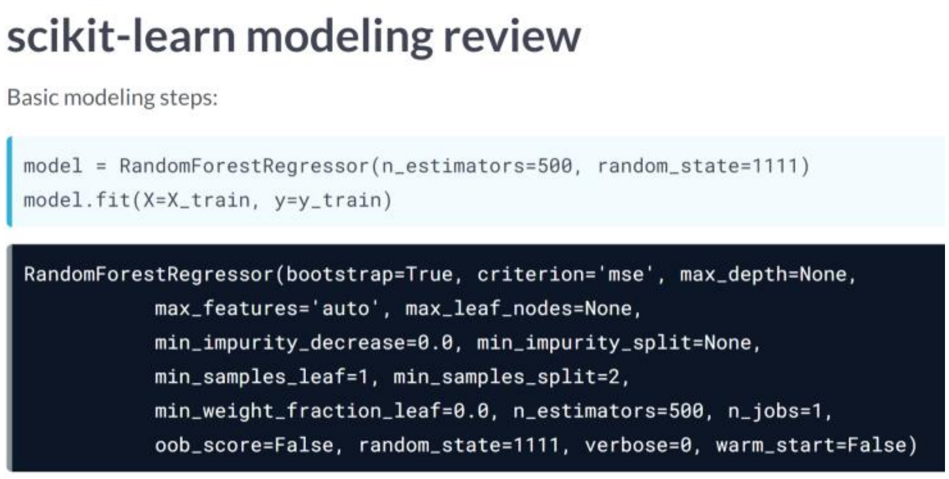
Basic Modeling in scikit-learn



What is model validation?

Model validation consists of:

- Ensuring your model performs as expected on new data
- Testing model performance on holdout datasets
- Selecting the best model, parameters, and accuracy metrics
- Achieving the best accuracy for the data given



scikit-learn modeling review

Basic modeling steps:

```
model = RandomForestRegressor(n_estimators=500, random_state=1111)
model.fit(X=X_train, y=y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
           oob_score=False, random_state=1111, verbose=0, warm_start=False)
```

Modeling review continued

```
predictions = model.predict(X_test)
print("{0:.2f}".format(mae(y_true=y_test, y_pred=predictions)))
```

```
10.84
```

Mean Absolute Error Formula

$$\frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

Seen vs. unseen data

Training data = seen data

```
model = RandomForestRegressor(n_estimators=500, random_state=1111)
model.fit(X_train, y_train)
train_predictions = model.predict(X_train)
```

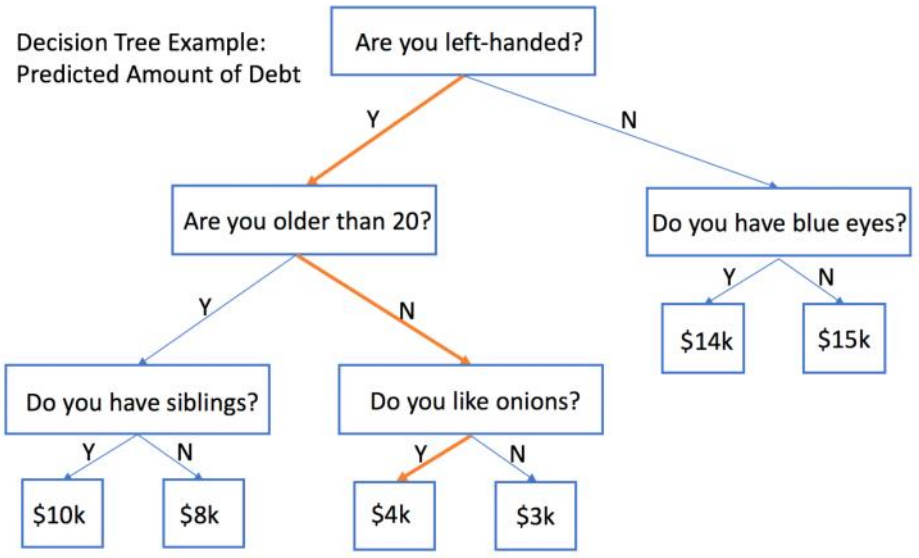
Testing data = unseen data

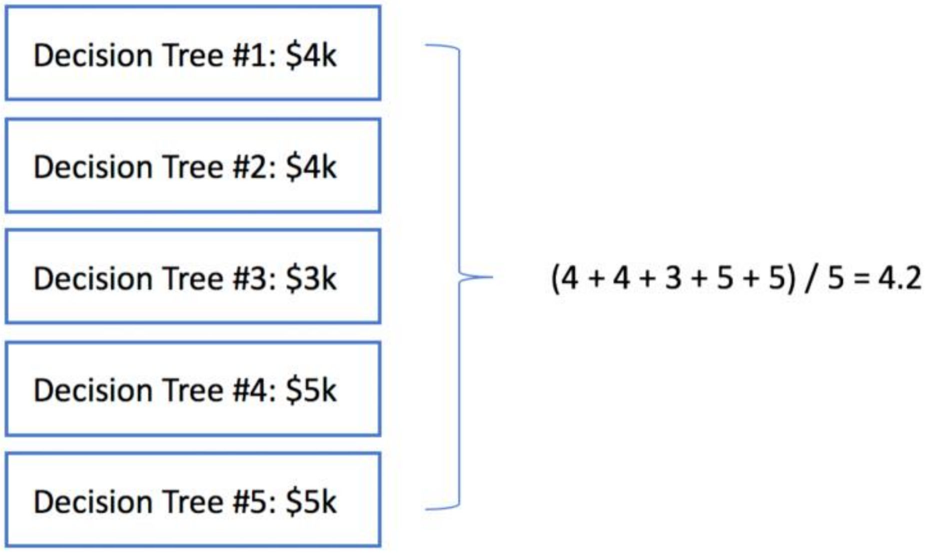
```
model = RandomForestRegressor(n_estimators=500, random_state=1111)
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)
```

Random forests in scikit-learn

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
```

```
rfr = RandomForestRegressor(random_state=1111)
rfc = RandomForestClassifier(random_state=1111)
```





Random forest parameters

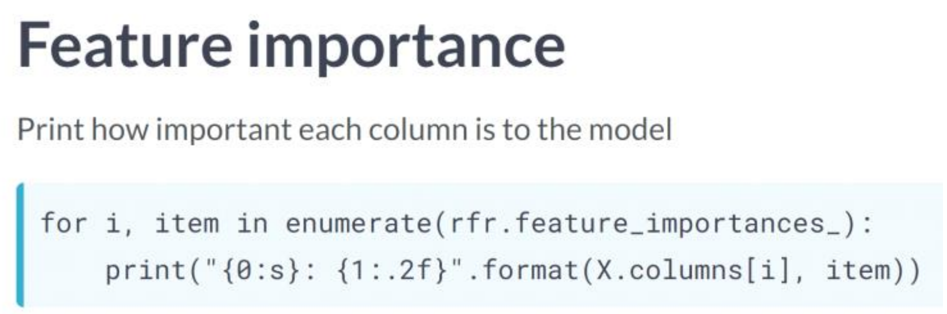
`n_estimators` : the number of trees in the forest

`max_depth` : the maximum depth of the trees

`random_state` : random seed

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=50, max_depth=10)
```

```
rfr = RandomForestRegressor(random_state=1111)
rfr.n_estimators = 50
rfr.max_depth = 10
```

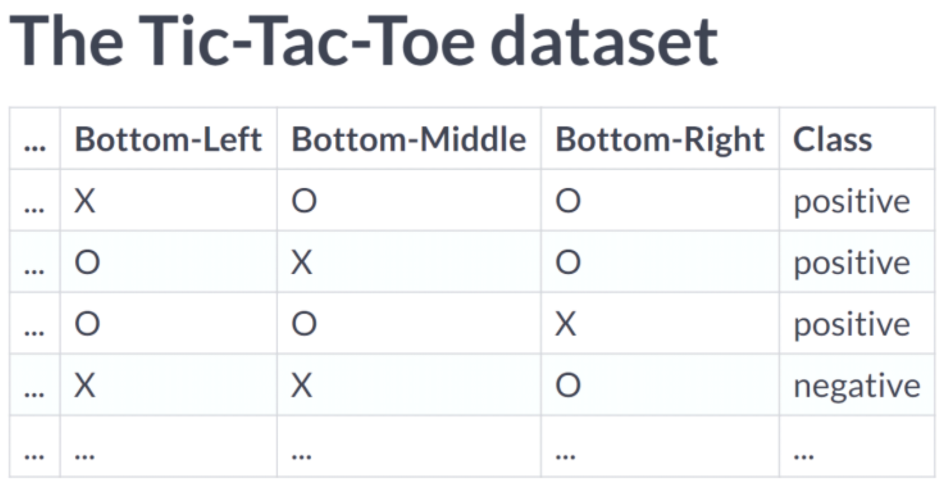


Feature importance

Print how important each column is to the model

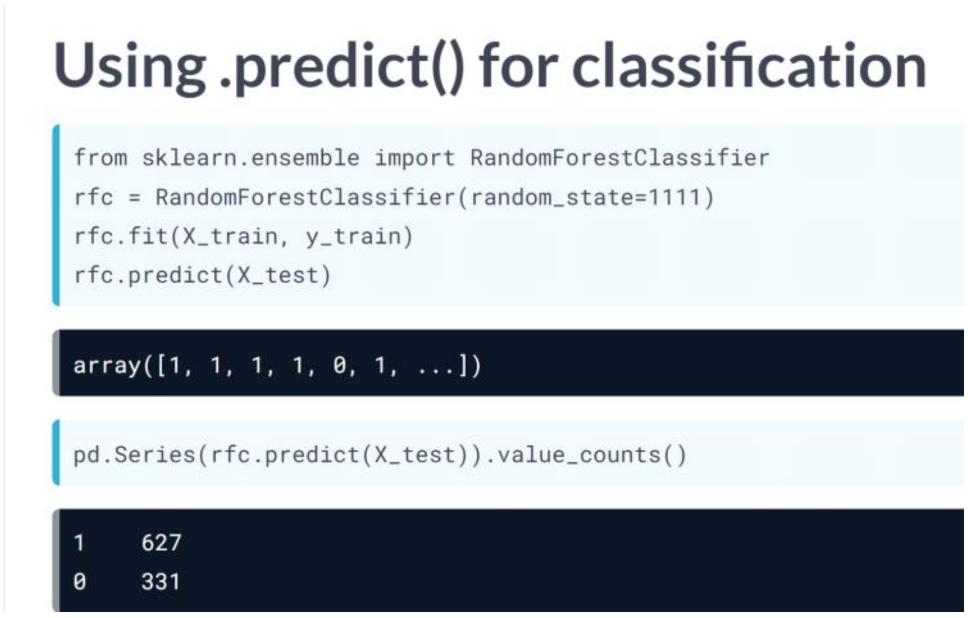
```
for i, item in enumerate(rfr.feature_importances_):
    print("{0:s}: {1:.2f}".format(X.columns[i], item))
```

The larger this number is, the more important that column was in the model.



The Tic-Tac-Toe dataset

...	Bottom-Left	Bottom-Middle	Bottom-Right	Class
...	X	O	O	positive
...	O	X	O	positive
...	O	O	X	positive
...	X	X	O	negative
...	...	...	...	...



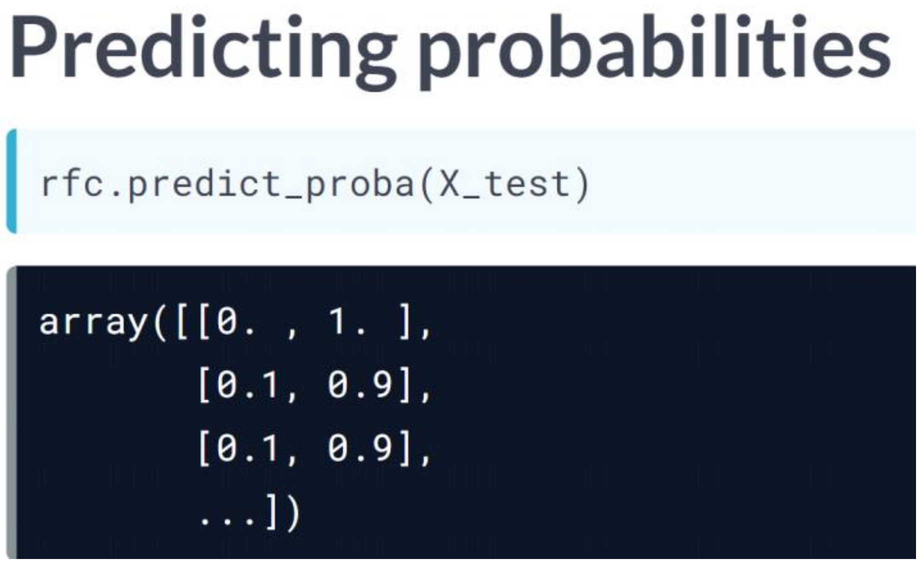
Using .predict() for classification

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=1111)
rfc.fit(X_train, y_train)
rfc.predict(X_test)
```

```
array([1, 1, 1, 1, 0, 1, ...])
```

```
pd.Series(rfc.predict(X_test)).value_counts()
```

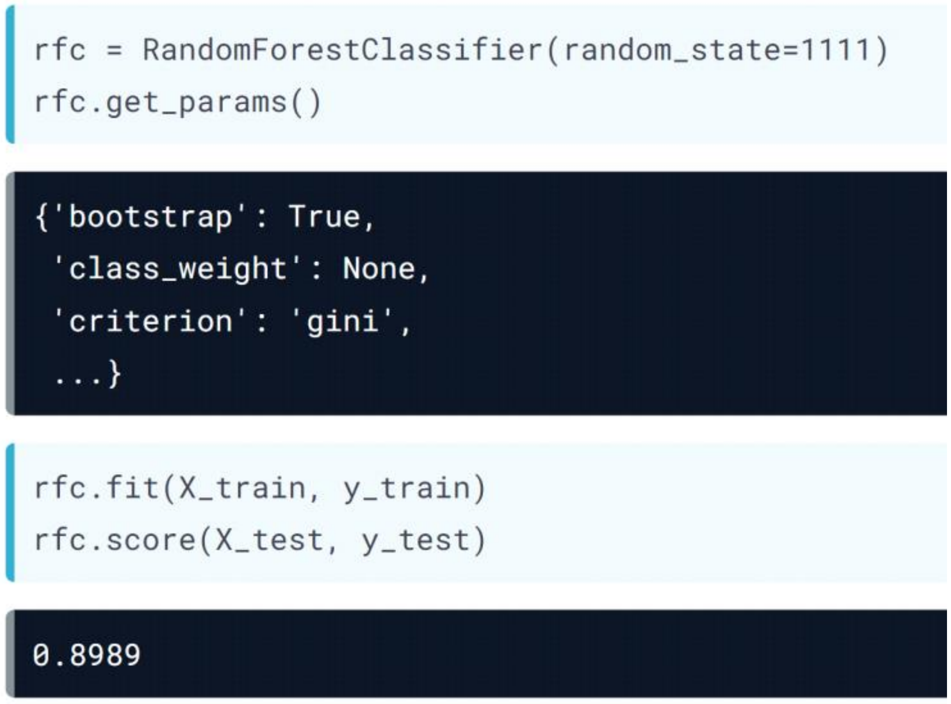
```
1    627
0    331
```



Predicting probabilities

```
rfc.predict_proba(X_test)
```

```
array([[0. , 1. ],
       [0.1, 0.9],
       [0.1, 0.9],
       ...])
```



```
rfc = RandomForestClassifier(random_state=1111)
rfc.get_params()
```

```
{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 ...}
```

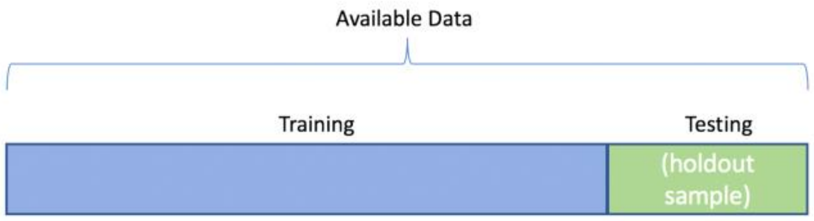
```
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
```

```
0.8989
```

Validation Basics

Traditional train/test split

- Seen data (used for training)
- Unseen data (unavailable for training)



Dataset definitions and ratios

Dataset	Definition
Train	The sample of data used when fitting models
Test (holdout sample)	The sample of data used to assess model performance

Ratio Examples

- 80:20
- 90:10 (used when we have little data)
- 70:30 (used when model is computationally expensive)

The X and y datasets

```
import pandas as pd

tic_tac_toe = pd.read_csv("tic-tac-toe.csv")
X = pd.get_dummies(tic_tac_toe.iloc[:,0:9])
y = tic_tac_toe.iloc[:, 9]
```

Python courses covering dummy variables:

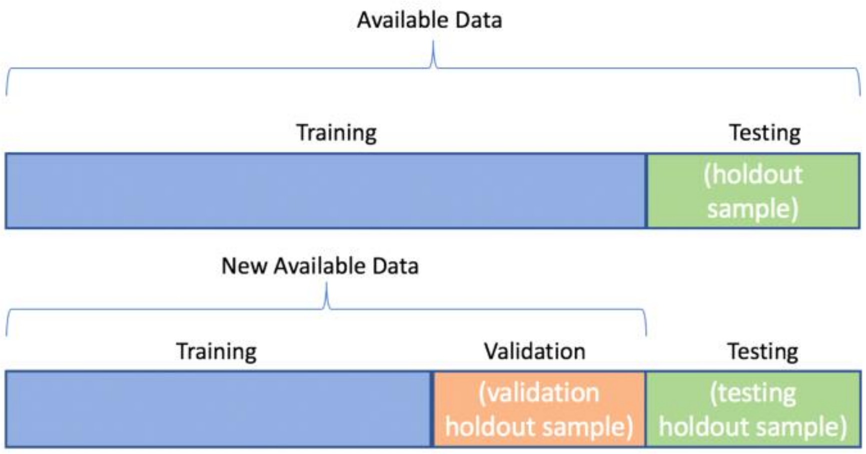
- Supervised Learning
- Preprocessing for Machine Learning

Creating holdout samples

```
X_train, X_test, y_train, y_test  =\
    train_test_split(X, y, test_size=0.2, random_state=1111)
```

Parameters:

- `test_size`
- `train_size`
- `random_state`



Train, validation, test continued

```
X_temp, X_test, y_temp, y_test  =\
    train_test_split(X, y, test_size=0.2, random_state=1111)
```

```
X_train, X_val, y_train, y_val =\
    train_test_split(X_temp, y_temp, test_size=0.25, random_state=11111)
```

Regression models

12.2 points

15 gallons of gas

$1,323,492

6 new puppies

4,320 people

Mean absolute error (MAE)

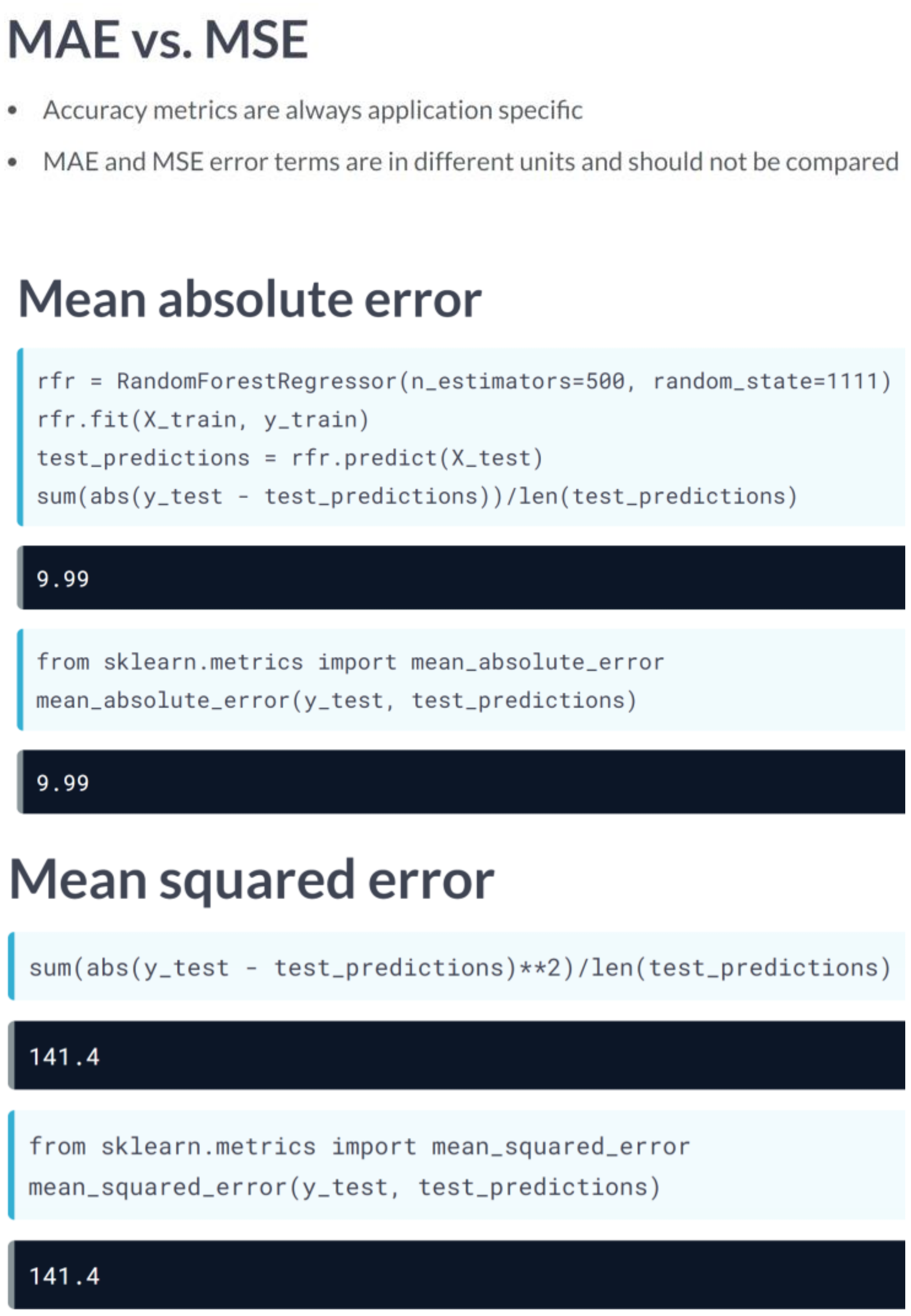
$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

- Simplest and most intuitive metric
- Treats all points equally
- Not sensitive to outliers

Mean squared error (MSE)

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

- Most widely used regression metric
- Allows outlier errors to contribute more to the overall error
- Random family road trips could lead to large errors in predictions



MAE vs. MSE

- Accuracy metrics are always application specific
- MAE and MSE error terms are in different units and should not be compared

Mean absolute error

```python
rfr = RandomForestRegressor(n_estimators=500, random_state=1111)
rfr.fit(X_train, y_train)
test_predictions = rfr.predict(X_test)
sum(abs(y_test - test_predictions))/len(test_predictions)
```

```
9.99
```

```python
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, test_predictions)
```

```
9.99
```

Mean squared error

```python
sum(abs(y_test - test_predictions)**2)/len(test_predictions)
```

```
141.4
```

```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, test_predictions)
```

```
141.4
```

Accuracy for a subset of data

```
chocolate_preds = rfr.predict(X_test[X_test[:, 1] == 1])
mean_absolute_error(y_test[X_test[:, 1] == 1], chocolate_preds)
```

```
8.79
```

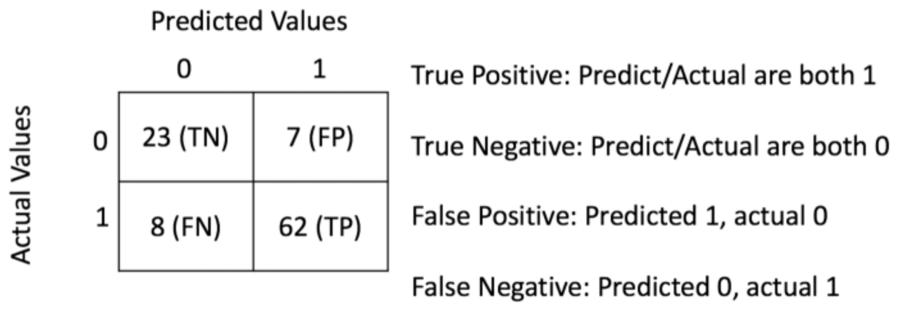
```
nonchocolate_preds = rfr.predict(X_test[X_test[:, 1] == 0])
mean_absolute_error(y_test[X_test[:, 1] == 0], nonchocolate_preds)
```

```
10.99
```

Classification metrics

- **Precision**
- **Recall** (also called sensitivity)
- **Accuracy**
- Specificity
- F1-Score, and its variations
- ...

Confusion matrix



Actual Values		Predicted Values 0	Predicted Values 1
	0	23 (TN)	7 (FP)
	1	8 (FN)	62 (TP)

True Positive: Predict/Actual are both 1

True Negative: Predict/Actual are both 0

False Positive: Predicted 1, actual 0

False Negative: Predicted 0, actual 1

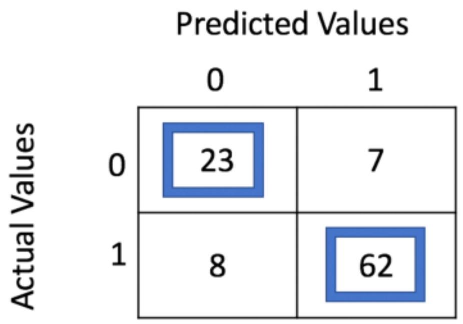
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, test_predictions)
print(cm)
```

```
array([[ 23,   7],
       [  8, 62]])
```

```
cm[<true_category_index>, <predicted_category_index>]
cm[1, 0]
```

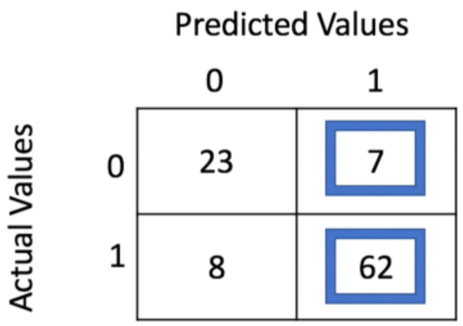
```
8
```

Accuracy



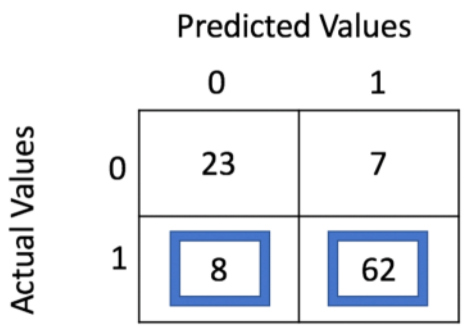
$$\frac{23(\text{TN})+62(\text{TP})}{23+7+8+62} = .85$$

Precision



$$\frac{62(\text{TP})}{62(\text{TP})+7(\text{FP})} = .90$$

Recall



		Predicted Values	
		0	1
Actual Values	0	23	7
	1	8	62

$$\frac{62(\text{TP})}{62(\text{TP})+8(\text{FN})} = .885$$

Accuracy, precision, recall

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
accuracy_score(y_test, test_predictions)
```

```
.85
```

```
precision_score(y_test, test_predictions)
```

```
.8986
```

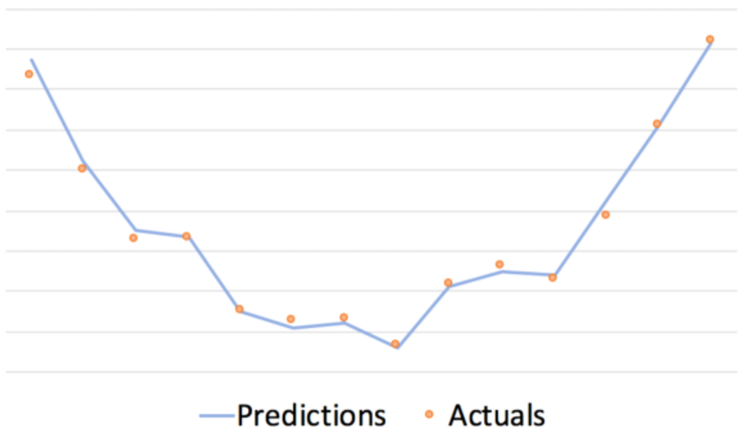
```
recall_score(y_test, test_predictions)
```

```
.8857
```

Variance

- Variance: following the training data too closely
  - Fails to generalize to the test data
  - Low training error but high testing error
  - Occurs when models are overfit and have high complexity

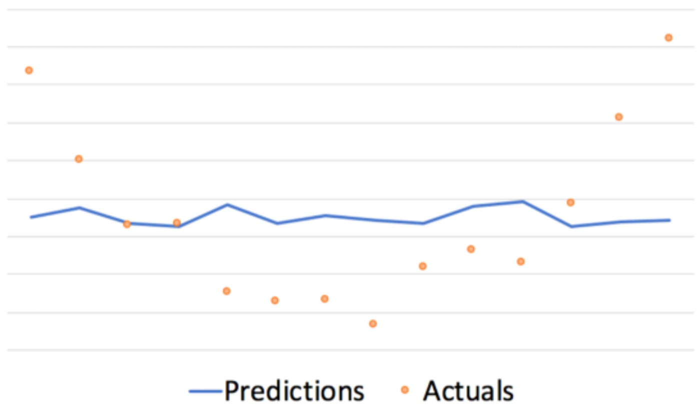
Overfitting models (high variance)



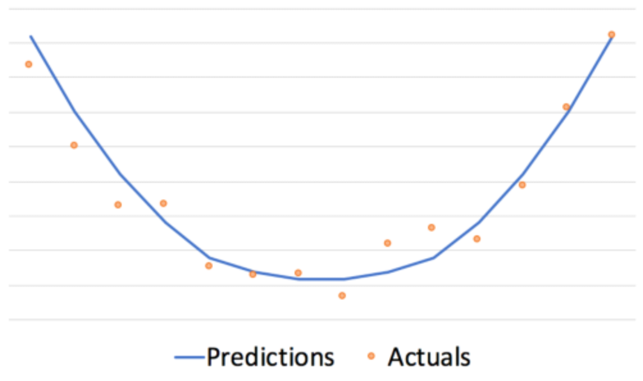
Bias

- Bias: failing to find the relationship between the data and the response
  - High training/testing error
  - Occurs when models are underfit

Underfitting models (high bias)



Optimal performance



- Bias-Variance Tradeoff

Parameters causing over/under fitting

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=4)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

```
Training: .84
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .77
```

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=14)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

```
Training: 1.0
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .83
```

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=10)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

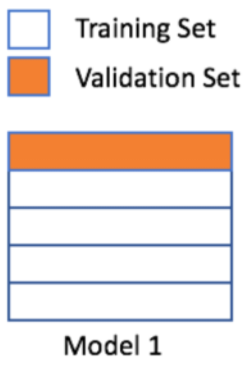
```
Training: .89
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .86
```

Cross Validation

Transition validation



```
X_train, X_val, y_train, y_val =
    train_test_split(X, y,
    test_size=0.2)

rf = RandomForestRegressor()

rf.fit(X_train, y_train)
out_of_sample = rf.predict(X_test)
print(mae(y_test, out_of_sample))
```

```
10.24
```

Traditional training splits

```
cd = pd.read_csv("candy-data.csv")
s1 = cd.sample(60, random_state=1111)
s2 = cd.sample(60, random_state=1112)
```

Overlapping candies:

```
print(len([i for i in s1.index if i in s2.index]))
```

```
39
```

Traditional training splits

Chocolate Candies:

```
print(s1.chocolate.value_counts()[0])
print(s2.chocolate.value_counts()[0])
```

```
34
30
```

The split matters

Sample 1 Testing Error

```
print('Testing error: {0:.2f}'.format(mae(s1_y_test, rfr.predict(s1_X_test))))
```

```
10.32
```

Sample 2 Testing Error

```
print('Testing error: {0:.2f}'.format(mae(s2_y_test, rfr.predict(s2_X_test))))
```

```
11.56
```

Train, validation, test

```
X_temp, X_val, y_temp, y_val = train_test_split(..., random_state=1111)
X_train, X_test, y_train, y_test = train_test_split(..., random_state=1111)

rfr = RandomForestRegressor(n_estimators=25, random_state=1111, max_features=4)
rfr.fit(X_train, y_train)
print('Validation error: {0:.2f}'.format(mae(y_test, rfr.predict(X_test))))
```

```
9.18
```

```
print('Testing error: {0:.2f}'.format(mae(y_val, rfr.predict(X_val))))
```

```
8.98
```

Round 2

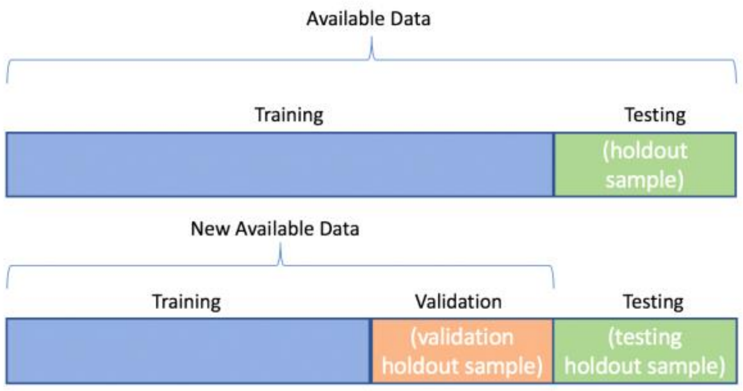
```
X_temp, X_val, y_temp, y_val = train_test_split(..., random_state=1171)
X_train, X_test, y_train, y_test = train_test_split(..., random_state=1171)

rfr = RandomForestRegressor(n_estimators=25, random_state=1111, max_features=4)
rfr.fit(X_train, y_train)
print('Validation error: {0:.2f}'.format(mae(y_test, rfr.predict(X_test))))
```

```
8.73
```

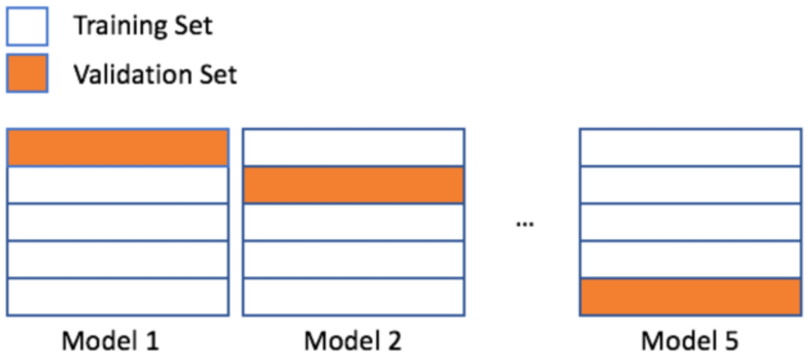
```
print('Testing error: {0:.2f}'.format(mae(y_val, rfr.predict(X_val))))
```

```
10.91
```

Cross-validation



Cross-validation



`n_splits` : number of cross-validation splits

`shuffle` : boolean indicating to shuffle data before splitting

`random_state` : random seed

```
from sklearn.model_selection import KFold

X = np.array(range(40))
y = np.array([0] * 20 + [1] * 20)

kf = KFold(n_splits=5)
splits = kf.split(X)
```

```
kf = KFold(n_splits=5)
splits = kf.split(X)
for train_index, test_index in splits:
    print(len(train_index), len(test_index))
```

```
32 8 32 8 32 8 32 8 32 8
```

```
# Print one of the index sets:
print(train_index, test_index)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 ...]
[32 33 34 35 36 37 38 39]
```

```
rfr = RandomForestRegressor(n_estimators=25, random_state=1111)
errors = []
for train_index, val_index in splits:
    X_train, y_train = X[train_index], y[train_index]
    X_val, y_val = X[val_index], y[val_index]

    rfr.fit(X_train, y_train)
    predictions = rfr.predict(X_test)
    errors.append(<some_accuracy_metric>)
print(np.mean(errors))
```

```
4.25
```

cross_val_score()

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

`estimator` : the model to use

`X` : the predictor dataset

`y` : the response array

`cv` : the number of cross-validation splits

```
cross_val_score(estimator=rfc, X=X, y=y, cv=5)
```

Using scoring and make_scorer

The cross_val_score `scoring` parameter:

```
# Load the Methods
from sklearn.metrics import mean_absolute_error, make_scorer
```

```
# Create a scorer
mae_scorer = make_scorer(mean_absolute_error)
```

```
# Use the scorer
cross_val_score(<estimator>, <X>, <y>, cv=5, scoring=mae_scorer)
```

Load all of the `sklearn` methods

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, make_scorer
```

Create a model and a scorer

```
rfc = RandomForestRegressor(n_estimators=20, max_depth=5, random_state=1111)
mse = make_scorer(mean_squared_error)
```

Run `cross_val_score()`

```
cv_results = cross_val_score(rfc, X, y, cv=5, scoring=mse)
```

Accessing the results

```
print(cv_results)
```

```
[196.765, 108.563, 85.963, 222.594, 140.942]
```

Report the mean and standard deviation:

```
print('The mean: {}'.format(cv_results.mean()))
print('The std: {}'.format(cv_results.std()))
```

```
The mean: 150.965
```

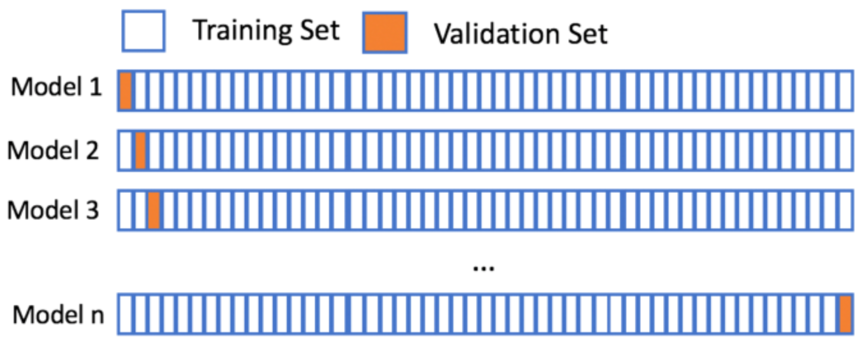
```
The std: 51.676
```

The smaller the standard deviation, the tighter your 5 means were.

This indicates that the actual accuracy for new data will

probably match the mean of the cross-validation score fairly well.

LOOCV



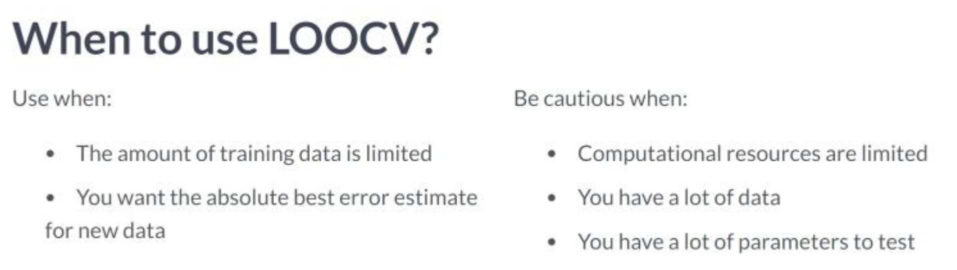
In leave-one-out-cross-validation, we are going to implement KFold

cross-validation, where k is equal to n, the number of observations in the data.

This means that every single point will be used in a validation set, completely by itself.

For the first model, we will use all of

the data for training, except for the first point, which will be used for validation.



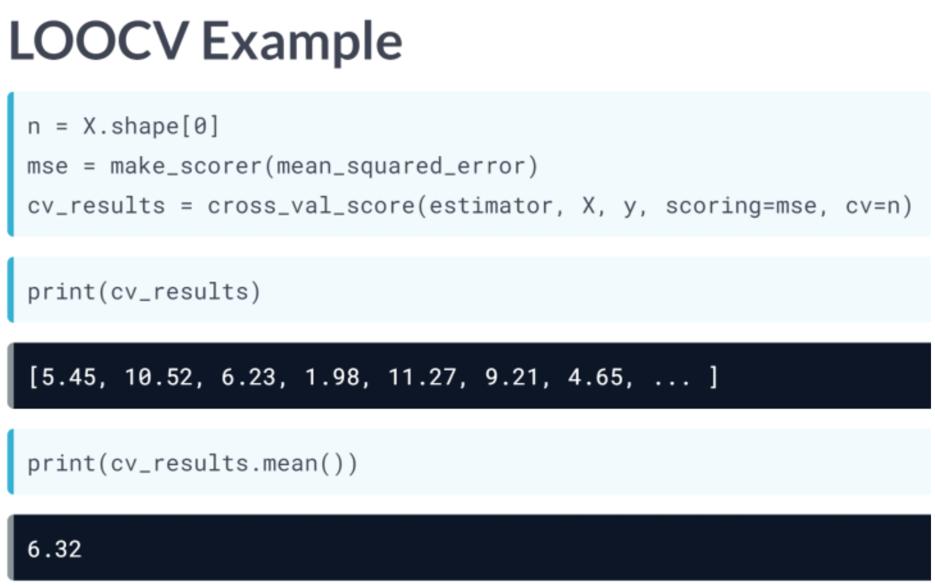
When to use LOOCV?

Use when:

- The amount of training data is limited
- You want the absolute best error estimate for new data

Be cautious when:

- Computational resources are limited
- You have a lot of data
- You have a lot of parameters to test



LOOCV Example

```
n = X.shape[0]
mse = make_scorer(mean_squared_error)
cv_results = cross_val_score(estimator, X, y, scoring=mse, cv=n)
```

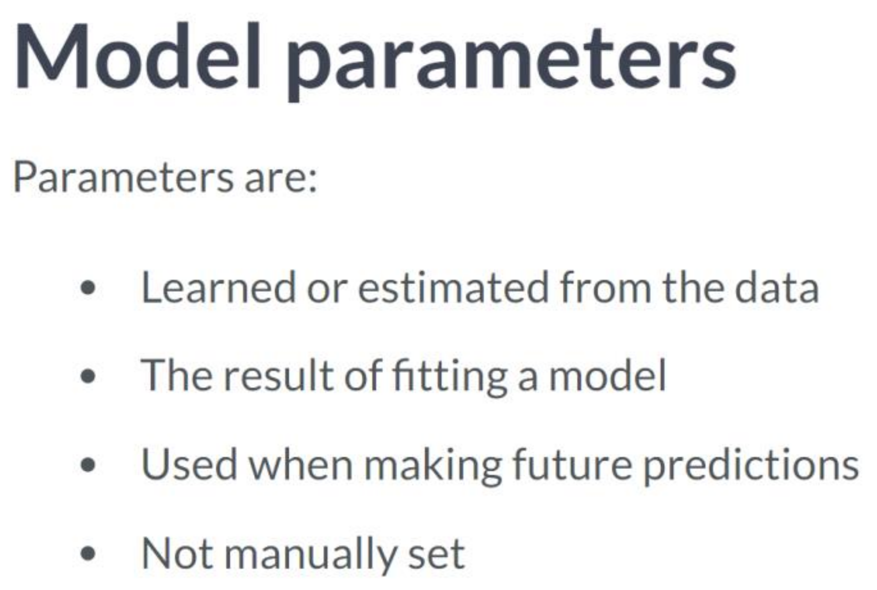
```
print(cv_results)
```

```
[5.45, 10.52, 6.23, 1.98, 11.27, 9.21, 4.65, ... ]
```

```
print(cv_results.mean())
```

```
6.32
```

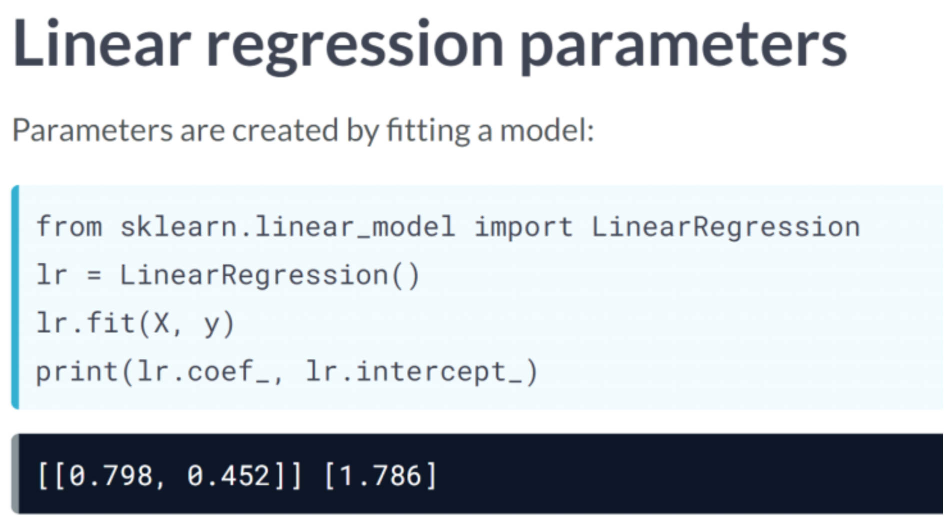
Selecting the best model with Hyperparameter tuning.



Model parameters

Parameters are:

- Learned or estimated from the data
- The result of fitting a model
- Used when making future predictions
- Not manually set

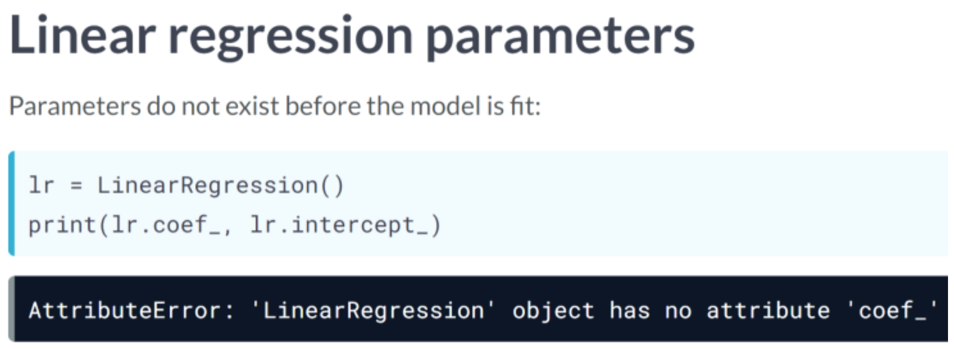


Linear regression parameters

Parameters are created by fitting a model:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X, y)
print(lr.coef_, lr.intercept_)
```

```
[[0.798, 0.452]] [1.786]
```

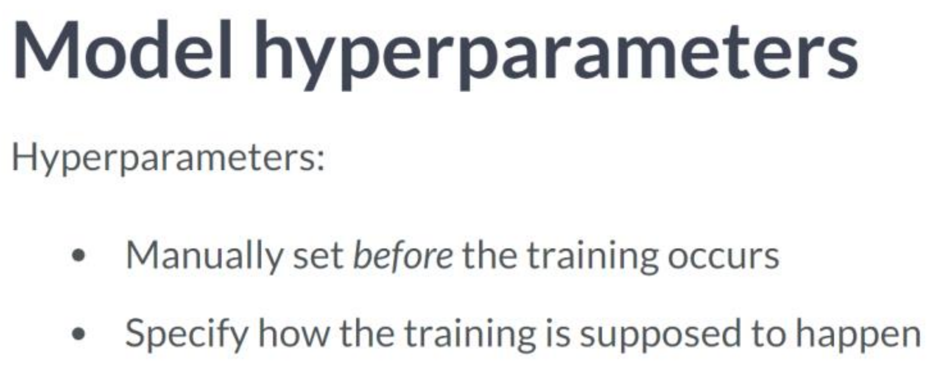


Linear regression parameters

Parameters do not exist before the model is fit:

```
lr = LinearRegression()
print(lr.coef_, lr.intercept_)
```

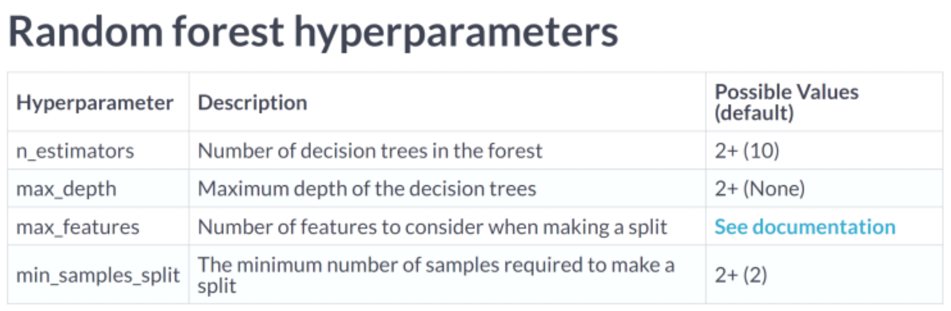
```
AttributeError: 'LinearRegression' object has no attribute 'coef_'
```



Model hyperparameters

Hyperparameters:

- Manually set *before* the training occurs
- Specify how the training is supposed to happen



Random forest hyperparameters

Hyperparameter	Description	Possible Values (default)
n_estimators	Number of decision trees in the forest	2+ (10)
max_depth	Maximum depth of the decision trees	2+ (None)
max_features	Number of features to consider when making a split	See documentation
min_samples_split	The minimum number of samples required to make a split	2+ (2)

What is hyperparameter tuning?

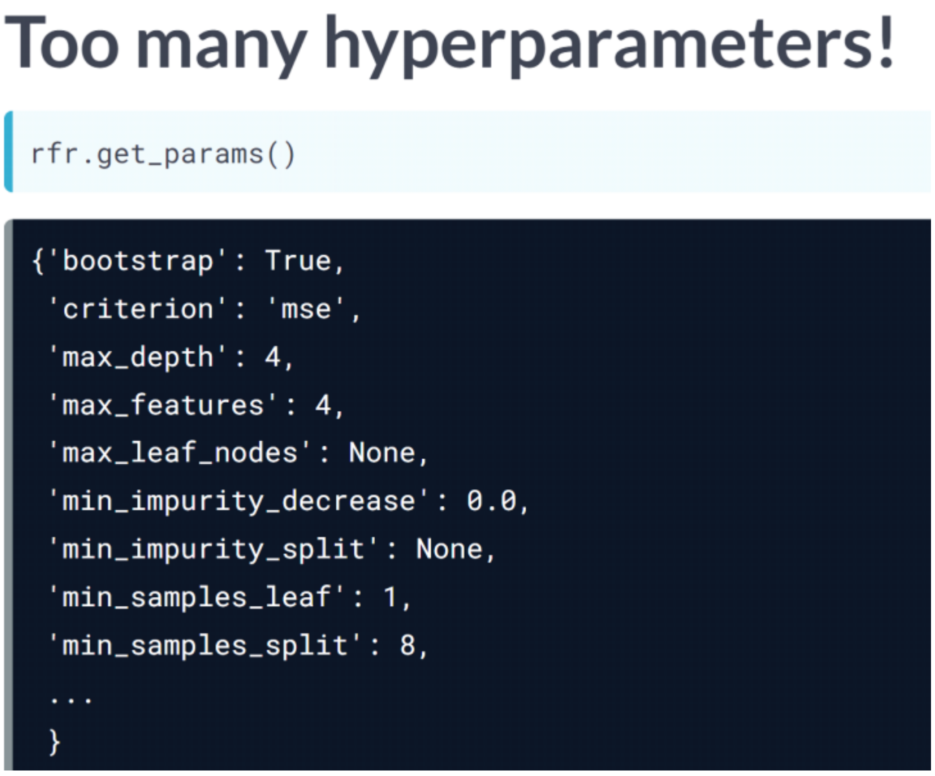
Hyperparameter tuning:

- Select hyperparameters
- Run a single model type at different value sets
- Create ranges of possible values to select from
- Specify a single accuracy metric

Specifying ranges

```
depth = [4, 6, 8, 10, 12]
samples = [2, 4, 6, 8]
features = [2, 4, 6, 8, 10]
# Specify hyperparameters
rfc = RandomForestRegressor(
    n_estimators=100, max_depth=depth[0],
    min_samples_split=samples[3], max_features=features[1])
rfr.get_params()
```

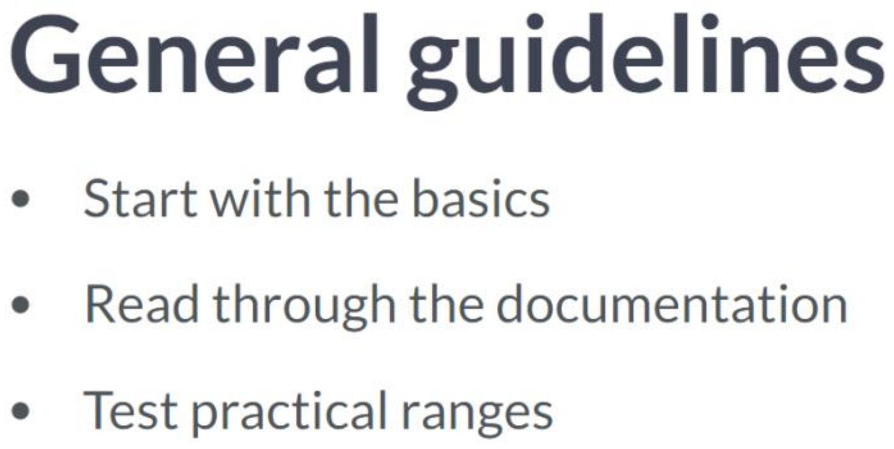
```
{'bootstrap': True,
 'criterion': 'mse'
 ...
}
```



Too many hyperparameters!

```
rfr.get_params()
```

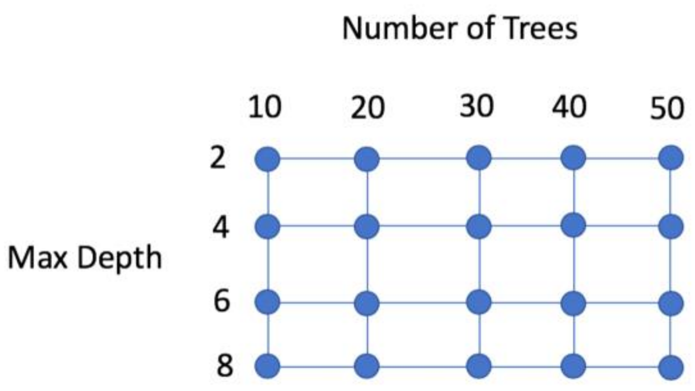
```
{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': 4,
 'max_features': 4,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 ...
 }
```



General guidelines

- Start with the basics
- Read through the documentation
- Test practical ranges

Grid searching hyperparameters



Grid searching continued

Benefits:

- Tests every possible combination

Drawbacks:

- Additional hyperparameters increase training time exponentially

Better methods

- **Random searching**
- **Bayesian optimization**

Random search

```
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV()
```

Parameter Distribution:

```
param_dist = {"max_depth": [4, 6, 8, None],
              "max_features": range(2, 11),
              "min_samples_split": range(2, 11)}
```

Setting RandomizedSearchCV parameters

```
param_dist = {"max_depth": [4, 6, 8, None],
              "max_features": range(2, 11),
              "min_samples_split": range(2, 11)}
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import make_scorer, mean_absolute_error

rfr = RandomForestRegressor(n_estimators=20, random_state=1111)
scorer = make_scorer(mean_absolute_error)
```

RandomizedSearchCV implemented

Setting up the random search:

```
random_search =\
    RandomizedSearchCV(estimator=rfr,
                       param_distributions=param_dist,
                       n_iter=40,
                       cv=5)
```

- We cannot do hyperparameter tuning without understanding model validation
- Model validation allows us to compare multiple models and parameter sets

RandomizedSearchCV implemented

Setting up the random search:

```
random_search =\
    RandomizedSearchCV(estimator=rfr,
                       param_distributions=param_dist,
                       n_iter=40,
                       cv=5)
```

Complete the random search:

```
random_search.fit(X, y)
```

```
# Best Score
rs.best_score_
```

```
5.45
```

```
# Best Parameters
rs.best_params_
```

```
{'max_depth': 4, 'max_features': 8, 'min_samples_split': 4}
```

```
# Best Estimator
rs.best_estimator_
```

Other attributes

```
rs.cv_results_
rs.cv_results_['mean_test_score']
```

```
array([5.45, 6.23, 5.87, 5,91, 5,67])
```

```
# Selected Parameters:
rs.cv_results_['params']
```

```
[{'max_depth': 10, 'min_samples_split': 8, 'n_estimators': 25},
 {'max_depth': 4, 'min_samples_split': 8, 'n_estimators': 50},
 ...]
```

Using .cv_results_

Group the max depths:

```
max_depth = [item['max_depth'] for item in rs.cv_results_['params']]
scores = list(rs.cv_results_['mean_test_score'])
d = pd.DataFrame([max_depth, scores]).T
d.columns = ['Max Depth', 'Score']
d.groupby(['Max Depth']).mean()
```

```
Max Depth  Score
2.0        0.677928
4.0        0.753021
6.0        0.817219
8.0        0.879136
10.0       0.896821
```

Other attributes continued

Uses of the output:

- Visualize the effect of each parameter
- Make inferences on which parameters have big impacts on the results

```
Max Depth  Score
2.0        0.677928
4.0        0.753021
6.0        0.817219
8.0        0.879136
10.0       0.896821
```

Selecting the best model

`rs.best_estimator_` contains the information of the best model

```
rs.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=8,
           max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=12, min_weight_fraction_leaf=0.0,
           n_estimators=20, n_jobs=1, oob_score=False, random_state=1111,
           verbose=0, warm_start=False)
```

Comparing types of models

Random forest:

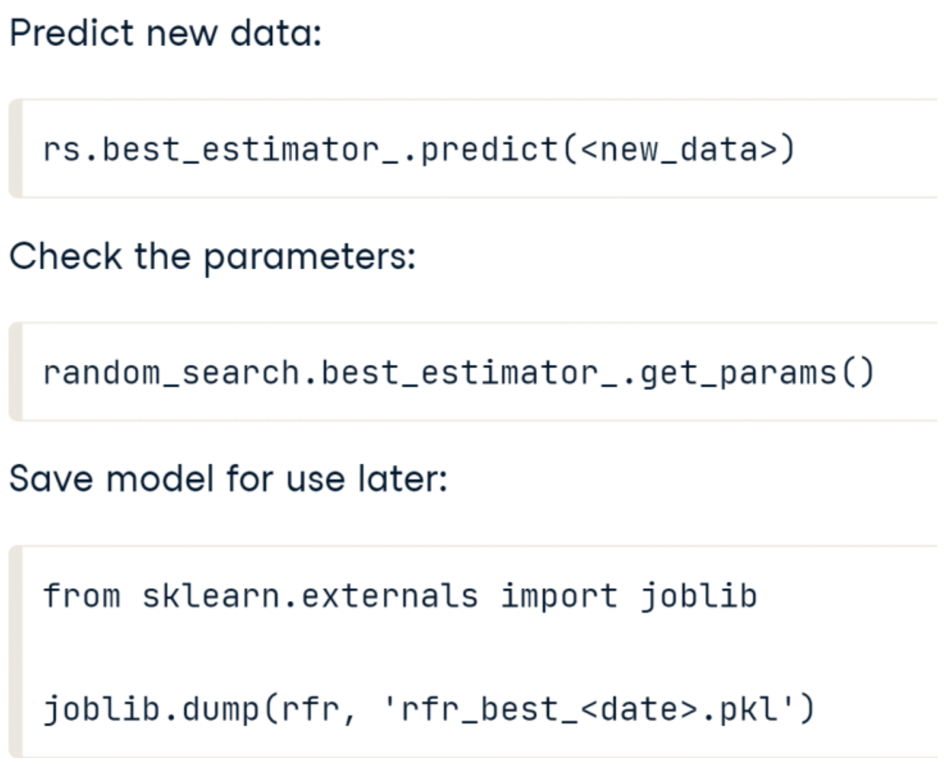
```
rfr.score(X_test, y_test)
```

```
6.39
```

Gradient Boosting:

```
gb.score(X_test, y_test)
```

```
6.23
```



Predict new data:

```
rs.best_estimator_.predict(<new_data>)
```

Check the parameters:

```
random_search.best_estimator_.get_params()
```

Save model for use later:

```
from sklearn.externals import joblib

joblib.dump(rfr, 'rfr_best_<date>.pkl')
```