DATA 301: Cross Validation and Hyperparameter tuning

Topics

Model training overview
Bias variance
Under and overfitting
Getting a good fit for Trees
Cross Validation
Hyperparameter tuning using Optuna

Model training overview

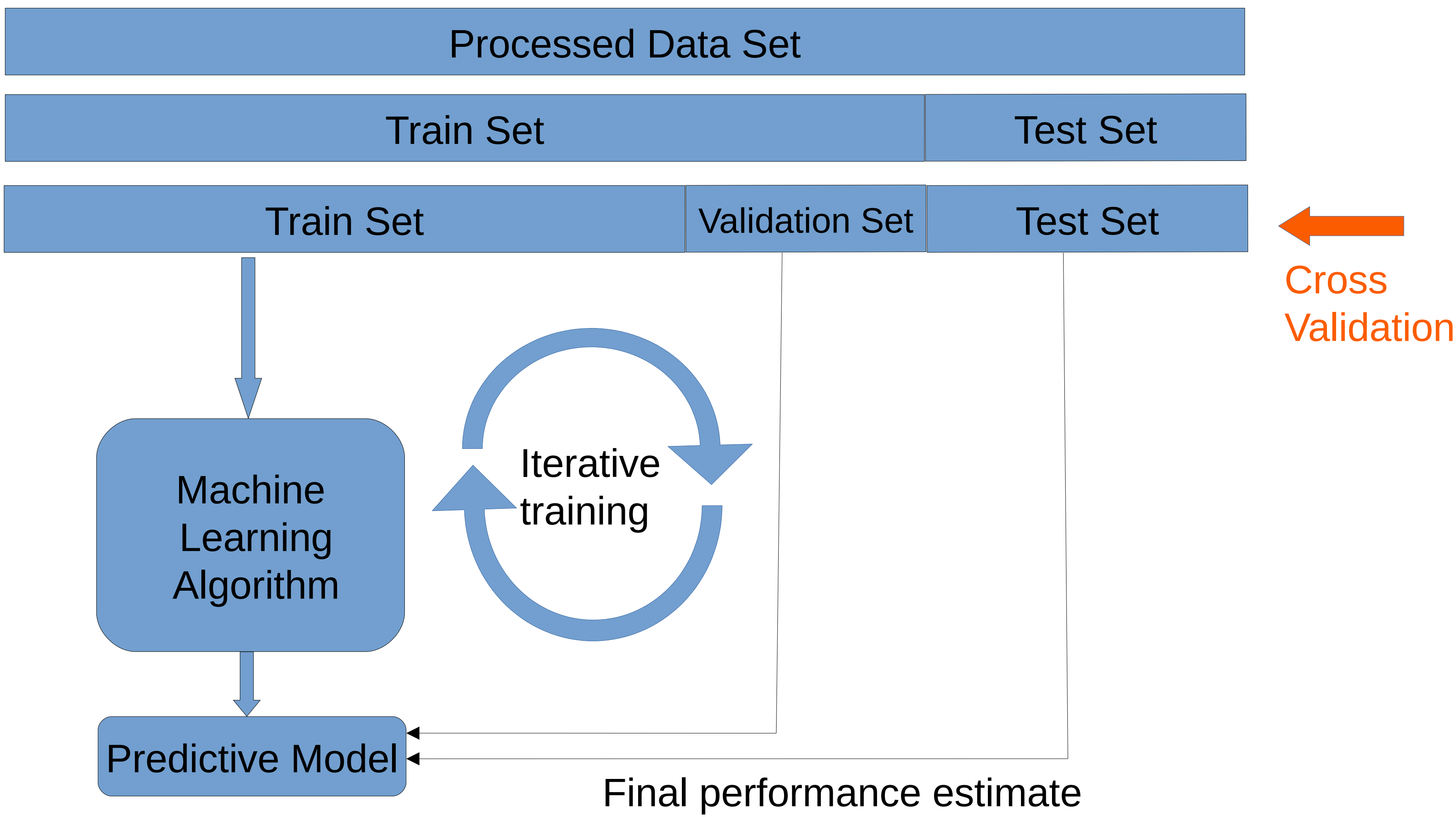
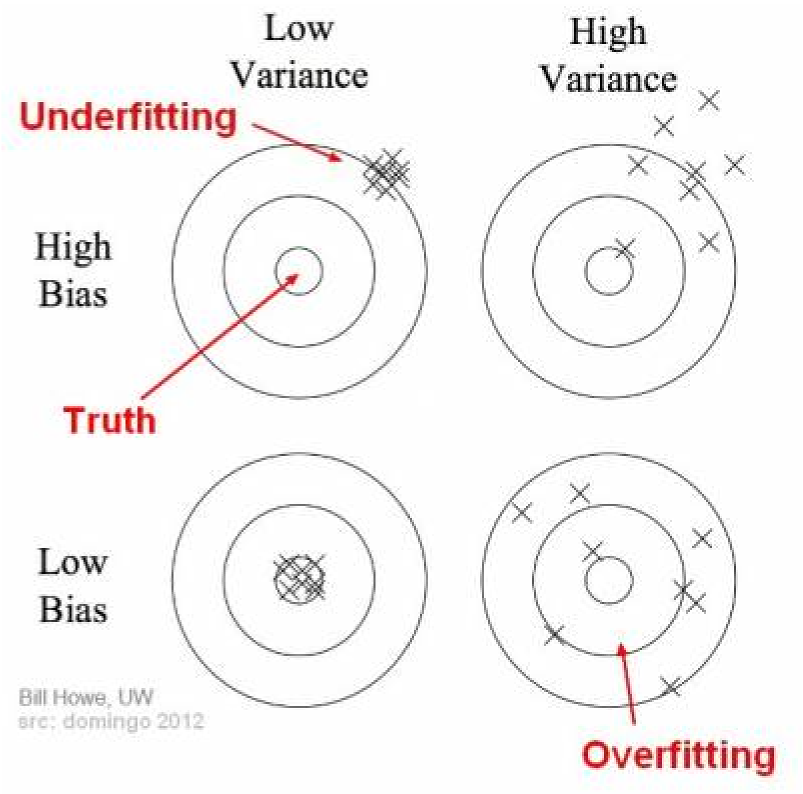


Figure from 'Python Machine Learning' by Sebastian Raschka

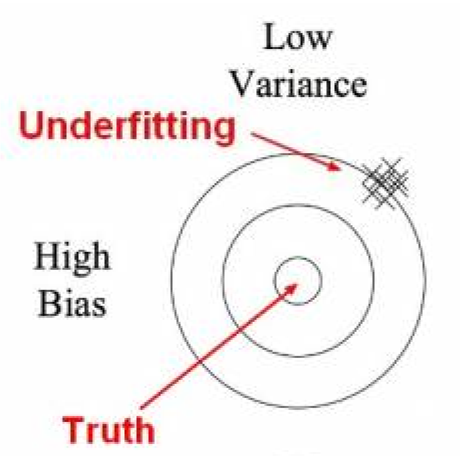
Bias/Variance – Supervised Learning

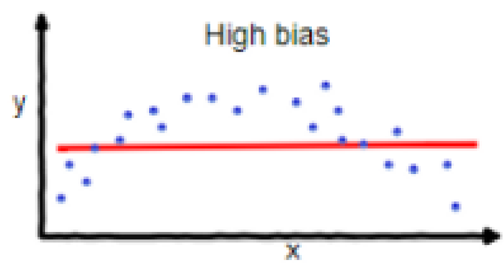
Bias - how close we are to being correct
Variance – how spread out our predictions are



Bias/Variance – Supervised Learning

Underfitting: a model is unable to capture the underlying pattern of the data. Usually have high bias and low variance. This happens if you don't have enough data to build an accurate model or you try to build a linear model with a nonlinear data.

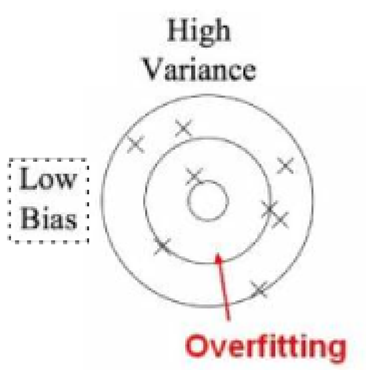


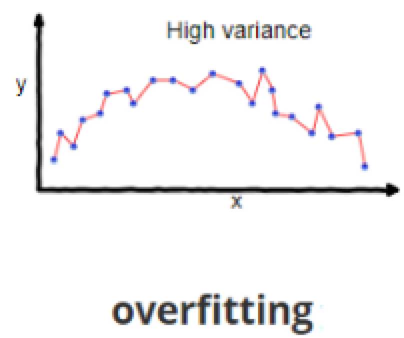


underfitting

Bias/Variance – Supervised Learning

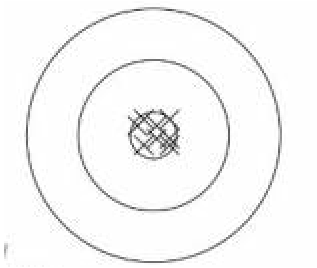
Overfitting: a model captures the noise along with the underlying pattern in the data. Have low bias and high variance and tend to be overly complex. Neural networks and decision trees are prone to this. This happens when you train a model too much or misconfigure hyperparameters (more coming on this). The end result is the model starts to memorize training data.

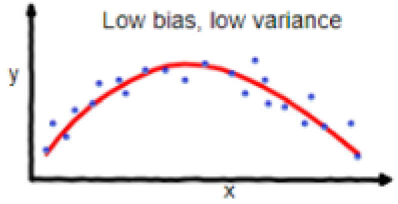




Bias/Variance – Supervised Learning

Well fitted: the model captures a function that describes the overall distribution of data. Or the model learns how a system behaves, and how it responds to given inputs. You can do more than predict with these systems BTW. You can also harness model knowledge to determine effects of varying an input.





A tangent

Example- Concrete mixer model: Say you train a model to determine concrete characteristics based on ingredients (cement, sand, polymers etc).

A tangent

Example- Concrete mixer model: Say you train a model to determine concrete characteristics based on ingredients (cement, sand, polymers etc).

You can use this model as a fast mix tester. That is, instead of physically mixing up a batch to see what happens when you double the sand, you can just double the sand input to the model and see what characteristics are predicted.

A tangent

Example- Concrete mixer model: Say you train a model to determine concrete characteristics based on ingredients (cement, sand, polymers etc).

You can use this model as a fast mix tester. That is, instead of physically mixing up a batch to see what happens when you double the sand, you can just double the sand input to the model and see what characteristics are predicted.

Or … you can write a loop to vary all ingredients, and see which one produces the mix with the traits you want.

A tangent

Example- Concrete mixer model: Say you train a model to determine concrete characteristics based on ingredients (cement, sand, polymers etc).

You can use this model as a fast mix tester. That is, instead of physically mixing up a batch to see what happens when you double the sand, you can just double the sand input to the model and see what characteristics are predicted.

Or … you can write a loop to vary all ingredients, and see which one produces the mix with the traits you want.

You can test hundreds or thousands of combinations in minutes

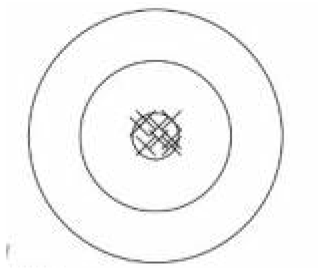
A tangent

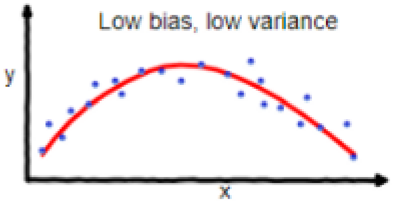
Example- Concrete mixer model: Say you train a model to determine concrete characteristics based on ingredients (cement, sand, polymers etc).
You can use this model as a fast mix tester. That is, instead of physically mixing up a batch to see what happens when you double the sand, you can just double the sand input to the model and see what characteristics are predicted.
Or … you can write a loop to vary all ingredients, and see which one produces the mix with the traits you want.
You can test hundreds or thousands of combinations in minutes

Now imagine applying this same technique to ferret out suitable compounds to treat disease.

Bias/Variance – Supervised Learning

Well fitted: the model captures a function that describes the overall distribution of data. Or the model learns how a system behaves, and how it responds to given inputs. You can do more than predict with these systems BTW. You can also harness it's knowledge to determine what happens when you purposefully vary an input.





How do you find this goldilocks model?

Getting a good fit for Trees - Theoretical

Bias Variance Tradeoff: If model is too simple it may have high bias and low variance. On the other hand if model is too complex then it’s going to have high variance and low bias.

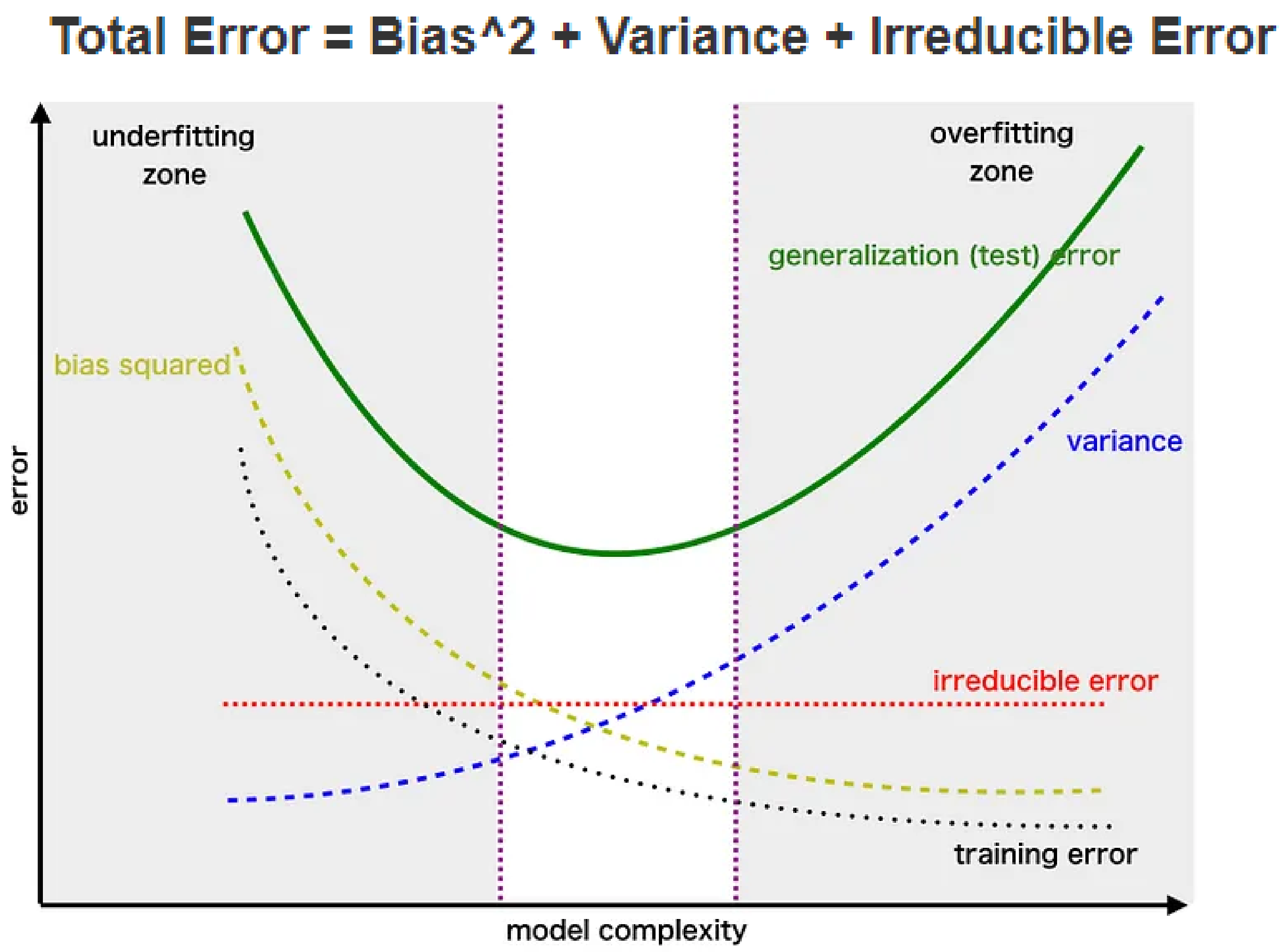
We need to find the right/good balance without overfitting or underfitting the data.

How? By reducing the total error

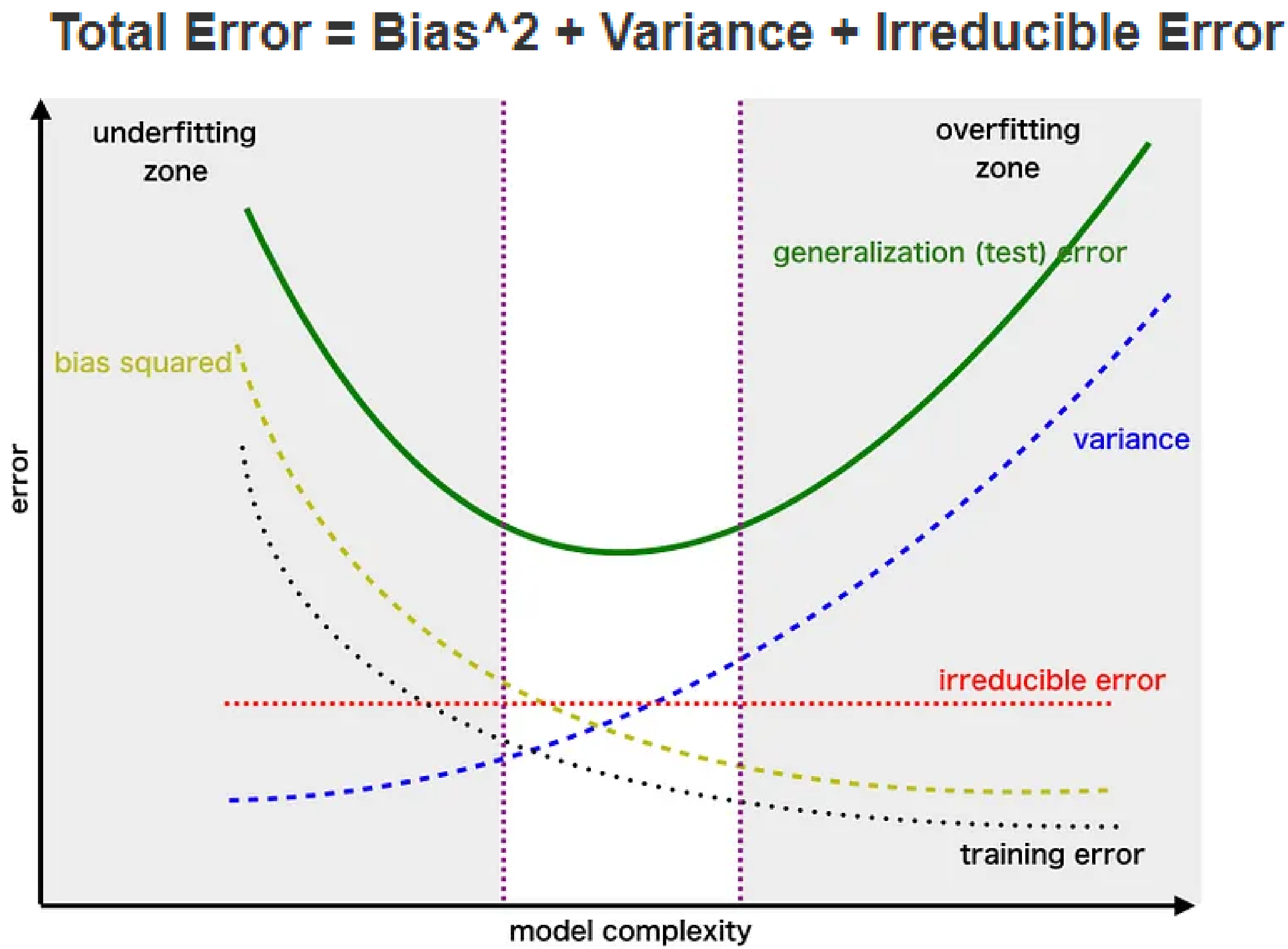
$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

See wikipedia for derivation

Getting a good fit for Trees - Theoretical



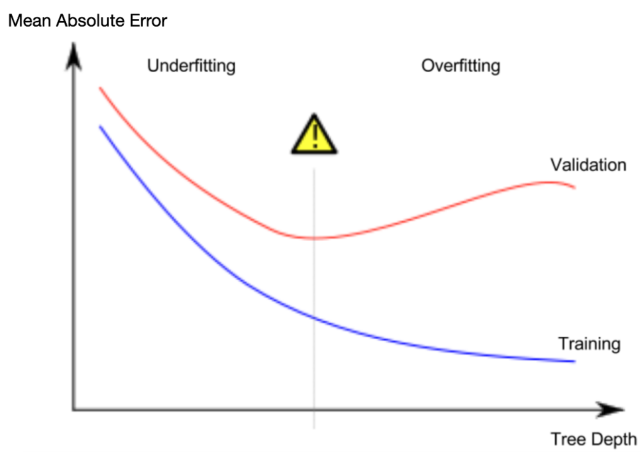
Getting a good fit for Trees - Theoretical



Go to the next slide for a simpler interpretation

Getting a good fit for Trees - Practical

To find a good model <u>for trees</u>, just find the tree hyperparameters that give the lowest Mean Absolute Error (MAE)* on the Validation set
(BTW this procedure is more complex for neural networks)



What tree hyperparameters do you adjust?

* Or whatever quality metric you are optimizing (Accuracy etc..)

Getting a good fit for Trees - Theoretical

Bias Variance Tradeoff: If model is too simple it may have high bias and low variance. On the other hand if model is too complex then it’s going to have high variance and low bias.

We need to find the right/good balance without overfitting or underfitting the data.

How? By reducing the total error

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

See wikipedia for derivation

Or, go to the next slide for a practical way

Which tree parameters to adjust

Decision trees over fit when they fit the training data too well and do not generalize to the validation data. They are too complex. Adjust complexity via estimator parameters:

Which tree parameters to adjust

Decision trees over fit when they fit the training data too well and do not generalize to the validation data. They are too complex. Adjust complexity via estimator parameters:

For instance here are the parameters you can adjust on sklearns RandomForestRegressor

```
forest_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': True,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Which tree parameters to adjust

Decision trees over fit when they fit the training data too well and do not generalize to the validation data. They are too complex. Adjust complexity via estimator parameters:

For instance here are the parameters you can adjust on sklearns RandomForestRegressor

Tree complexity **is not** controlled by:
n_estimators
you can have as many as you want,
does not lead to overfitting

```
forest_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': True,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Which tree parameters to adjust

Decision trees over fit when they fit the training data too well and do not generalize to the validation data. They are too complex. Adjust complexity via estimator parameters:

For instance here are the parameters you can adjust on sklearns RandomForestRegressor

Tree complexity **is not** controlled by:
n_estimators
you can have as many as you want,
does not lead to overfitting

Tree complexity **is** controlled by:
max_depth
min_sample_split
These (and others) **may** lead to overfitting

```
forest_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': True,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Which tree parameters to adjust

Decision trees over fit when they fit the training data too well and do not generalize to the validation data. They are too complex. Adjust complexity via estimator parameters:

For instance here are the parameters you can adjust on sklearns RandomForestRegressor

Tree complexity **is not** controlled by:
n_estimators
you can have as many as you want,
does not lead to overfitting

Tree complexity **is** controlled by:
max_depth
min_sample_split
These (and others) **may** lead to overfitting

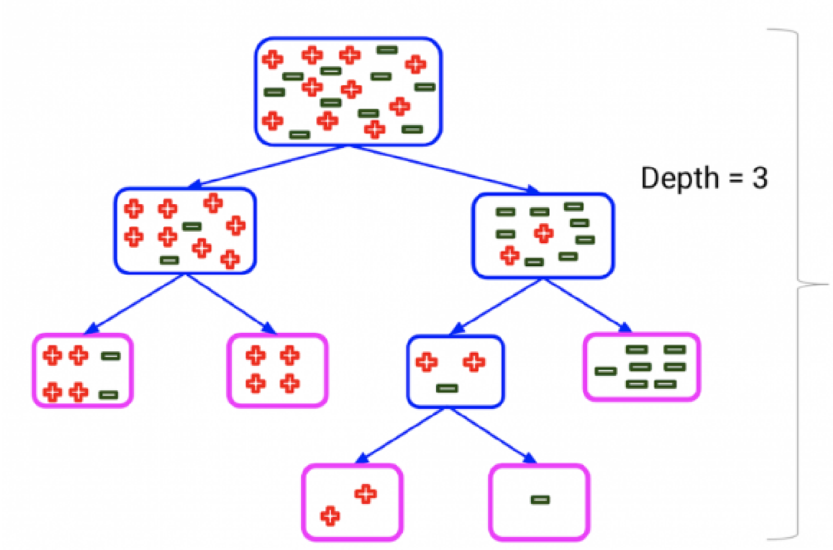
See sklearn documentation for other parameters

```
forest_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': True,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Hyperparameter explanation

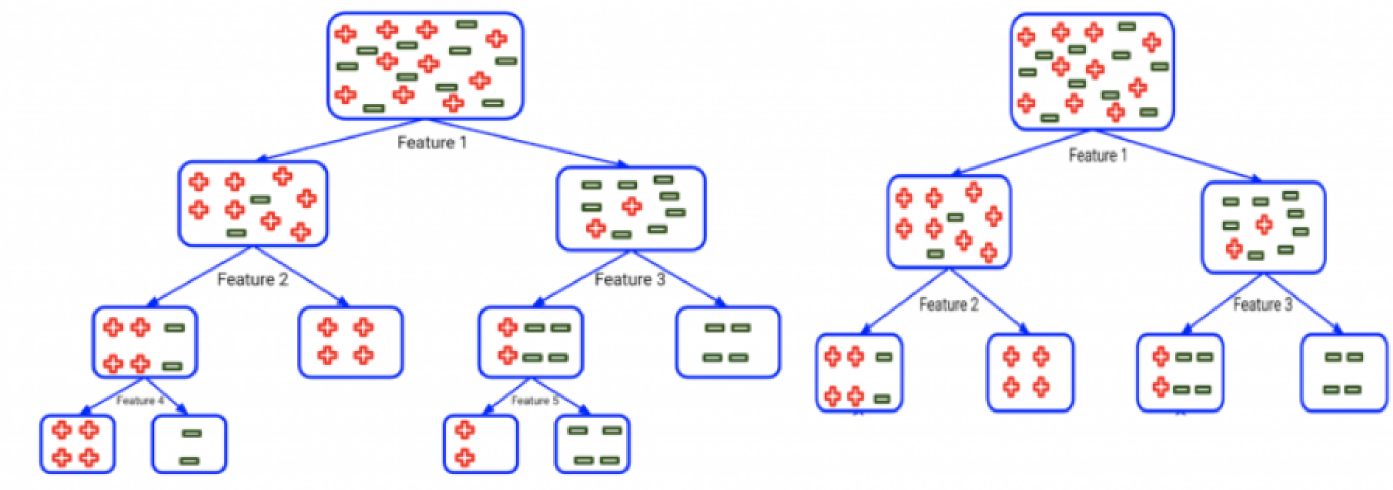
max_depth – how many levels in tree.
The more levels the more complex the tree. Eventually the tree will fit the training set perfectly, but will not generalize to the test or validation set



See https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Hyperparameter explanation

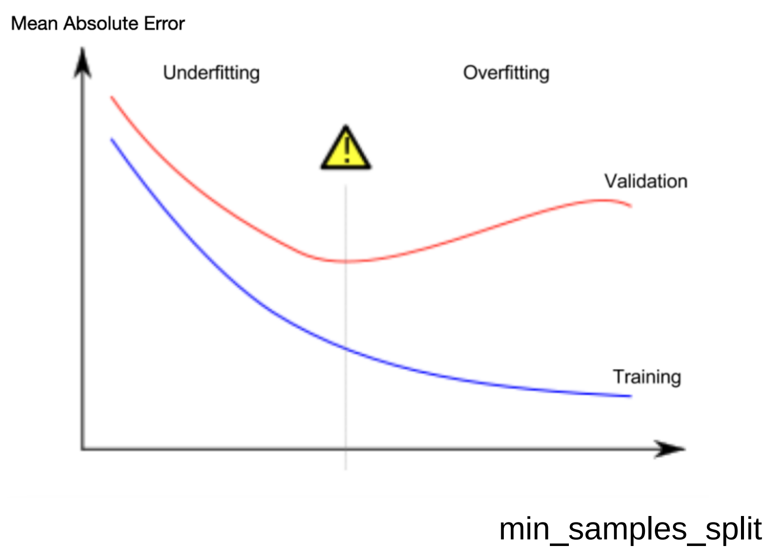
min_sample_split – the minimum required number of observations in any given node in order to split it. default=2.
This means that if any terminal node has more than two observations and is not pure, it can be split into subnodes. This has the effect that a tree keeps splitting until it gets mostly pure nodes, or impure nodes with just 2 members. Increasing this value reduces the number of splits and the tendency to overfit.



See https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Hyperparameter tuning

Sklearn has fine default hyperparameters for it’s models
But you should **ALWAYS** adjust them to make your model fit your dataset better



Using something called cross validation and hyperparameter tuning

Cross Validation

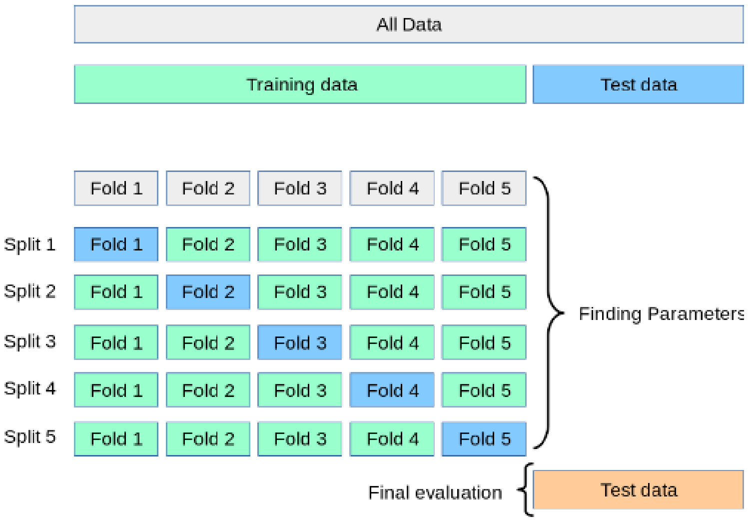
Divide data, into train and test sets, Use kfold cross validation on the train set

```
# Define the model
clf = sklearn.ensemble.RandomForestRegressor()

#get the cross validation score
numb_folds=5
mae=sklearn.model_selection.cross_val_score(clf, train_X, train_y, cv=numb_folds, scoring='neg_mean_absolute_error').mean()
print(f'The mean absolute error={mae}')
```

What does this do?

1. Divides train_X,train_y into 5 chunks
2. trains 5 models, 1 for each split
   each model uses the blue fold for validation and all other green folds for training
3. When done, takes average of all 5 models to give average mae.



Note: It just generates an estimate of model performance.

Cross Validation

Divide data, into train and test sets, Use kfold cross validation on the train set

```
# Define the model
clf = sklearn.ensemble.RandomForestRegressor()

#get the cross validation score
numb_folds=5
mae=sklearn.model_selection.cross_val_score(clf, train_X, train_y, cv=numb_folds, scoring='neg_mean_absolute_error').mean()
print(f'The mean absolute error={mae}')
```

Advantage:

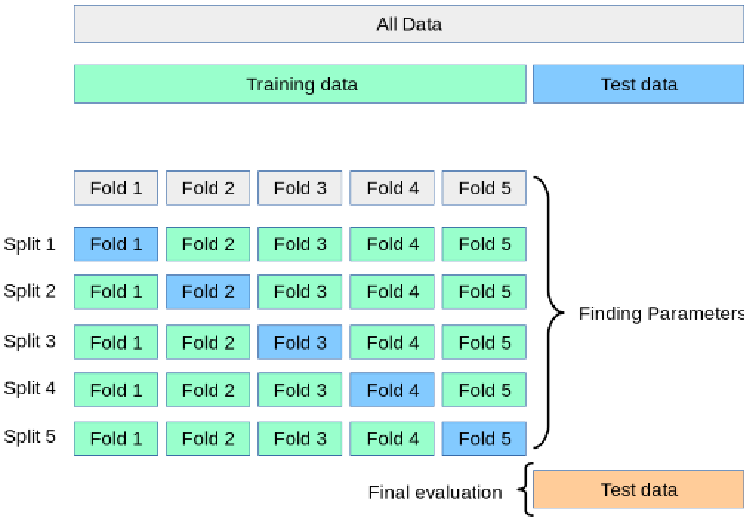
Generates an estimate for models best performance

Model trains on all training data

Disadvantages:

Takes k times as long

Does not produce a unified model (although you could use all models as an ensemble and take the average of their estimates)



Cross Validation

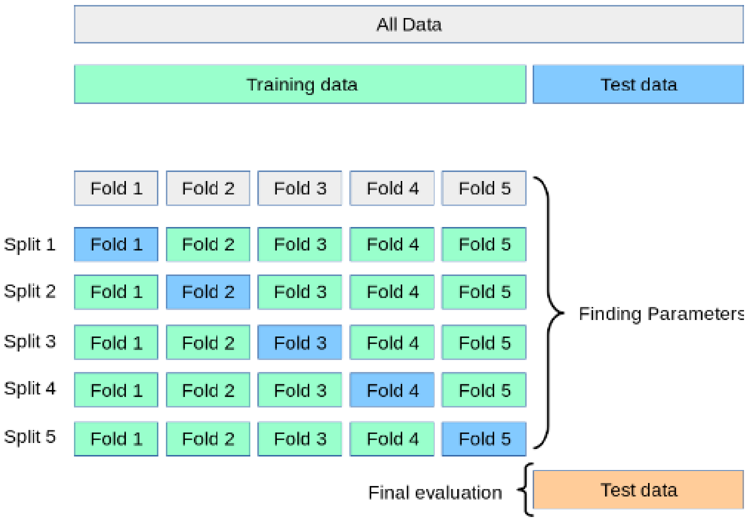
Divide data, into train and test sets, Use kfold cross validation on the train set

```
# Define the model
clf = sklearn.ensemble.RandomForestRegressor()

#get the cross validation score
numb_folds=5
mae=sklearn.model_selection.cross_val_score(clf, train_X, train_y, cv=numb_folds, scoring='neg_mean_absolute_error').mean()
print(f'The mean absolute error={mae}')
```

Advantage:
Generates an estimate for models best performance
Model trains on all training data

Disadvantages:
Takes k times as long
Does not produce a unified model (although you could use all models as an ensemble and take the average of their estimates)



Why use it?
Because you can run cross validation in a loop with different hyperparameters, keeping track of scores.
When done, choose hyperparameter combination that returns the best score

Finally, hyperparameter optimization

We will use Optuna.
It's fast, flexible and very efficient.
(Plus it's well respected and widely used by the cut throat data science competitors at Kaggle)

See notebook for code walk through.

Summary

Bias variance, Under and overfitting
How to get a good fit: Adjust hyperparameters until validation score starts getting worse
Cross Validation : A way to use all your data for training to get a best case estimate for model performance given a set of hyperparameters
Hyperparameter tuning using Optuna: An efficient way to explore a hyperparameter space for the best combination of hyperparmeters