Pipeline

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

Leakage?

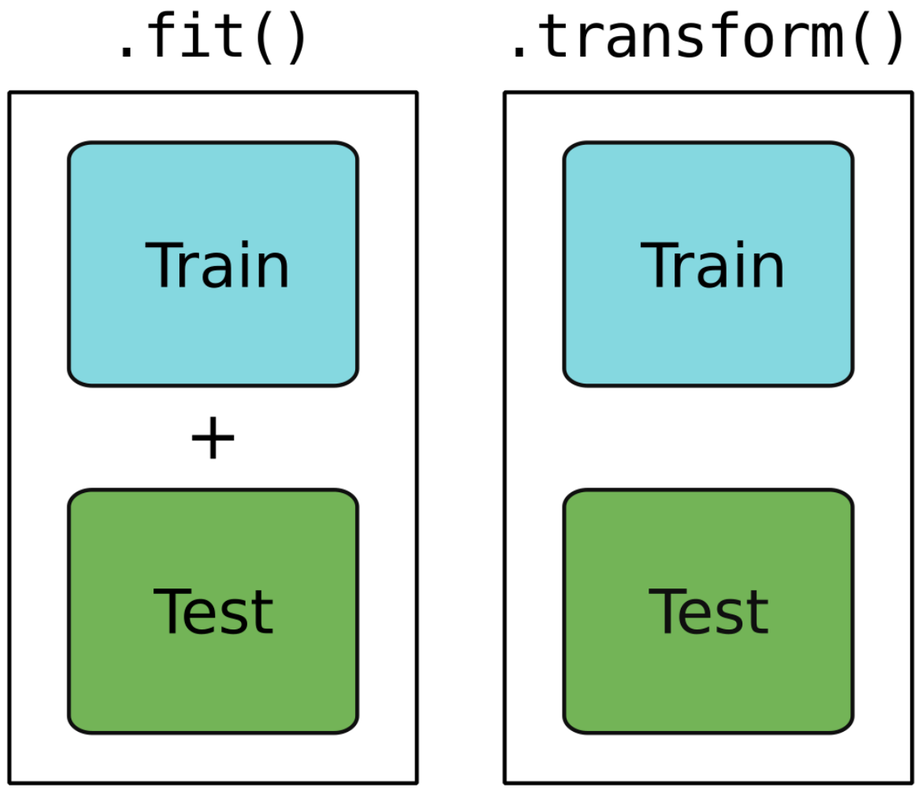
`.fit()`

Only for training data.

`.transform()`

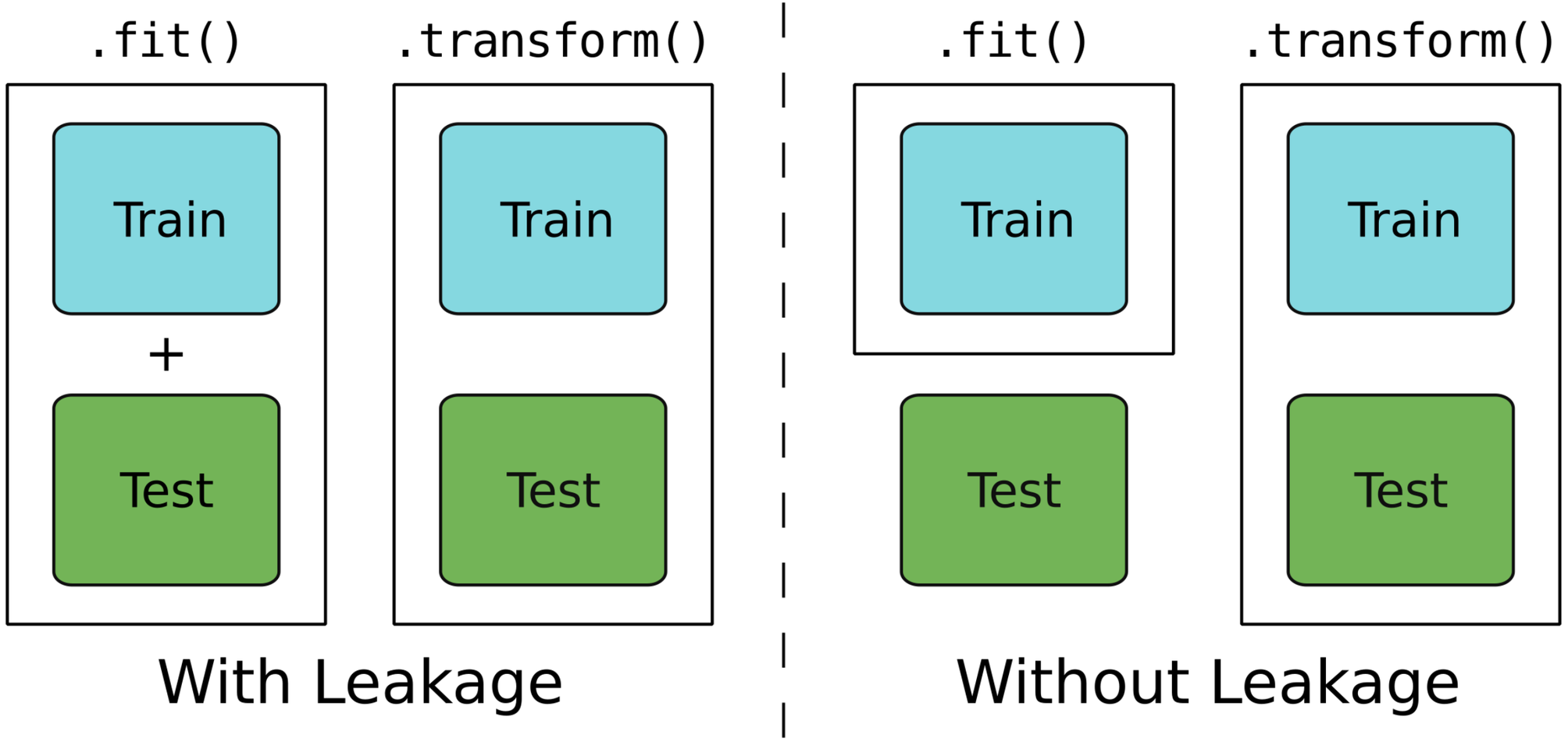
For testing and training data.

A leaky model



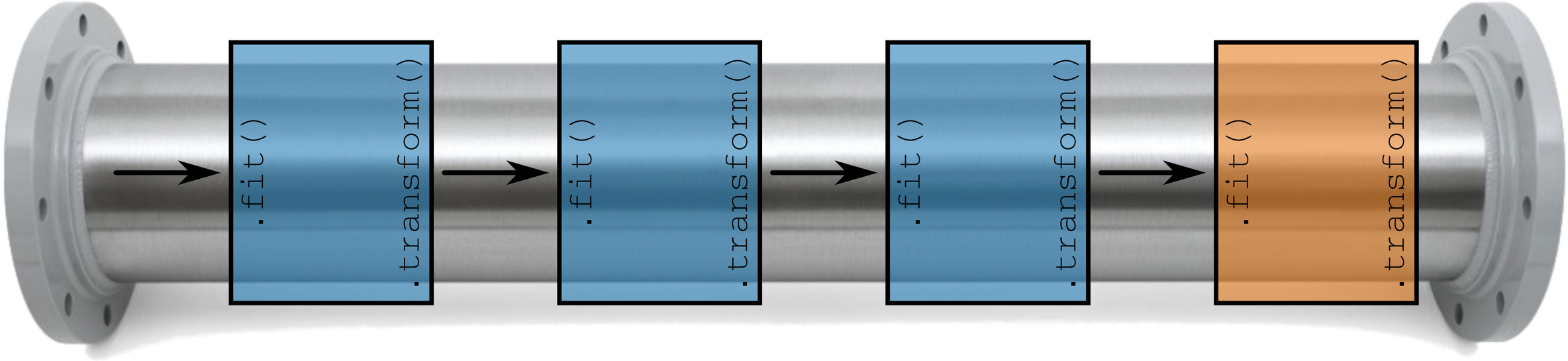
With Leakage

A watertight model



Pipeline

A pipeline consists of a series of operations.



You could apply each operation individually... or you could just apply the pipeline!

Cars model: Steps

```
indexer = StringIndexer(inputCol='type', outputCol='type_idx')
onehot = OneHotEncoder(inputCols=['type_idx'], outputCols=['type_dummy'])
assemble = VectorAssembler(
  inputCols=['mass', 'cyl', 'type_dummy'],
  outputCol='features'
)
regression = LinearRegression(labelCol='consumption')
```

Cars model: Applying steps

Training data

```
indexer = indexer.fit(cars_train)
cars_train = indexer.transform(cars_train)
```

```
onehot = onehot.fit(cars_train)
cars_train = onehot.transform(cars_train)
```

```
cars_train = assemble.transform(cars_train)
```

```
# Fit model to training data
regression = regression.fit(cars_train)
```

Testing data

```
#
cars_test  = indexer.transform(cars_test)
```

```
#
cars_test  = onehot.transform(cars_test)
```

```
cars_test  = assemble.transform(cars_test)
```

```
# Make predictions on testing data
predictions = regression.transform(cars_test)
```

Cars model: Pipeline

Combine steps into a pipeline.

```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[indexer, onehot, assemble, regression])
```

Training data

```
pipeline = pipeline.fit(cars_train)
```

Testing data

```
predictions = pipeline.transform(cars_test)
```

Cars model: Stages

Access individual stages using the `.stages` attribute.

```
# The LinearRegression object (fourth stage -> index 3)
pipeline.stages[3]

print(pipeline.stages[3].intercept)
```

```
4.19433571782916
```

```
print(pipeline.stages[3].coefficients)
```

```
DenseVector([0.0028, 0.2705, -1.1813, -1.3696, -1.1751, -1.1553, -1.8894])
```

Pipelines streamline workflow!

MACHINE LEARNING WITH PYSPARK

Cross-Validation

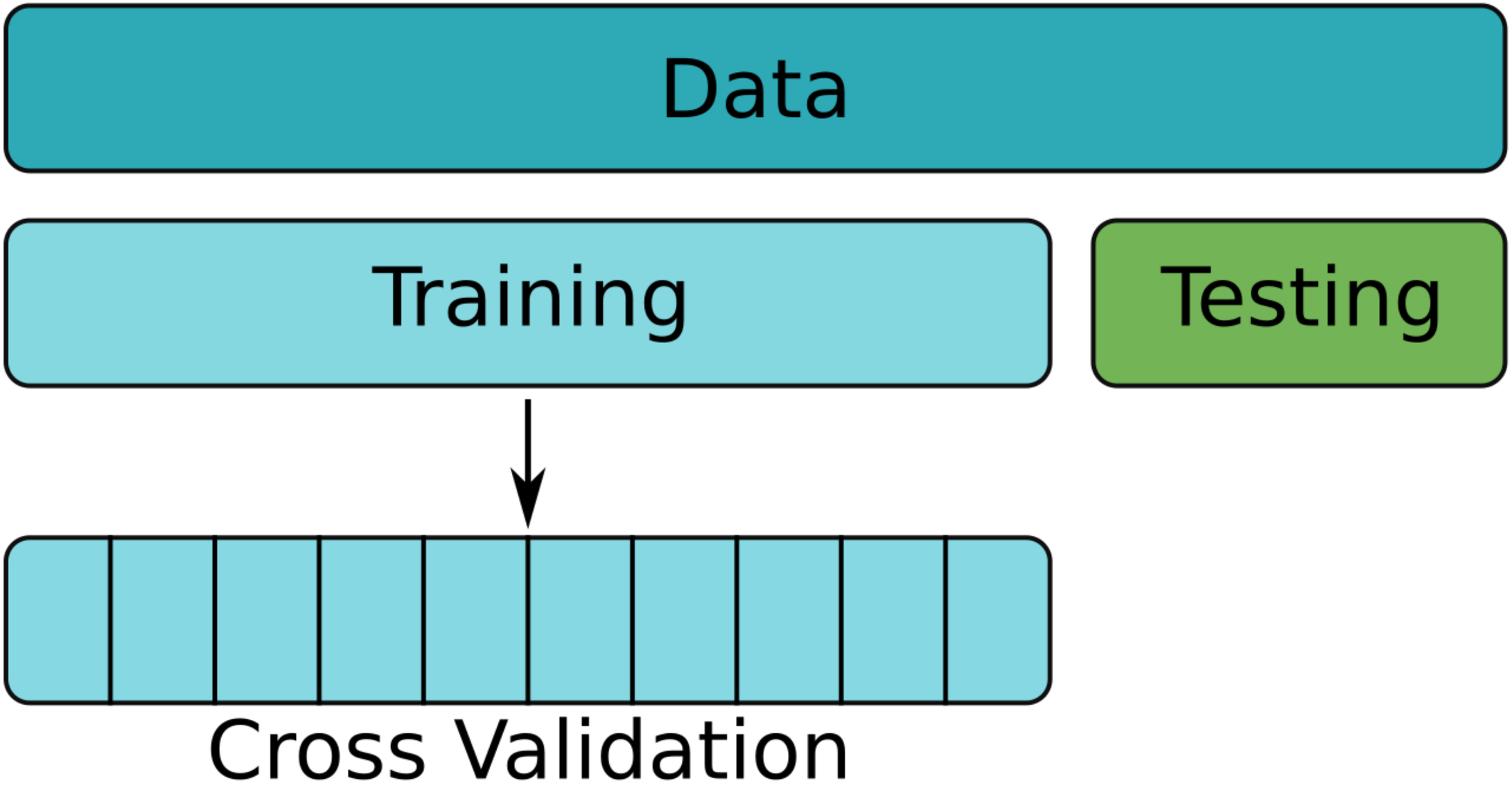
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

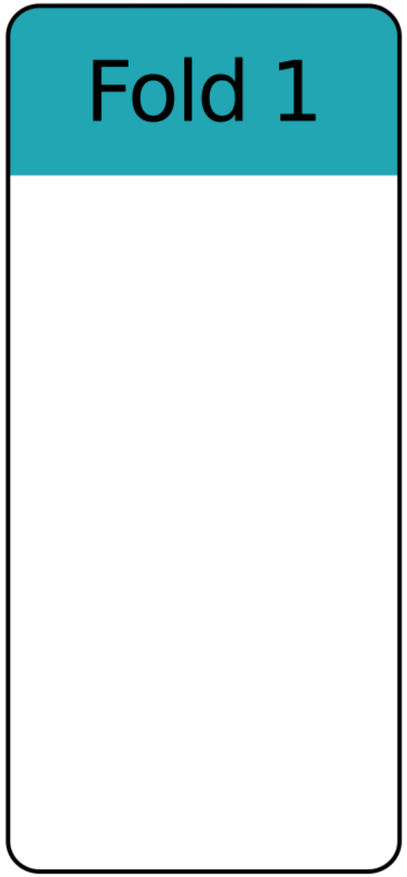
datacamp

Data

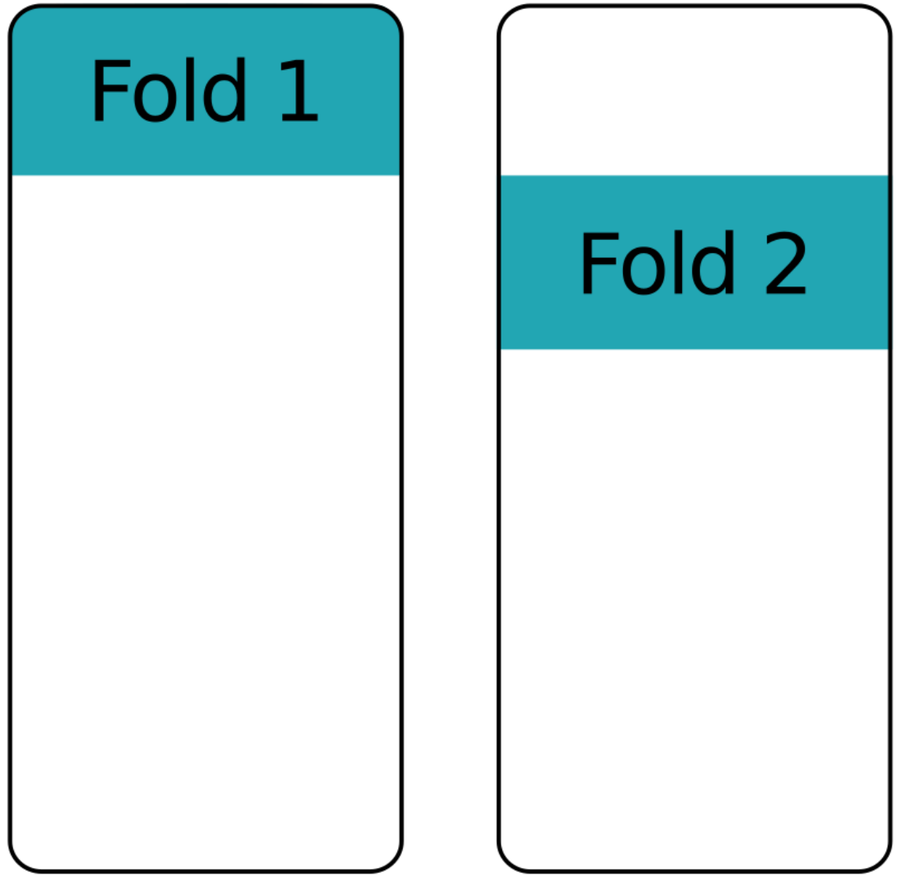

Data
Training
Testing


Data
Training
Testing
Cross Validation

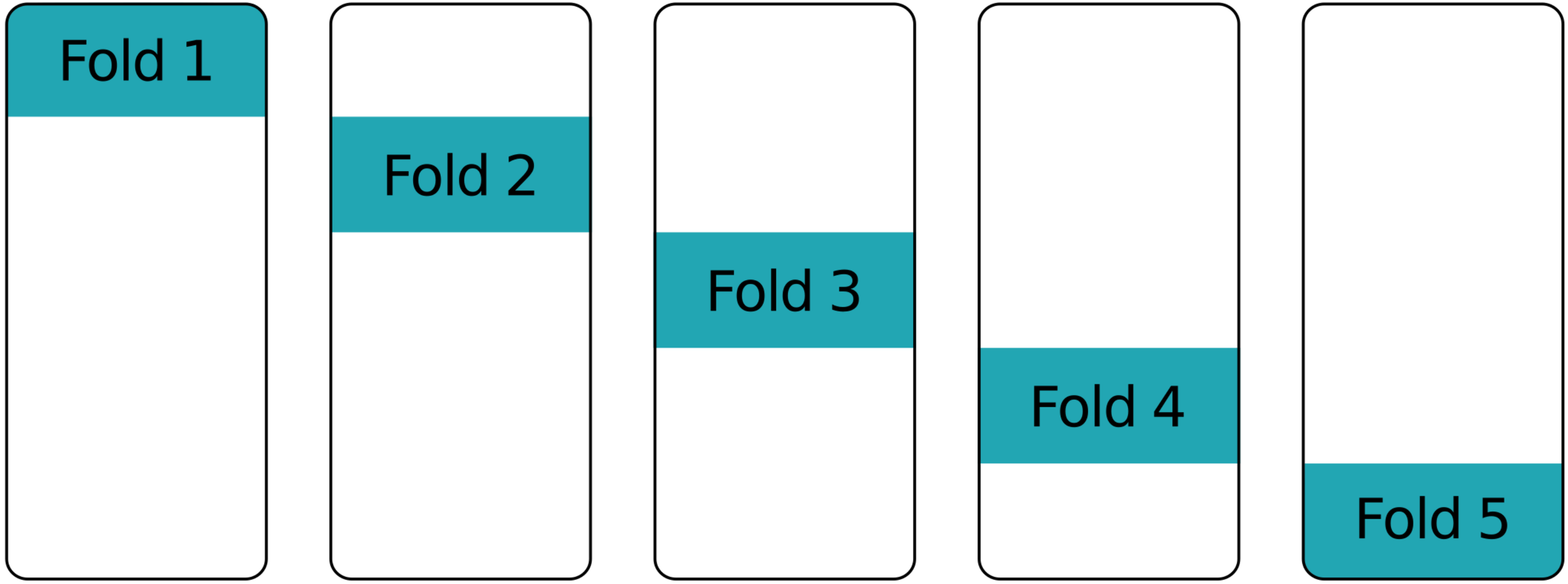
Fold upon fold - first fold



Fold upon fold - second fold



Fold upon fold - other folds



Cars revisited

```
cars.select('mass', 'cyl', 'consumption').show(5)
```

```
+------+---+-----------+
|  mass|cyl|consumption|
+------+---+-----------+
|1451.0|  6|       9.05|
|1129.0|  4|       6.53|
|1399.0|  4|       7.84|
|1147.0|  4|       7.84|
|1111.0|  4|       9.05|
+------+---+-----------+
```

Estimator and evaluator

An object to build the model. This can be a pipeline.

```
regression = LinearRegression(labelCol='consumption')
```

An object to evaluate model performance.

```
evaluator = RegressionEvaluator(labelCol='consumption')
```

Grid and cross-validator

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

A grid of parameter values (empty for the moment).

```
params = ParamGridBuilder().build()
```

The cross-validation object.

```
cv = CrossValidator(estimator=regression,
                    estimatorParamMaps=params,
                    evaluator=evaluator,
                    numFolds=10, seed=13)
```

Cross-validators need training too

Apply cross-validation to the training data.

```
cv = cv.fit(cars_train)
```

What's the average RMSE across the folds?

```
cv.avgMetrics
```

```
[0.8006637221515 72]
```

Cross-validators act like models

Make predictions on the original testing data.

```
evaluator.evaluate(cv.transform(cars_test))
```

```
# RMSE on testing data
0.745974203928479
```

Much smaller than the cross-validated RMSE.

```
# RMSE from cross-validation
0.800663722151572
```

A simple train-test split would have given an overly optimistic view on model performance.

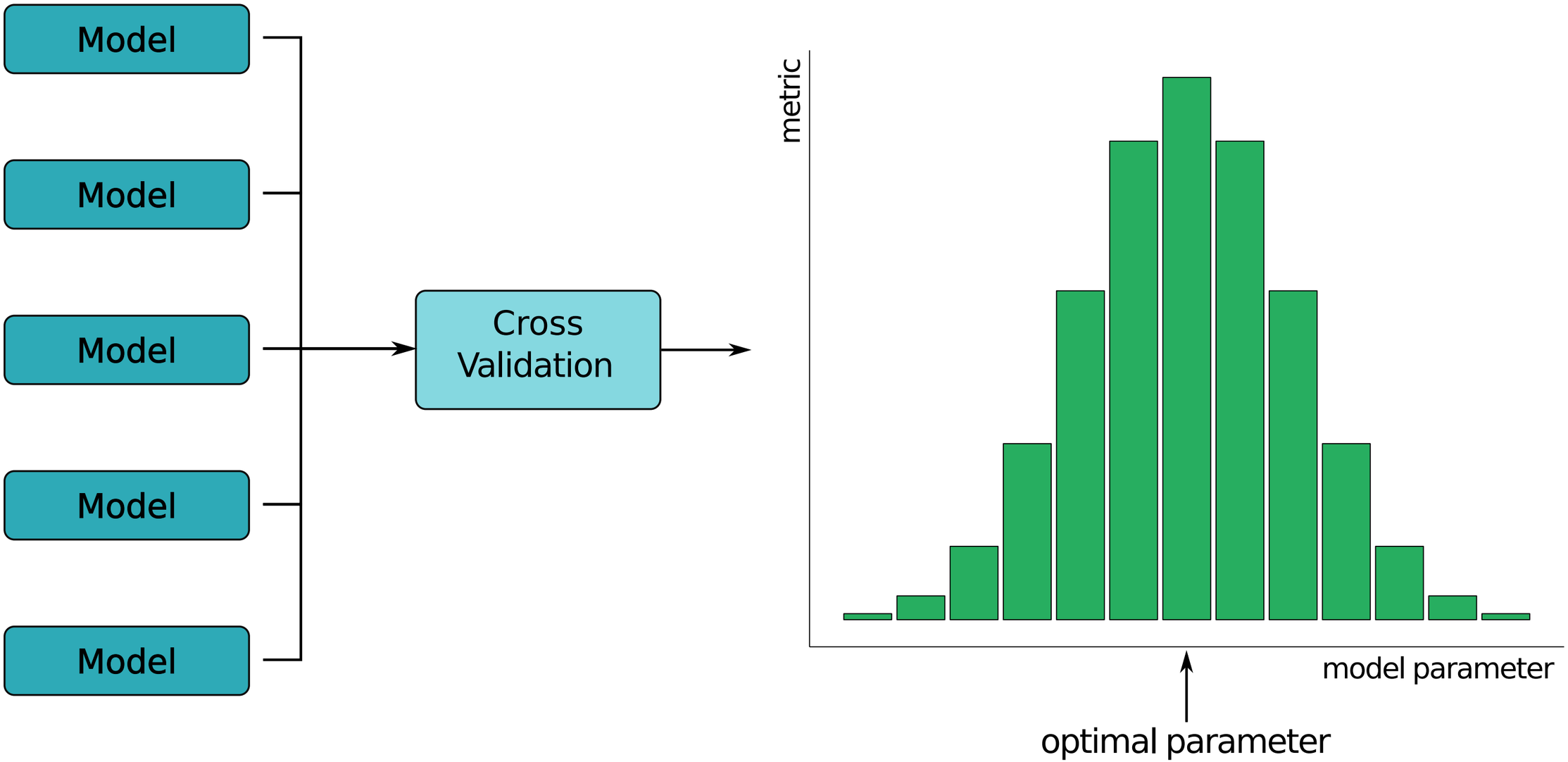
Cross-validate all the models!

MACHINE LEARNING WITH PYSPARK

Grid Search

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics


Model
Model
Model
Model
Model
Cross Validation
metric
model parameter
optimal parameter

Cars revisited (again)

```
cars.select('mass', 'cyl', 'consumption').show(5)
```

```
+------+---+-----------+
|  mass|cyl|consumption|
+------+---+-----------+
|1451.0|  6|       9.05|
|1129.0|  4|       6.53|
|1399.0|  4|       7.84|
|1147.0|  4|       7.84|
|1111.0|  4|       9.05|
+------+---+-----------+
```

Fuel consumption with intercept

Linear regression *with* an intercept. Fit to training data.

```
regression = LinearRegression(labelCol='consumption', fitIntercept=True)
regression = regression.fit(cars_train)
```

Calculate the RMSE on the testing data.

```
evaluator.evaluate(regression.transform(cars_test))
```

```
# RMSE for model with an intercept
0.745974203928479
```

Fuel consumption without intercept

Linear regression *without* an intercept. Fit to training data.

```
regression = LinearRegression(labelCol='consumption', fitIntercept=False)
regression = regression.fit(cars_train)
```

Calculate the RMSE on the testing data.

```
# RMSE for model without an intercept (second model)
0.852819012439
```

```
# RMSE for model with an intercept    (first model)
0.745974203928
```

Parameter grid

```
from pyspark.ml.tuning import ParamGridBuilder

# Create a parameter grid builder
params = ParamGridBuilder()
# Add grid points
params = params.addGrid(regression.fitIntercept, [True, False])
# Construct the grid
params = params.build()

# How many models?
print('Number of models to be tested: ', len(params))
```

```
Number of models to be tested:  2
```

Grid search with cross-validation

Create a cross-validator and fit to the training data.

```
cv = CrossValidator(estimator=regression,
                    estimatorParamMaps=params,
                    evaluator=evaluator)
cv = cv.setNumFolds(10).setSeed(13).fit(cars_train)
```

What's the cross-validated RMSE for each model?

```
cv.avgMetrics
```

```
[0.800663722151, 0.907977823182]
```

The best model & parameters

```
# Access the best model
cv.bestModel
```

Or just use the cross-validator object.

```
predictions = cv.transform(cars_test)
```

Retrieve the best parameter.

```
cv.bestModel.explainParam('fitIntercept')
```

```
'fitIntercept: whether to fit an intercept term (default: True, current: True)'
```

A more complicated grid

```
params = ParamGridBuilder() \
            .addGrid(regression.fitIntercept, [True, False]) \
            .addGrid(regression.regParam, [0.001, 0.01, 0.1, 1, 10]) \
            .addGrid(regression.elasticNetParam, [0, 0.25, 0.5, 0.75, 1]) \
            .build()
```

How many models now?

```
print ('Number of models to be tested: ', len(params))
```

```
Number of models to be tested:  50
```

Find the best parameters!

MACHINE LEARNING WITH PYSPARK

Ensemble

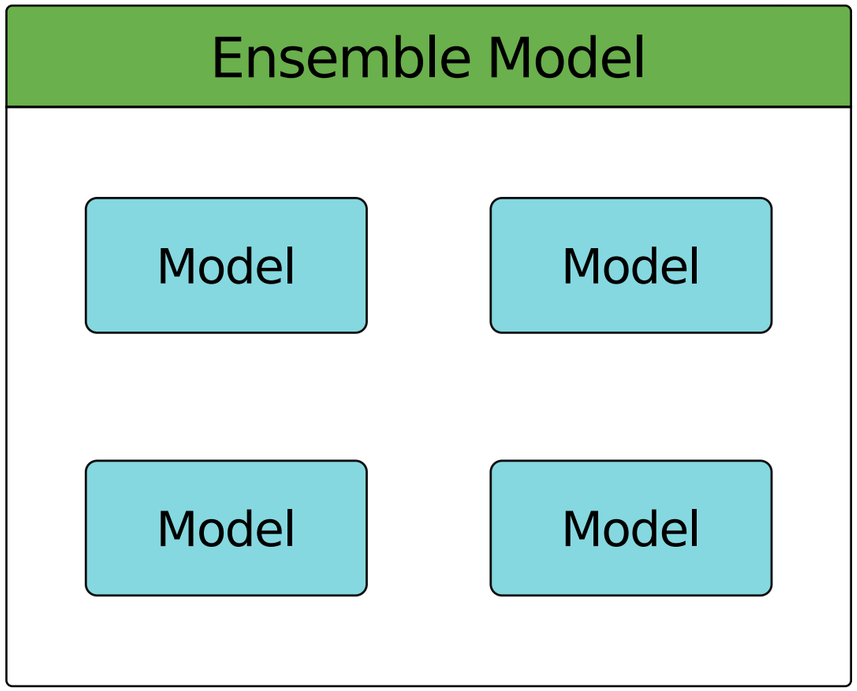
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

datacamp

What's an ensemble?

It's a collection of models.



Wisdom of the Crowd — collective opinion of a group better than that of a single expert.

Ensemble diversity

> **Diversity** and **independence** are important because the best collective decisions are the product of disagreement and contest, not consensus or compromise.
>
> ? James Surowiecki, *The Wisdom of Crowds*

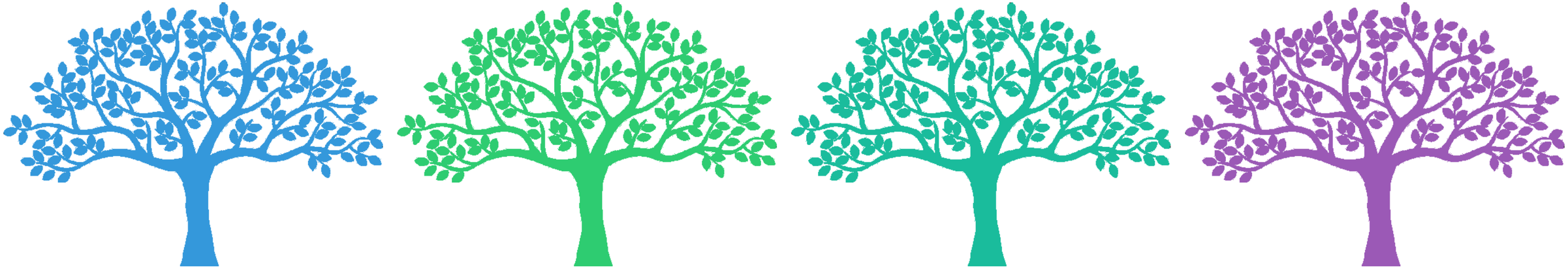
Random Forest

Random Forest — an ensemble of Decision Trees

Creating model diversity:

- each tree trained on *random subset* of data
- *random subset* of features used for splitting at each node

No two trees in the forest should be the same.



Create a forest of trees

Returning to cars data: manufactured in USA (`0.0`) or not (`1.0`).

Create Random Forest classifier.

```
from pyspark.ml.classification import RandomForestClassifier

forest = RandomForestClassifier(numTrees=5)
```

Fit to the training data.

```
forest = forest.fit(cars_train)
```

Seeing the trees

How to access trees within forest?

```
forest.trees
```

```
[DecisionTreeClassificationModel (uid=dtc_aa66702a4ce9) of depth 5 with 17 nodes,
 DecisionTreeClassificationModel (uid=dtc_99f7efedafe9) of depth 5 with 31 nodes,
 DecisionTreeClassificationModel (uid=dtc_9306e4a5fa1d) of depth 5 with 21 nodes,
 DecisionTreeClassificationModel (uid=dtc_d643bd48a8dd) of depth 5 with 23 nodes,
 DecisionTreeClassificationModel (uid=dtc_a2d5abd67969) of depth 5 with 27 nodes]
```

These can each be used to make individual predictions.

Predictions from individual trees

What predictions are generated by each tree?

```
+------+------+------+------+------+-----+
|tree 0|tree 1|tree 2|tree 3|tree 4|label|
+------+------+------+------+------+-----+
|   0.0|   0.0|   0.0|   0.0|   0.0|  0.0| <- perfect agreement
|   1.0|   1.0|   0.0|   1.0|   0.0|  0.0|
|   0.0|   0.0|   0.0|   1.0|   1.0|  1.0|
|   0.0|   0.0|   0.0|   1.0|   0.0|  0.0|
|   0.0|   1.0|   1.0|   1.0|   0.0|  1.0|
|   1.0|   1.0|   0.0|   1.0|   1.0|  1.0|
|   1.0|   1.0|   1.0|   1.0|   1.0|  1.0| <- perfect agreement
+------+------+------+------+------+-----+
```

Consensus predictions

Use the `.transform()` method to generate consensus predictions.

```
+-----+----------------------------------------+----------+
|label|probability                             |prediction|
+-----+----------------------------------------+----------+
|0.0  |[0.8,0.2]                               |0.0       |
|0.0  |[0.4,0.6]                               |1.0       |
|1.0  |[0.5333333333333333,0.4666666666666666] |0.0       |
|0.0  |[0.7177777777777778,0.2822222222222226]|0.0       |
|1.0  |[0.39396825396825397,0.606031746031746] |1.0       |
|1.0  |[0.17660818713450294,0.823391812865497] |1.0       |
|1.0  |[0.05396825396825396,0.946031746031746]|1.0       |
+-----+----------------------------------------+----------+
```

Feature importances

The model uses these features: `cyl`, `size`, `mass`, `length`, `rpm` and `consumption`.

Which of these is most or least important?

```
forest.featureImportances
```

```
SparseVector(6, {0: 0.0205, 1: 0.2701, 2: 0.108, 3: 0.1895, 4: 0.2939, 5: 0.1181})
```

Looks like:

- `rpm` is most important
- `cyl` is least important.

Gradient-Boosted Trees

Iterative boosting algorithm:

1. Build a Decision Tree and add to ensemble.
2. Predict label for each training instance using ensemble.
3. Compare predictions with known labels.
4. Emphasize training instances with incorrect predictions.
5. Return to 1.

Model improves on each iteration.

Boosting trees

Create a Gradient-Boosted Tree classifier.

```
from pyspark.ml.classification import GBTClassifier

gbt = GBTClassifier(maxIter=10)
```

Fit to the training data.

```
gbt = gbt.fit(cars_train)
```

Comparing trees

Let's compare the three types of tree models on testing data.

```
# AUC for Decision Tree
0.5875

# AUC for Random Forest
0.65

# AUC for Gradient-Boosted Tree
0.65
```

Both of the ensemble methods perform better than a plain Decision Tree.

Ensemble all of the models!

MACHINE LEARNING WITH PYSPARK

Closing thoughts

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

Things you've learned

- Load & prepare data
- Classifiers
  - Decision Tree
  - Logistic Regression
- Regression
  - Linear Regression
  - Penalized Regression
- Pipelines
- Cross-validation & grid search
- Ensembles

Learning more

Documentation at https://spark.apache.org/docs/latest/.

Congratulations!

MACHINE LEARNING WITH PYSPARK