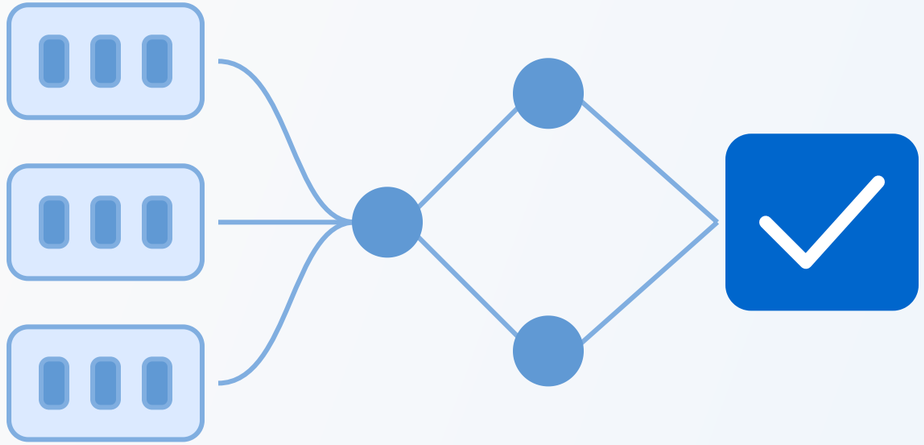
Spark ML

From customer records to reliable predictions



CONTENTS

Topic overview

Follow a customer-churn problem from the prediction target to a model whose results can be evaluated fairly.

Topic	Content
1. Define the prediction	Tasks, prediction unit and the churn example
2. Split the data	Training, validation, test sets and leakage
3. Represent the features	Encoding, vectors, scaling and text
4. Fit and apply the model	Pipelines, logistic regression and predictions
5. Evaluate and select	Confusion matrix, precision, recall, AUC and tuning
6. Apply and review	Lab 09, reliable predictions and a worked final case

Customer churn

Churn means a customer leaves a service. A binary classifier predicts whether a customer belongs to the churn or no-churn class.

We use small invented records to trace the workflow; they are teaching examples, not measured lab results.

Available information

Tenure (months)

Charges

Contract



Model

Learn from past cases

Score another customer



Decision

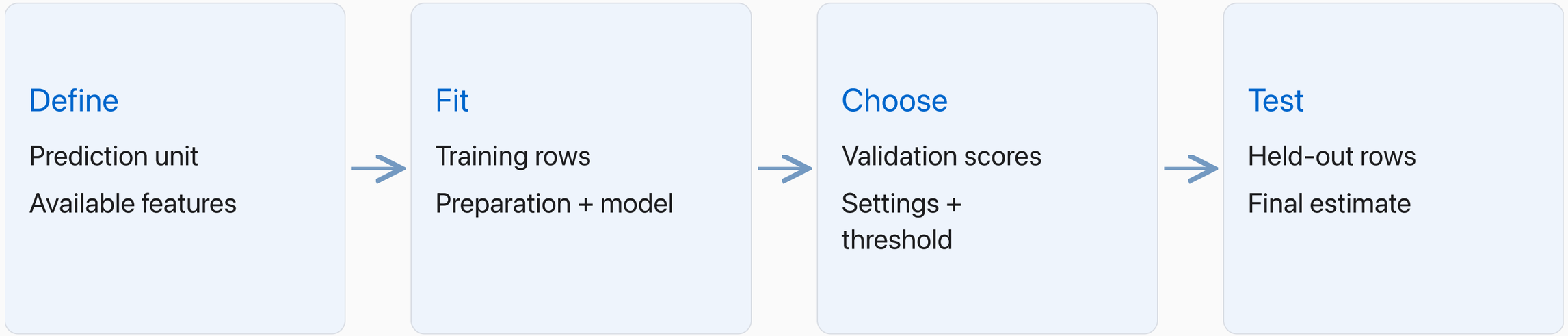
Estimate churn risk

Choose an intervention

Prediction workflow

Supervised learning uses examples with known outcomes to learn a relationship between features and a label.

Preparation and model fitting use training rows; validation guides choices; the test set checks the final choice.



Learning tasks

Classification predicts categories, regression predicts numbers, and clustering groups similar records without known target labels.

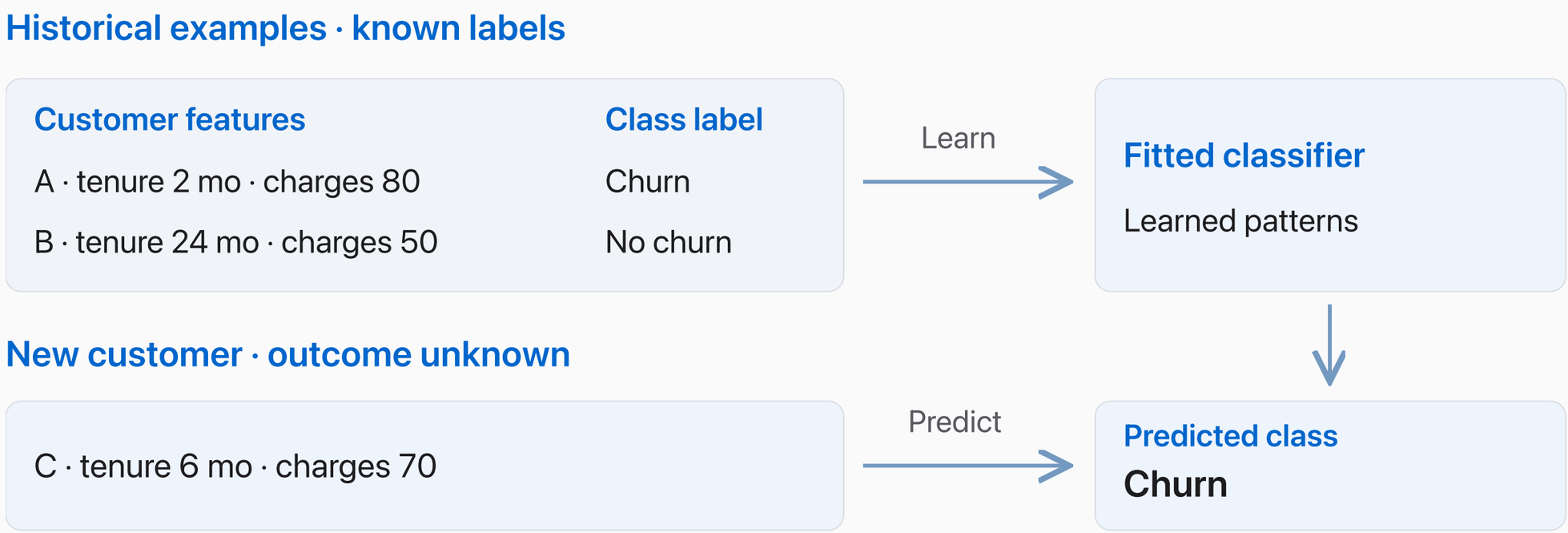
Choose the task from the desired output before choosing an algorithm.

Task	Example question	Output
Classification	Will this customer churn?	Churn / no churn
Regression	What will next month’s spend be?	A numeric amount
Clustering	Which customers look similar?	A group assignment

Classification

Classification learns from examples with known class labels to predict a category for a new example. Binary classification has two classes; multiclass classification has more than two.

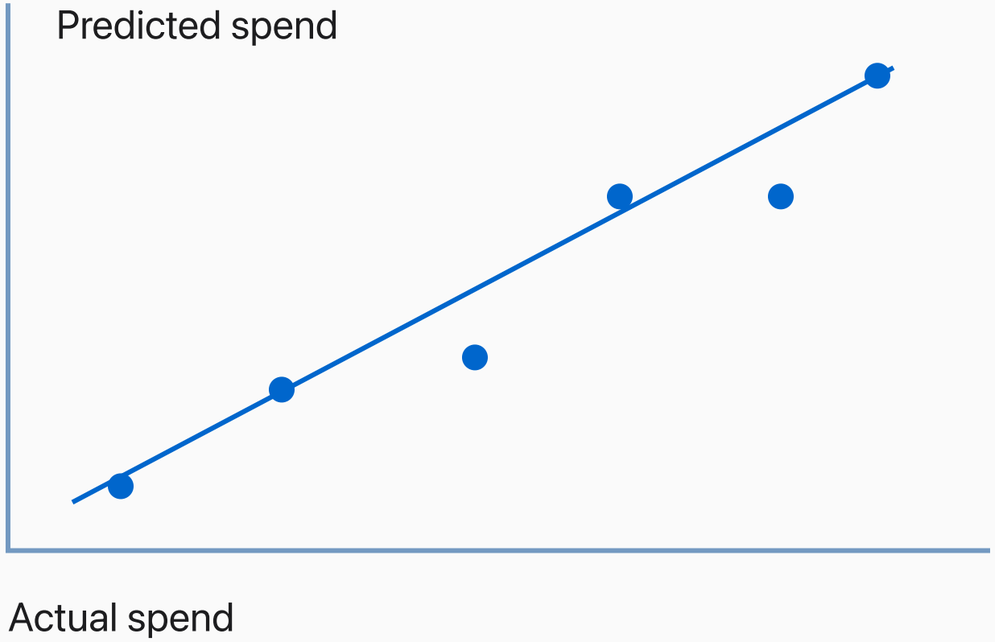
For customer churn, the inputs are customer features and the predicted class is churn or no churn—not a spending amount.



Regression

A regression model predicts a numeric quantity. Prediction error is the difference between the predicted and observed values.

For an actual spend of 80 and a prediction of 65, the error is −15 and the absolute error is 15.



One prediction

Actual spend: 80

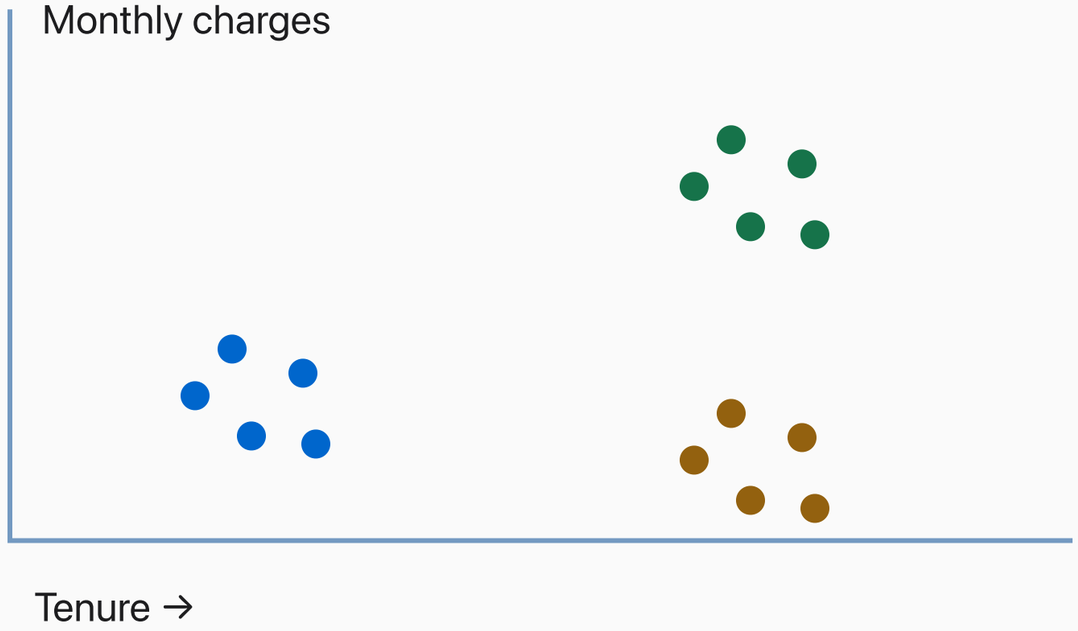
Predicted spend: 65

Absolute error: |65 − 80| = 15

Clustering

Clustering assigns similar feature vectors to groups without using a known outcome label.

A group of short-tenure customers is a pattern to investigate; calling it “likely to churn” requires evidence.



Similarity groups

Coordinates are features.

Colours show cluster assignments.

Churn labels were not used.

Prediction unit

The prediction unit is the entity and time represented by one row. Here it is one customer at a defined observation date.

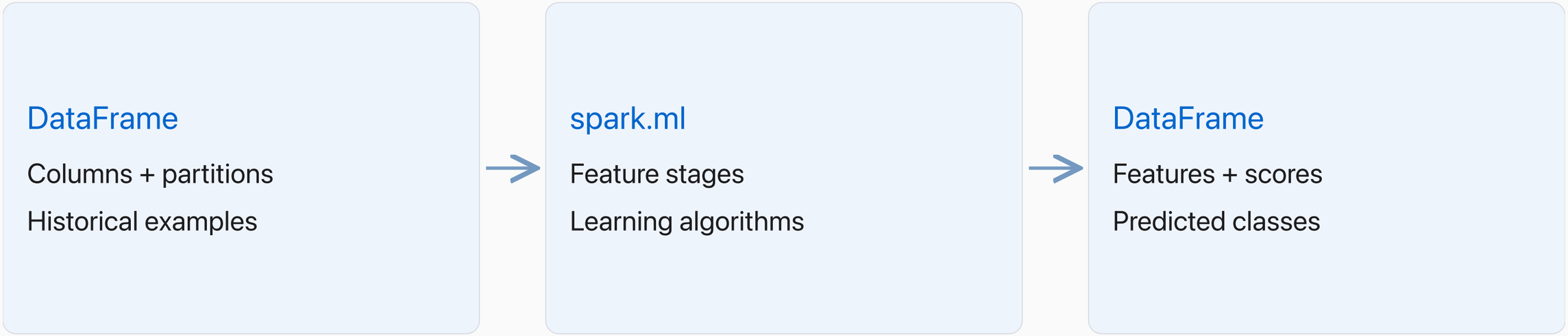
Tenure is time with the service, measured in months. Only information available at the observation date belongs in the feature vector.

Customer at observation	Available information	Later outcome
A at month end	Tenure 2 months; charges 80	Churn = Yes
B at month end	Tenure 24 months; charges 50	Churn = No
C at month end	Tenure 6 months; charges 70	Churn = Yes

Spark ML

MLlib is Spark’s machine learning library. Its DataFrame API, spark.ml, represents features, labels and predictions as columns.

Distributed execution helps process larger datasets; it does not make an invalid target or evaluation valid.



Training, validation and test

Training learns parameters; validation chooses settings; testing estimates performance after those choices are fixed.

The 60/20/20 allocation is illustrative, not a rule; the three roles also apply to very large datasets.

Training

Learn preparation + model

60%

Validation

Choose settings

20%

Test

Final estimate

20%

Illustrative allocation · split by row, customer or time to match deployment.

Splitting very large datasets

Choose validation and test sizes by the precision and coverage needed, not a fixed percentage. With 10 million eligible examples, 1% is already 100,000 examples.

An illustrative 98/1/1 split is not automatically sufficient: at 0.1% positive prevalence, each holdout has only about 100 positives, and smaller subgroups have fewer.

Training · 98%

9,800,000 examples

Fit preparation + model

Validation · 1%

100,000 examples

Compare settings

Test · 1%

100,000 examples

Evaluate the final choice

Random splits

A random split assigns rows probabilistically. It suits a target population where mixing rows does not leak time or entity information.

The weights are expected proportions, not exact row counts; retain row identities as well as a seed.

```
train, valid, test = labeled.randomSplit(
    [0.98, 0.01, 0.01], seed=42
)
```

Expected allocation

10 million eligible input rows

Train: about 9.8 million

Valid / test: about 100,000 each

Time boundaries

A time split trains on earlier observations and evaluates on later observations. It matches a future-prediction use case.

Features and labels must be available at their allowed times; overlapping outcome windows can also leak information.

Train

January–June

Learn from earlier observations

Validate

July–August

Choose settings on later data

Test

September–October

Check the fixed choice



Earlier observations

Later observations

Repeated customers

An entity split keeps all rows for one customer in the same set when the goal is to generalise to new customers.

A random row split can reward recognition of a familiar customer instead of learning a transferable pattern.

Row split

A in train

A again in test

Customer information crosses

Customer split

A only in train

B only in test

New-customer evaluation

Leakage

Leakage occurs when training or selection uses information that would be unavailable for the intended prediction.

It can enter through a feature, a fitted preparation step, repeated entities or repeated inspection of test results.

Leakage route	Example	Why the score misleads
Future feature	Cancellation reason	Outcome is already revealed
Preparation	Scaler fitted on all rows	Held-out distribution influences fit
Repeated entity	Same customer in both sets	Evaluation is not independent
Selection	Choose using test AUC	Test guides the chosen model

Feature availability

A feature is valid only if its value is available when the prediction must be made.

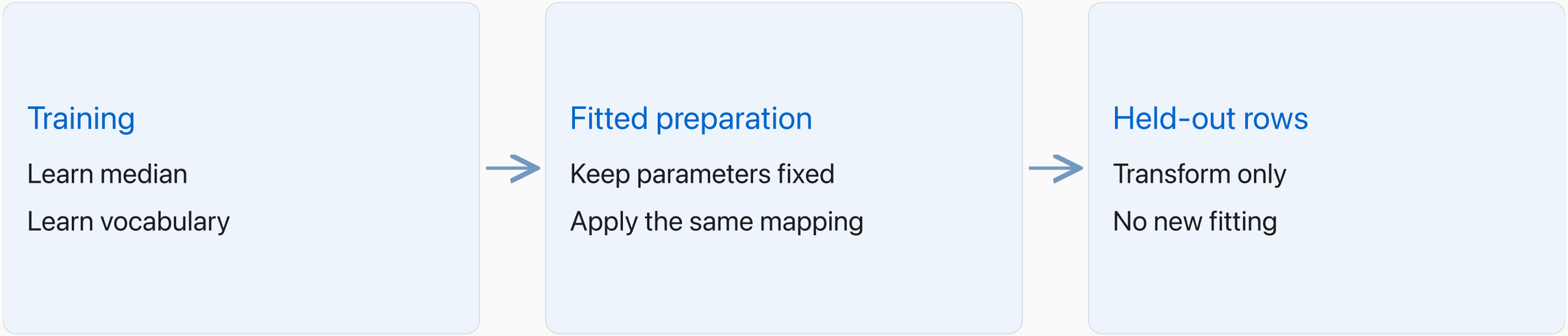
Cancellation date and refund reason describe later events; current tenure and charges can describe the earlier decision.

Candidate feature	Available at observation?	Use for this task
Current tenure	Yes	Candidate input
Current monthly charge	Yes	Candidate input
Later cancellation date	No	Exclude
Refund after leaving	No	Exclude

Fitted preparation

Preparation that learns a mapping or statistic must be fitted on training rows, then reused on validation and test rows.

A fixed parsing rule can be applied consistently to all sets; a learned median or vocabulary belongs inside the fit boundary.



Scaling without leakage

A scaler learns statistics that put numeric features onto a common scale. Fitting on all rows lets validation and test values influence that preparation.

Split first, fit the scaler on training rows only, then reuse it unchanged on validation and test rows.

Training charges: 20, 40, 60 · Test charge: 100

Min–max rule: scaled charge = (charge – minimum) / (maximum – minimum)

Fit on training only

Minimum 20 · maximum 60

Training charge 60:

(60 – 20) / (60 – 20) = 1.0

Fit on training + test

Minimum 20 · maximum 100

Same training charge 60:

(60 – 20) / (100 – 20) = 0.5

The test value changes the training representation when it is included in the fit.

Positive class

The positive class is the outcome treated as label 1. We explicitly use Churn = Yes → 1 and No → 0.

StringIndexer orders labels from data by default, so its index 1 is not automatically “churn”.

```
from pyspark.sql import functions as F
known = raw.filter(F.col("Churn").isin("Yes", "No"))
labeled = known.withColumn(
    "label", (F.col("Churn") == "Yes").cast("double")
)
```

Declared meaning

Yes → 1.0

No → 0.0

Unknown labels: audit separately

Choose the evaluation split

A service has six monthly records per customer. It wants to predict churn for customers never seen in training.

Choose a split and decide whether a median charge should be learned before or after that split.

Available data

Repeated customer IDs

Monthly charges

Known churn labels

Decisions

Split rows or customer IDs?

Fit the median on which rows?

When may test scores be read?

Evaluation boundary

Keep each customer’s records within one set. Learn the median from training customers and apply it unchanged to the others.

Use validation to choose settings, then evaluate the fixed workflow on test customers.

Group by customer

No customer crosses sets

Save split membership

Fit on training

Learn median + model

Validate candidate choices

Final test

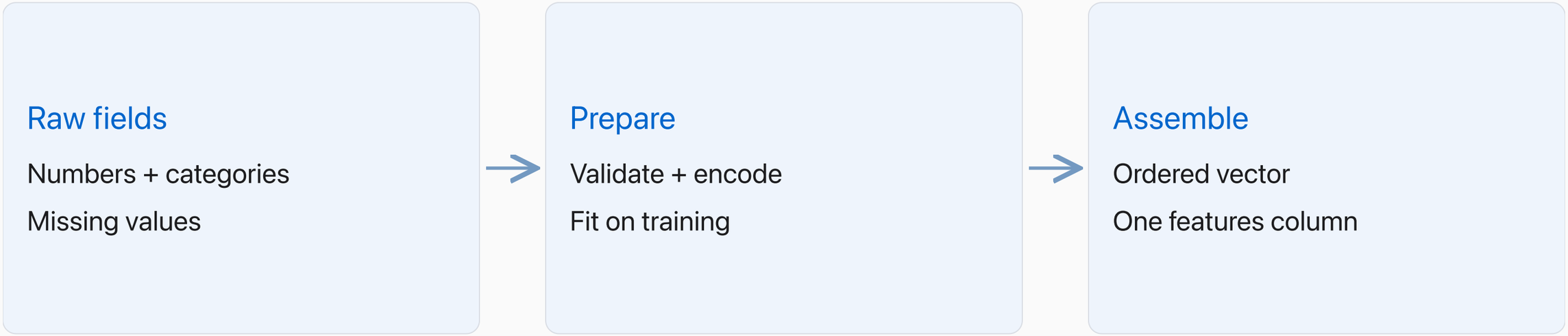
New customers

Fixed preparation + model

Feature engineering

Feature engineering converts available observations into a numerical representation a model can use.

The same input must produce the same feature positions during training and prediction.



Numeric fields

Casting converts a field to the declared numeric type. Failed parses and missing values need an explicit policy.

Dropping invalid TotalCharges rows changes the training population; record how many rows are excluded.

```
def prepare_numeric(frame):
    return (frame.withColumn(
        "TotalCharges", F.col("TotalCharges").cast("double"))
        .filter(F.col("TotalCharges").isNotNull()))
train = prepare_numeric(train)
valid = prepare_numeric(valid)
test = prepare_numeric(test)
```

Policy to inspect

"120.5" → 120.5

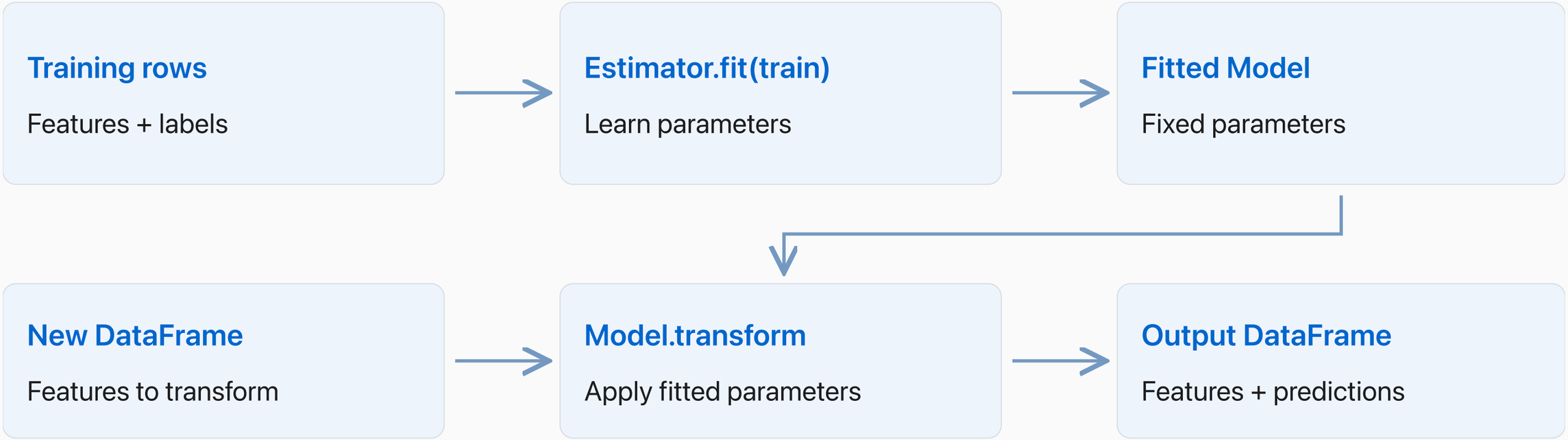
Blank or invalid → inspect

Count excluded rows

Estimator and Transformer

An Estimator learns from data through fit. A Transformer applies a defined operation through transform.

A fitted Model is a Transformer; fitting and transforming serve different purposes.



StringIndexer

StringIndexer learns a mapping from category strings to numeric indices. The indices identify categories rather than measuring their size.

Fit once on training rows and reuse that mapping; do not fit a separate mapping on validation rows.

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(
    inputCol="Contract", outputCol="contract_idx",
    handleInvalid="keep"
)
index_model = indexer.fit(train)
encoded = index_model.transform(valid)
```

Example fitted mapping

Month → 0

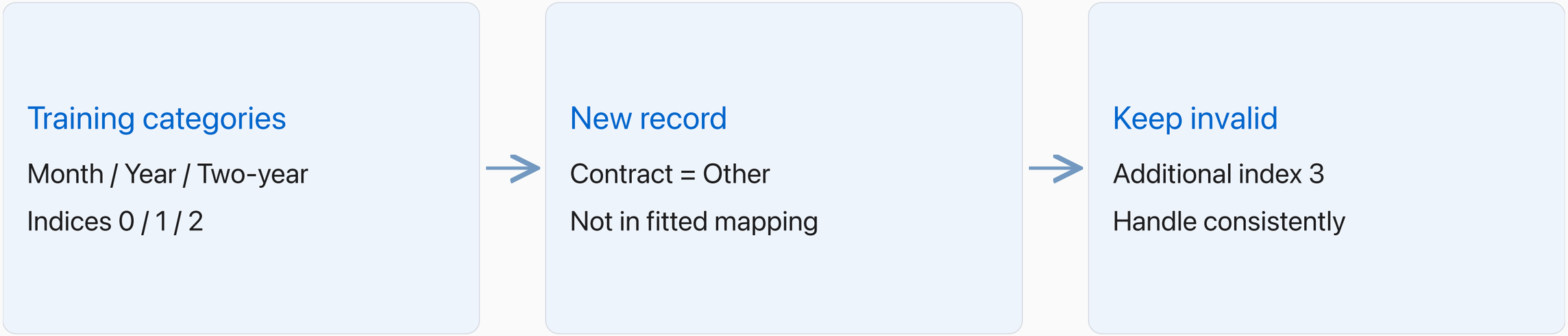
Year → 1

Two-year → 2

Unseen categories

An unseen category was absent when the indexer was fitted. handleInvalid="keep" assigns it to an additional index.

Keeping an unknown category avoids one failure mode; its downstream vector still needs a consistent definition.



One-hot encoding

One-hot encoding represents category membership with separate vector positions. It avoids treating category indices as numeric distances.

This example keeps four positions: three known contract types and the indexer's extra position for unseen categories.



Encoding configuration

OneHotEncoder fits the category-vector representation. dropLast=False keeps the full representation used in the diagram.

With the fitted indexer’s metadata, the four-position contract vector becomes part of the final feature vector.

```
from pyspark.ml.feature import OneHotEncoder
encoder = OneHotEncoder(
    inputCols=["contract_idx"],
    outputCols=["contract_vec"],
    dropLast=False
)
```

Pipeline stage

Input: fitted category index

Output: contract_vec

Vector size follows metadata

VectorAssembler

VectorAssembler combines numeric columns and vector columns into one features vector, in inputCols order.

Changing feature order changes the meaning of the vector even if all its numbers remain the same.

Customer fields

tenure = 12

MonthlyCharges = 70

contract = Month



features

12	70	1	0	0	0
tenure	charges	Month	Year	Two-year	Unseen

Two numeric fields + four contract positions = six features.

Feature order

The assembler defines the feature contract used by training and prediction. Labels and customer IDs are excluded from this input list.

An assembled vector is a representation; it is not yet a trained model.

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(
    inputCols=["tenure", "MonthlyCharges", "contract_vec"],
    outputCol="features"
)
```

Position meanings

0: tenure

1: MonthlyCharges

2 onward: contract vector

Scaling

Scaling changes feature magnitudes using statistics learned from training rows. StandardScaler can divide by standard deviation and optionally subtract the mean.

withMean=False preserves sparse structure; centring can turn a sparse vector into a dense one.

Illustrative values	Divide by sample std = 2	Also subtract mean = 2
0	0	−1
2	1	0
4	2	1

Text features

Tokenization splits text into tokens. A vectorizer represents token occurrences as numbers.

CountVectorizer learns a vocabulary from training text; HashingTF uses a fixed hash space that can contain collisions.



Feature readiness

Before fitting, check types, missing-value policy, category handling and vector order.

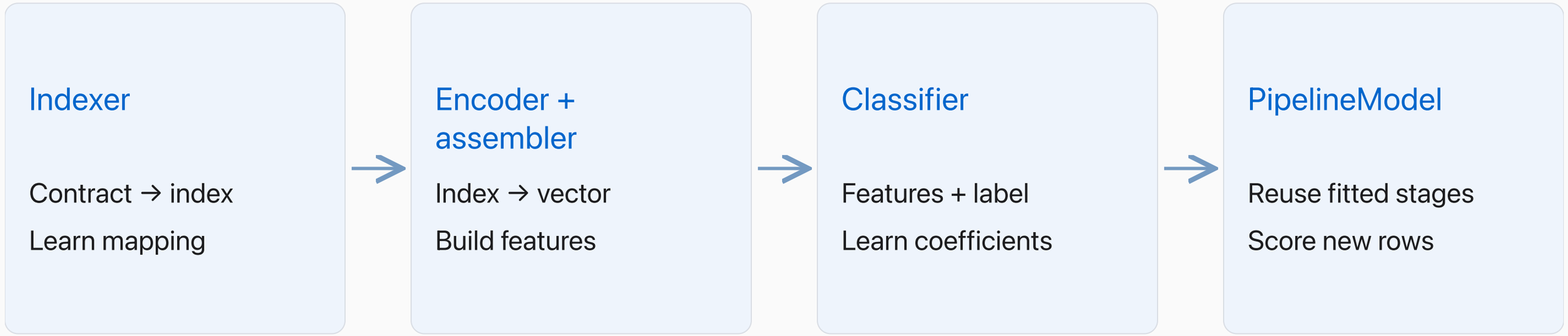
The validation and prediction paths must reuse the training-fitted representation.

Check	Example failure	Consequence
Type	Charges remain strings	Invalid model input
Missing policy	Rows silently disappear	Changed population
Category mapping	Refit on validation	Indices change meaning
Feature order	Charges and tenure swapped	Wrong interpretation

ML Pipeline

A Pipeline is an ordered sequence of Transformers and Estimators. fit(train) fits its learned stages and returns a PipelineModel.

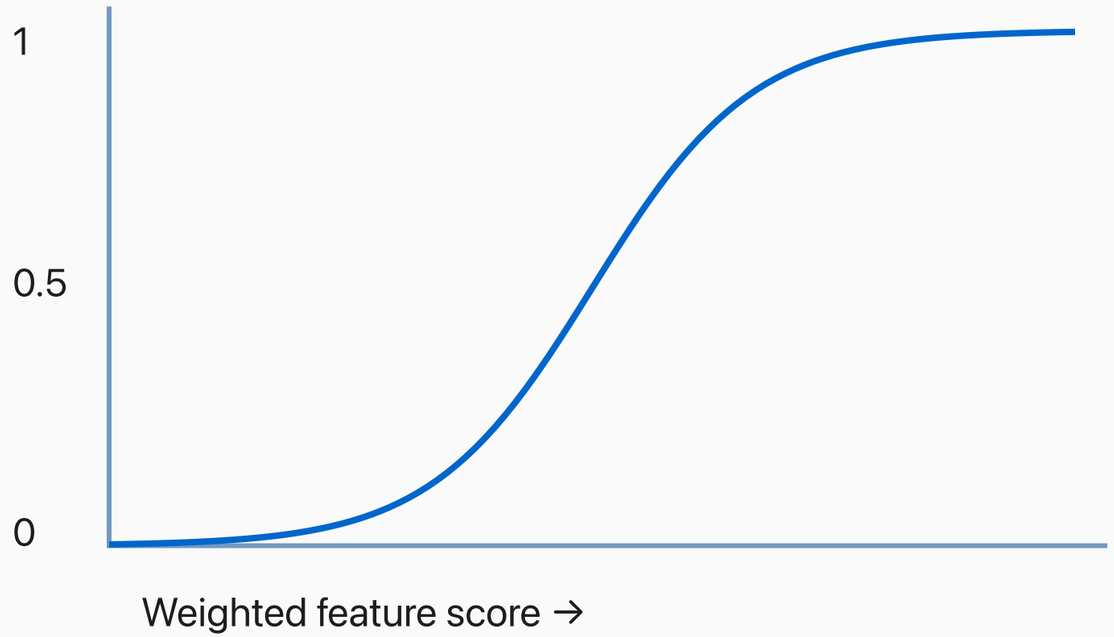
The PipelineModel applies the fitted stages in the same order to validation or new rows.



Logistic regression

Binary logistic regression turns a weighted combination of features into a score between zero and one for class 1.

A decision threshold converts that score into a predicted class; the score is not a guarantee about an individual outcome.



From score to class

z = intercept + weighted features

p = 1 / (1 + exp(−z))

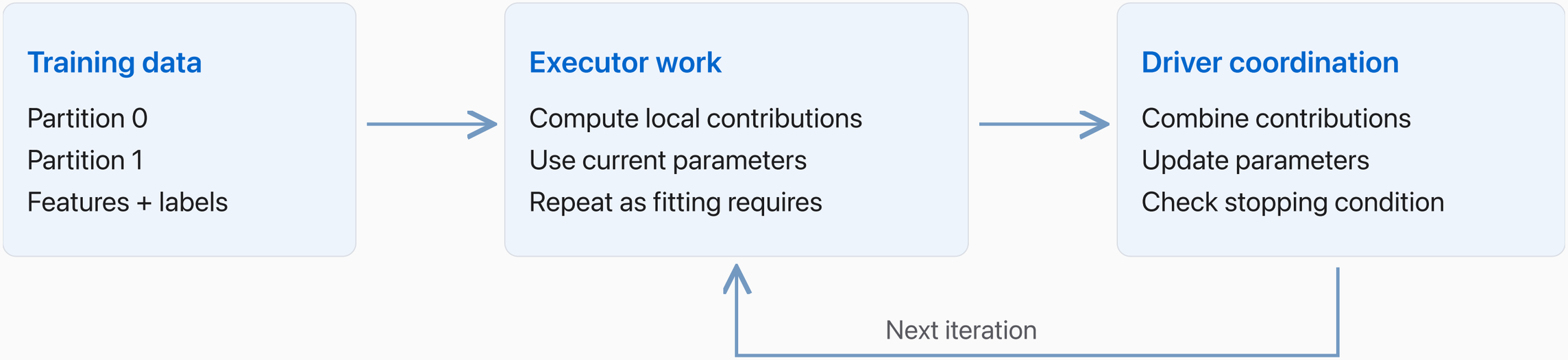
Apply a decision threshold

Class 1 means churn here

Distributed fitting

Fitting logistic regression repeatedly uses training data to update model parameters. Executors compute on partitions and the driver coordinates the work.

One fit can involve several jobs or passes over the data; it is not just a lazy DataFrame assignment.



Training the pipeline

The label mapping and train/validation/test split are already defined. The indexer, encoder and assembler are the stages introduced earlier.

Fit this whole pipeline on train; transform valid without refitting it.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(
    featuresCol="features", labelCol="label")
pipeline = Pipeline(
    stages=[indexer, encoder, assembler, lr])
model = pipeline.fit(train)
predictions = model.transform(valid)
```

Objects returned

pipeline: Estimator

model: PipelineModel

predictions: DataFrame

Prediction columns

label is the observed class; probability contains class scores; prediction is the thresholded class. rawPrediction is the model's unnormalised scoring output.

For our explicit label mapping, probability at index 1 refers to churn.

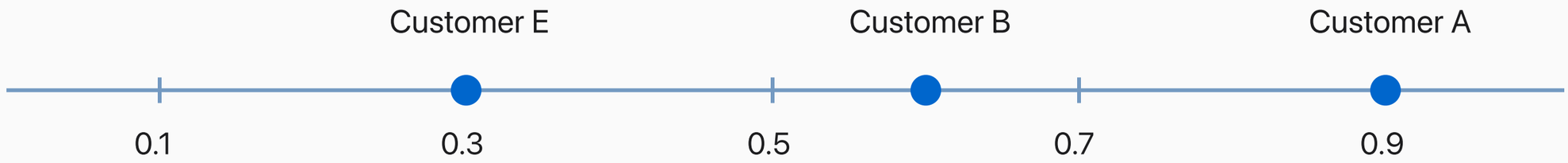
Column	Example	Meaning
label	1.0	Observed churn
rawPrediction	Model-specific score vector	Used by score-based evaluators
probability	[0.2, 0.8]	Scores for classes 0 and 1
prediction	1.0	Class chosen by threshold

Decision threshold

A threshold converts a class-1 score into a class decision. Changing it changes which customers receive an alert without refitting the score model.

Choose the threshold using validation evidence and the cost of missed churn versus unnecessary alerts.

Churn score



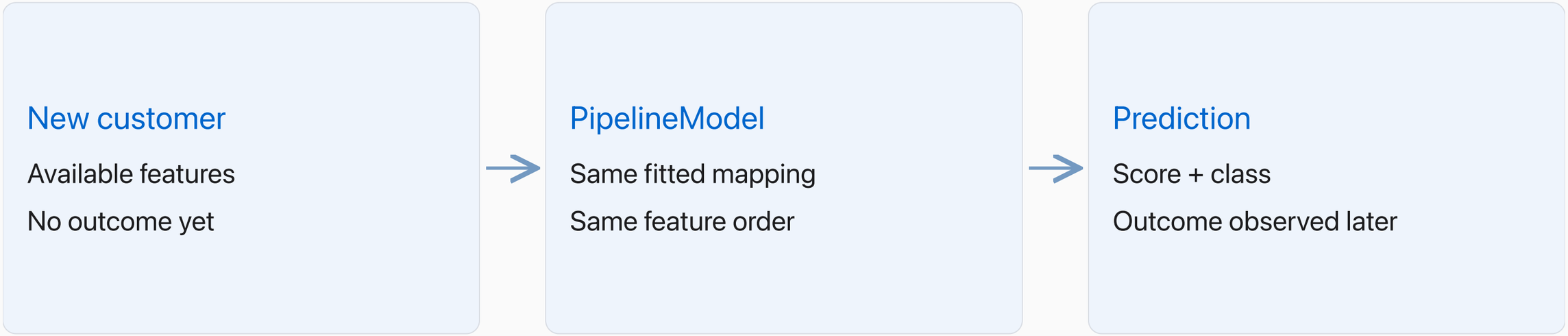
Threshold 0.5: Alert B and A

Threshold 0.7: Alert A only

New-customer prediction

Inference applies the fitted workflow to new feature rows. The true label is not needed to make a prediction.

Evaluation is different: it needs later observed outcomes to judge those predictions.



Saving the workflow

Saving the fitted PipelineModel preserves the preparation stages and model together. A useful record also identifies the feature schema and label meaning.

Use a new versioned destination and verify that a loaded model produces consistent predictions.

```
from pyspark.ml import PipelineModel
model.write().save("models/churn-v1")
loaded = PipelineModel.load("models/churn-v1")
rescored = loaded.transform(valid)
```

Preserve with the artifact

Feature meanings + label mapping

Training data and split identities

Library version + evaluation results

Trace fit and transform

Consider pipeline.fit(train), model.transform(valid), and model.transform(new_customers).

Identify what learns parameters, what returns a fitted workflow, and which calls require a known label.

Expression	Task
pipeline.fit(train)	Identify what is learned
model.transform(valid)	Identify what is reused
model.transform(new_customers)	Decide whether a label is needed

Fitted workflow

fit(train) learns preparation and model parameters. transform applies the fitted workflow and adds predictions.

Labels are needed to train and evaluate this classifier; they are not needed merely to score a new customer.

Expression	Result	Label role
pipeline.fit(train)	PipelineModel	Target used for classifier fitting
model.transform(valid)	Scored DataFrame	Retained for evaluation
model.transform(new_customers)	Scored DataFrame	Not required for prediction

Six scored customers

These six invented validation cases keep score ranking and class decisions visible. Churn is label 1; use threshold 0.5.

The same cases support the confusion matrix, precision, recall and ROC examples.

Customer	Actual label	Churn score
A	1	0.9
B	1	0.6
C	0	0.8
D	0	0.4
E	1	0.3
F	0	0.1

Confusion matrix

A confusion matrix counts correct and incorrect class decisions. Positive means churn, and predicted positive means an alert.

At threshold 0.5: two true positives, one false positive, one false negative and two true negatives.

	Alert	No alert
Actual churn	**True positive** A, B Count = 2	**False negative** E Count = 1
Actual no churn	**False positive** C Count = 1	**True negative** D, F Count = 2

Precision and recall

Precision is the share of alerts that are correct. Recall is the share of actual churn cases that receive an alert.

For the six cases, precision = 2/3 and recall = 2/3; the equal values come from different denominators.

Precision

TP / (TP + FP)

2 / (2 + 1) = 2/3

Among the three alerts

Recall

TP / (TP + FN)

2 / (2 + 1) = 2/3

Among the three churn cases

Accuracy and imbalance

Accuracy is the fraction of all class decisions that are correct. With rare positives, it can hide a failure to detect the outcome of interest.

If only 5 of 100 customers churn, predicting “no churn” for everyone gives 95% accuracy and zero churn recall.

Six-case classifier

4 correct / 6 total

Accuracy = 66.7%

Two churn cases detected

Imbalanced baseline

95 correct / 100 total

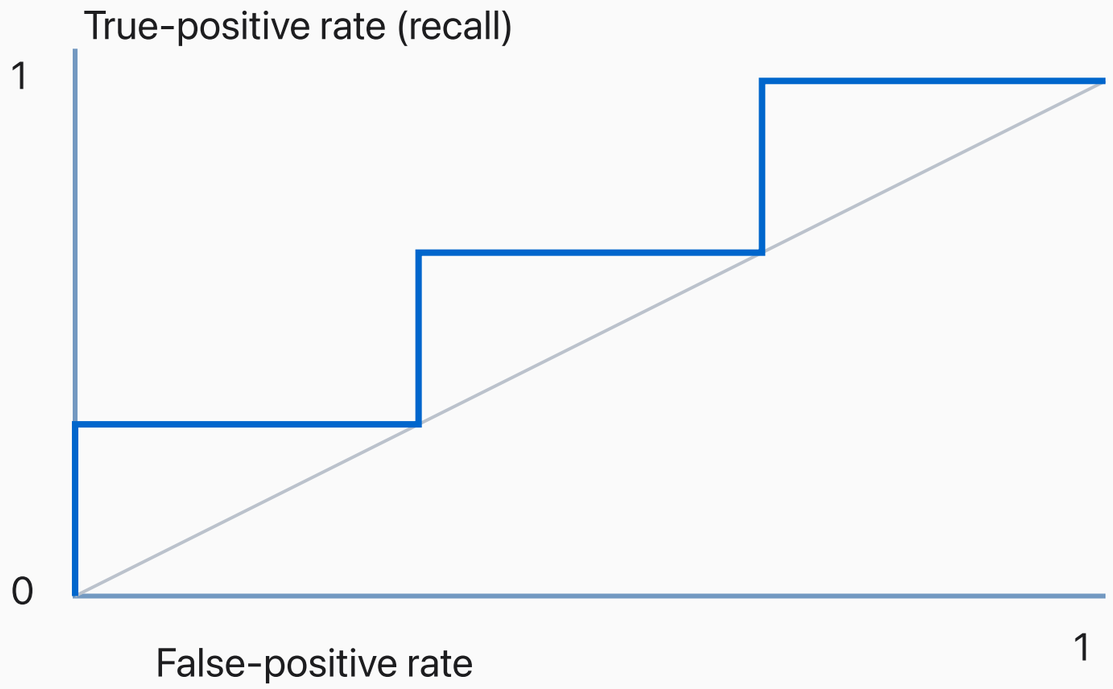
Accuracy = 95%

No churn cases detected

ROC AUC

The receiver operating characteristic (ROC) curve compares true-positive rate with false-positive rate across score thresholds. AUC is its area.

It measures score ranking across thresholds, not the accuracy of one selected threshold.



Six-case example

At threshold 0.5:

True-positive rate = 2/3

False-positive rate = 1/3

Ranking pairs

ROC AUC can be interpreted as the probability that a random positive case scores above a random negative case, with ties receiving half credit.

Here the positive scores win 6 of 9 positive–negative comparisons, so ROC AUC is 2/3.

Positive score	Negative scores beaten	Wins
0.9	0.8, 0.4, 0.1	3
0.6	0.4, 0.1	2
0.3	0.1	1
Total	3 positives × 3 negatives	6 / 9

Precision–recall trade-off

A precision–recall curve evaluates alerts across thresholds. Its summary area is different from accuracy and from ROC AUC.

On the same six cases, lowering the threshold from 0.5 to 0.2 detects every churn case but adds an alert for D.

Threshold	Precision	Recall
0.7: alert A, C	1 / 2	1 / 3
0.5: alert A, B, C	2 / 3	2 / 3
0.2: alert A–E	3 / 5	3 / 3

BinaryClassificationEvaluator

BinaryClassificationEvaluator compares score outputs with known binary labels. metricName selects the metric that is computed.

The variable name cannot change the metric: areaUnderROC is ROC AUC even if a program calls the result “accuracy”.

```
from pyspark.ml.evaluation import (
    BinaryClassificationEvaluator
)
evaluator = BinaryClassificationEvaluator(
    labelCol="label", rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)
validation_auc = evaluator.evaluate(predictions)
```

Report accurately

Metric: validation ROC AUC

Positive class: churn = 1

Data: held-out validation rows

Hyperparameter selection

Hyperparameters are settings chosen outside model fitting, such as regularisation strength. Validation compares candidate settings.

Two regParam values and two elasticNetParam values create four candidates; test outcomes must not select the winner.

```
from pyspark.ml.tuning import ParamGridBuilder
grid = (ParamGridBuilder()
    .addGrid(lr.regParam, [0.0, 0.1])
    .addGrid(lr.elasticNetParam, [0.0, 1.0])
    .build())
```

Four combinations

(0.0, 0.0) · (0.0, 1.0)

(0.1, 0.0) · (0.1, 1.0)

Fit on train; compare on validation

Cross-validation

Cross-validation rotates the held-out fold within the development data and averages validation results. It helps compare settings when one split is unstable.

With 4 candidates and 3 folds, there are 12 fold fits, plus a final refit; learned preparation must be inside the tuned pipeline.

Development data · the final test set stays outside

Fit 1	Fold 1: validate	Fold 2: train	Fold 3: train
Fit 2	Fold 1: train	Fold 2: validate	Fold 3: train
Fit 3	Fold 1: train	Fold 2: train	Fold 3: validate

Baseline and model choice

Compare a candidate with a simple baseline on the same held-out data and metric. A random forest is an alternative learner, not automatically a better solution.

Keep the split and feature contract fixed; weigh ranking, alert errors, training cost and prediction cost.

Candidate	Evidence to compare	Decision
Always no churn	Accuracy high if churn is rare	Measure missed positives
Logistic regression	Ranking + threshold metrics	Simple reference model
Random forest	Same metrics and split	Justify extra complexity

Lab 09

Lab 09 uses tenure, MonthlyCharges and TotalCharges to train logistic regression. It prints areaUnderROC with an “accuracy” label.

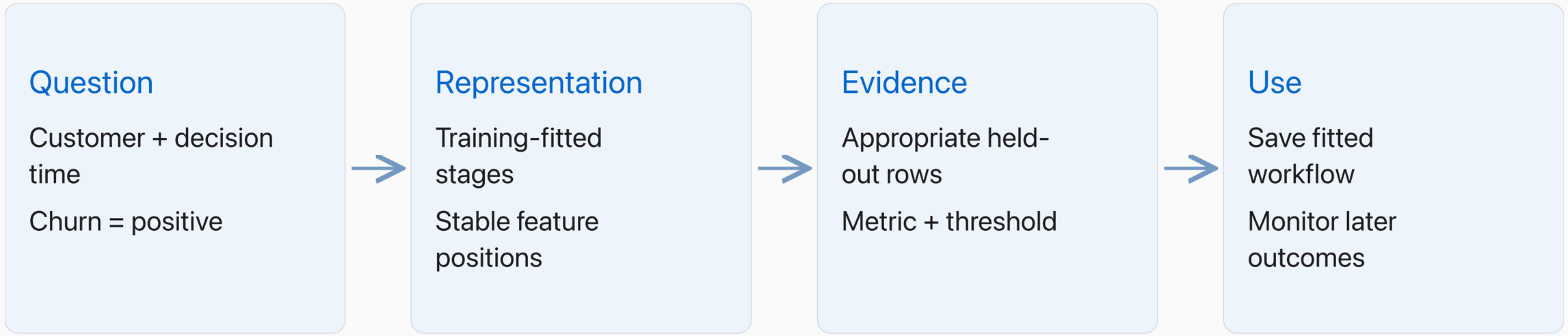
Inspect the fitted Churn mapping and report ROC AUC. The lecture’s stronger workflow splits before learned preparation and reserves validation for tuning.

Lab step	What to understand
StringIndexer on Churn	Inspect which class is index 1
VectorAssembler	Three numeric feature positions
randomSplit 70/30	A train/test demonstration
BinaryClassificationEvaluator	ROC AUC, despite the printed label
Cleanup	Preserve evidence, then terminate cluster

Prediction and evaluation

A useful prediction workflow connects an available feature set to a defined outcome, then tests that workflow on appropriate unseen cases.

A high score is interpretable only when the label, split, metric and selection procedure are clear.



Evaluation roles

Training learns parameters, validation selects choices, and testing estimates the fixed workflow’s generalisation.

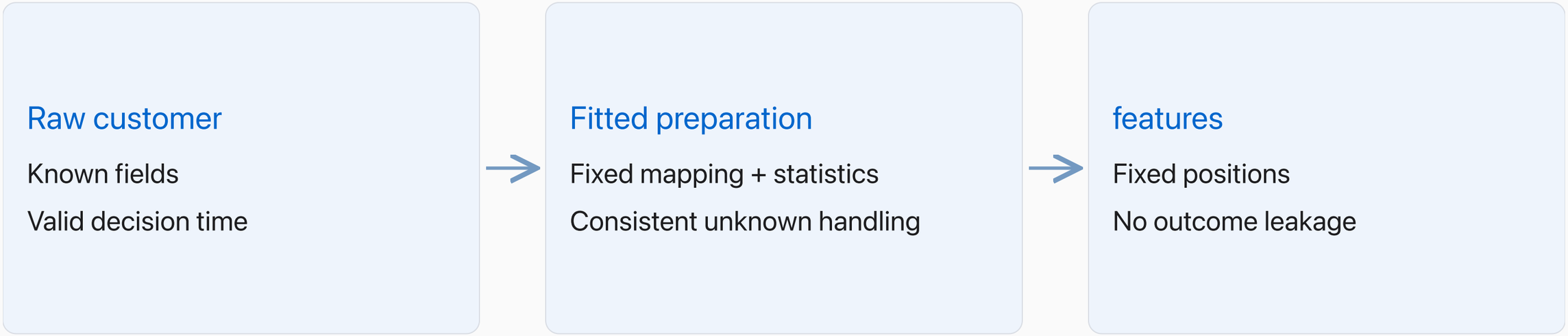
Entity and time boundaries must match the intended deployment population.

Set	Permitted use	What would invalidate it
Training	Fit preparation and model	Unavailable future features
Validation	Select settings and threshold	Repeated entities leaking from train
Test	Final estimate after selection	Choosing the winner from test scores

Feature contract

A fitted workflow must preserve field meaning, missing-value policy, category mappings and vector order.

The same customer information should receive the same representation during validation and deployment.



fit and transform

fit learns from training data and returns a Model. transform applies a Transformer or fitted Model to a DataFrame.

Prediction can happen without labels; evaluation needs known outcomes to compare with the scores.

Operation	Input	Output
Estimator.fit	Training examples	Fitted Model
Transformer.transform	DataFrame	Transformed DataFrame
Model.transform	New features	Scores and predictions
Evaluator.evaluate	Scores + known labels	A named metric

Metric meanings

Precision judges the alerts, recall judges coverage of actual positives, accuracy counts all correct labels, and ROC AUC measures ranking.

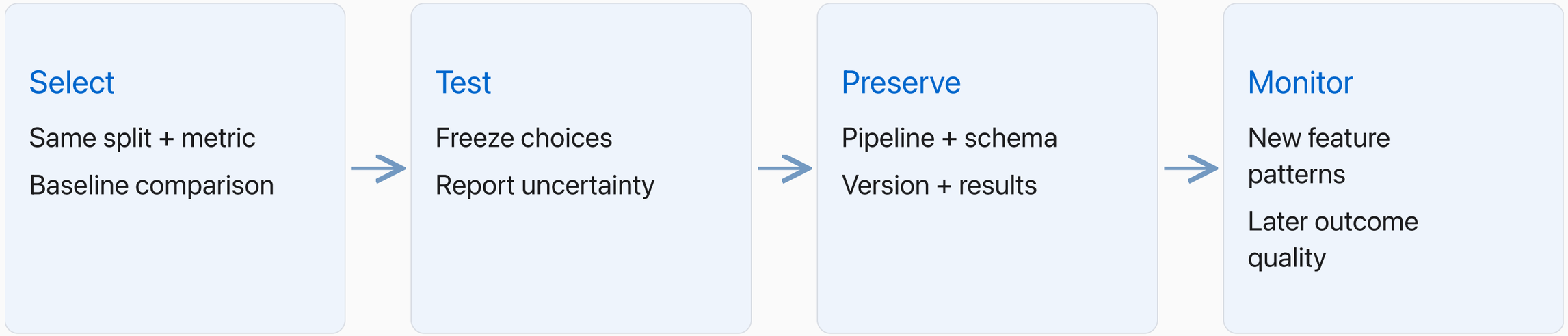
Choose a metric and threshold that match the decision; no single number describes every error cost.

Measure	Question answered
Precision	How many alerts are correct?
Recall	How many actual positives are found?
Accuracy	How many class decisions are correct?
ROC AUC	How well are positives ranked above negatives?
PR curve	How do alert precision and recall vary?

Selection and reuse

Select settings using validation evidence, evaluate the fixed choice on test data, and preserve the fitted pipeline with its provenance.

Reusing a test set for repeated improvements makes its score optimistic; a saved model still needs monitoring on new data.



Final churn case

A team learns category mappings from all rows, chooses a model using test AUC, and predicts no churn for everyone in a population with 5% churn.

Identify two evaluation errors, calculate accuracy and recall, and propose a corrected workflow.

Workflow

Indexer fitted on all rows

Winner chosen on test AUC

Predictions

100 customers

5 actual churn cases

No positive predictions

A defensible churn result

The preparation and selection steps both use held-out information. The no-alert rule has 95% accuracy and 0% churn recall.

Split appropriately, fit preparation on training rows, select with validation, then test the fixed pipeline and report the actual metric.

Issue	Reasoned correction
Mapping fitted on all rows	Fit it on training; reuse on held-out rows
Winner chosen by test AUC	Use validation for selection
No-alert baseline	95 correct out of 100; 0 of 5 churn found
Final evidence	State positive class, metric, split and threshold