Building Classification and Regression Models in Spark ML

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

High level abstractions such as Estimators and Transformers

Chained together in a pipeline i.e. machine learning workflow

Special libraries for feature engineering

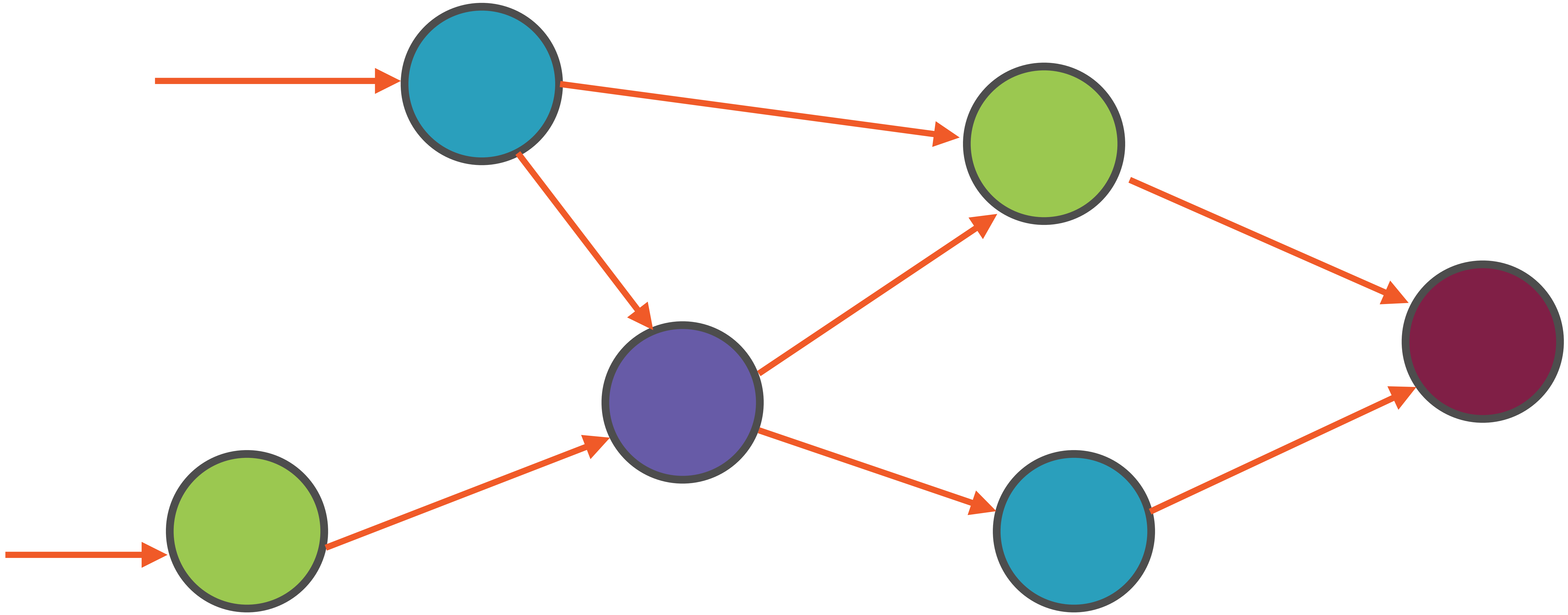
Evaluating classifiers using the confusion matrix

Decision trees and random forests for classification

Specialized regression models such as Lasso and Ridge regression

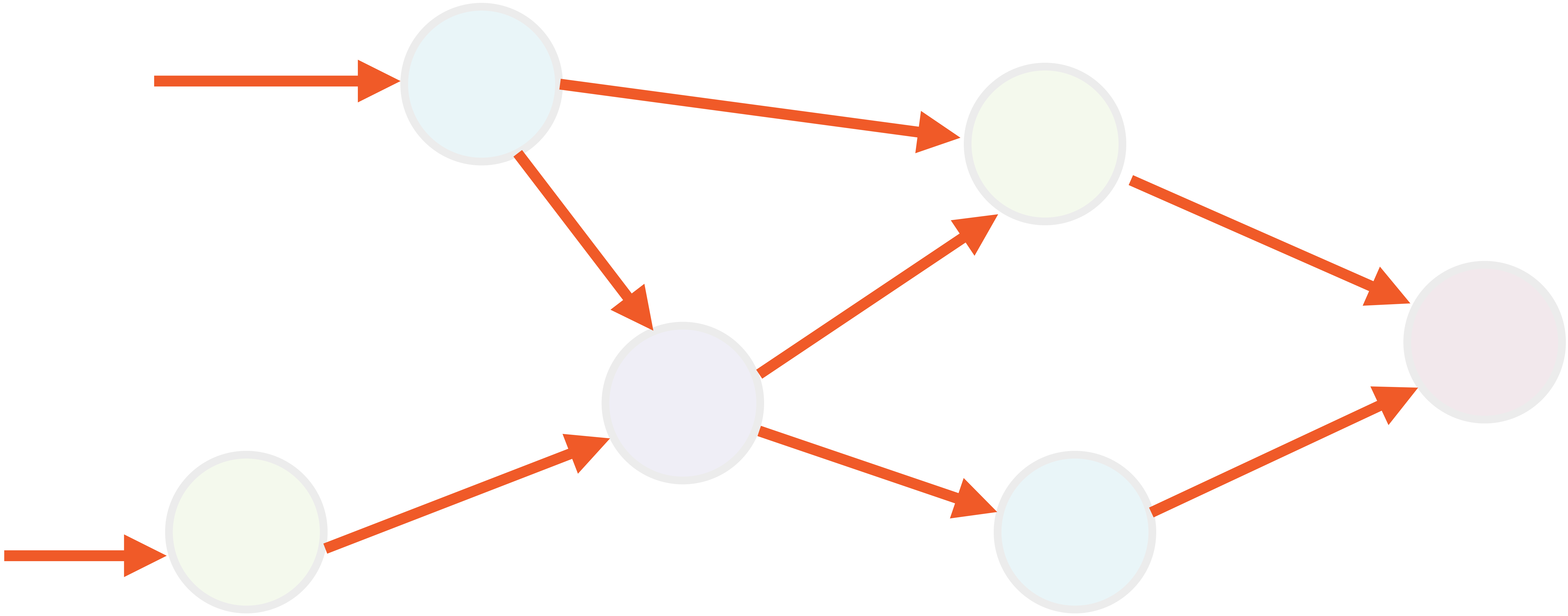
ML Pipelines, Estimators and Transformers

ML Model as Pipeline



A network

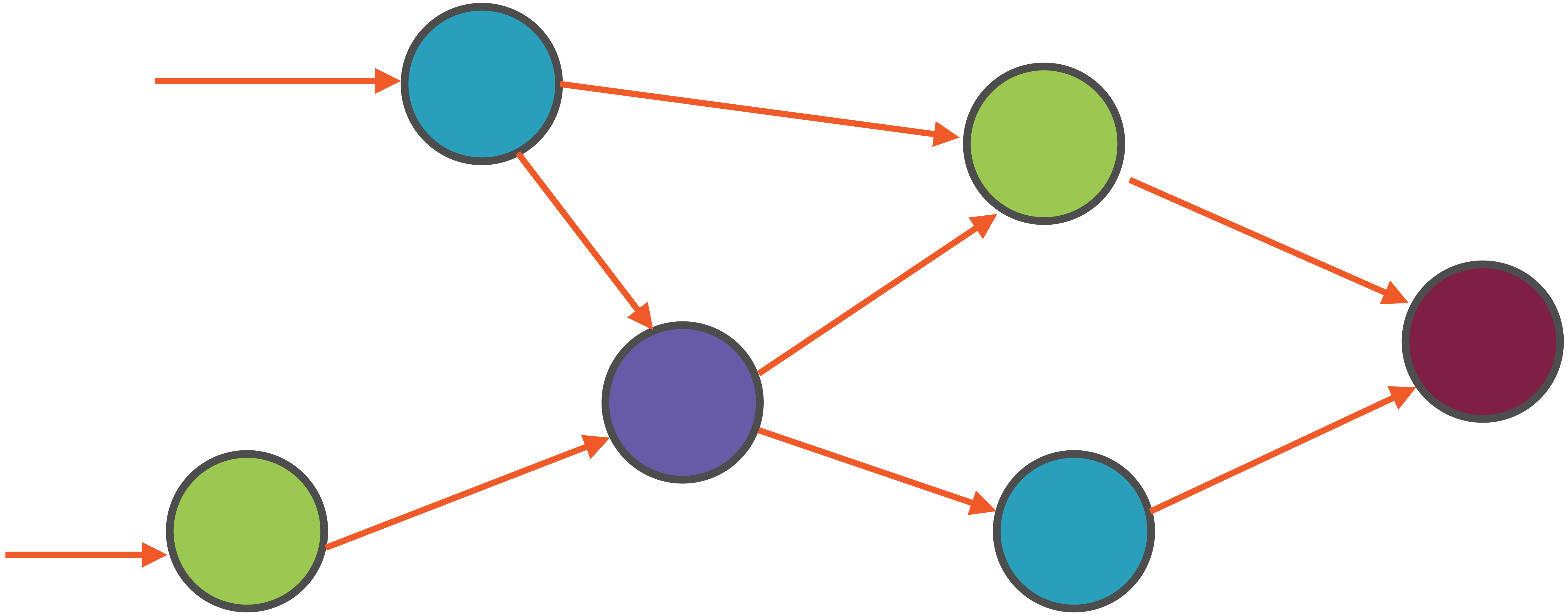
ML Model as Pipeline



DataFrames

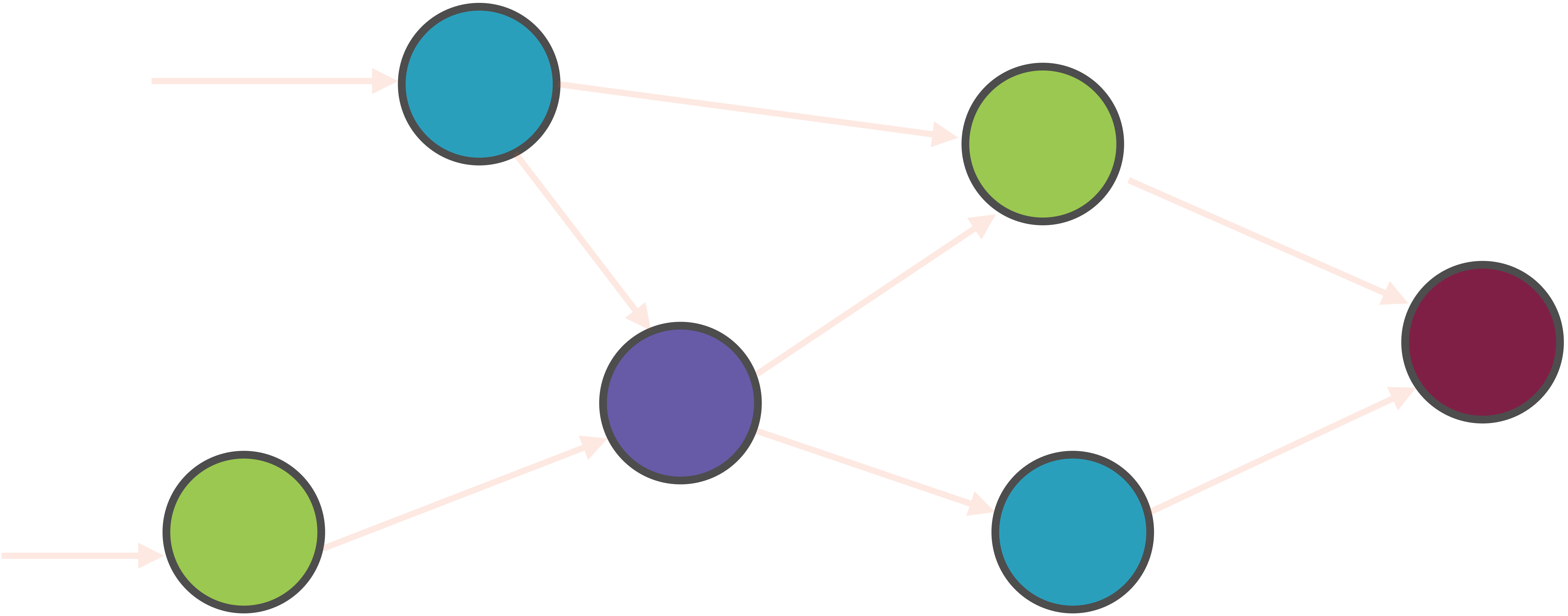
Data

DataFrames Flow Through the Pipeline



...and get transformed along the way

ML Model as Pipeline



Computations

Estimators or Transformers

Pipeline Concepts

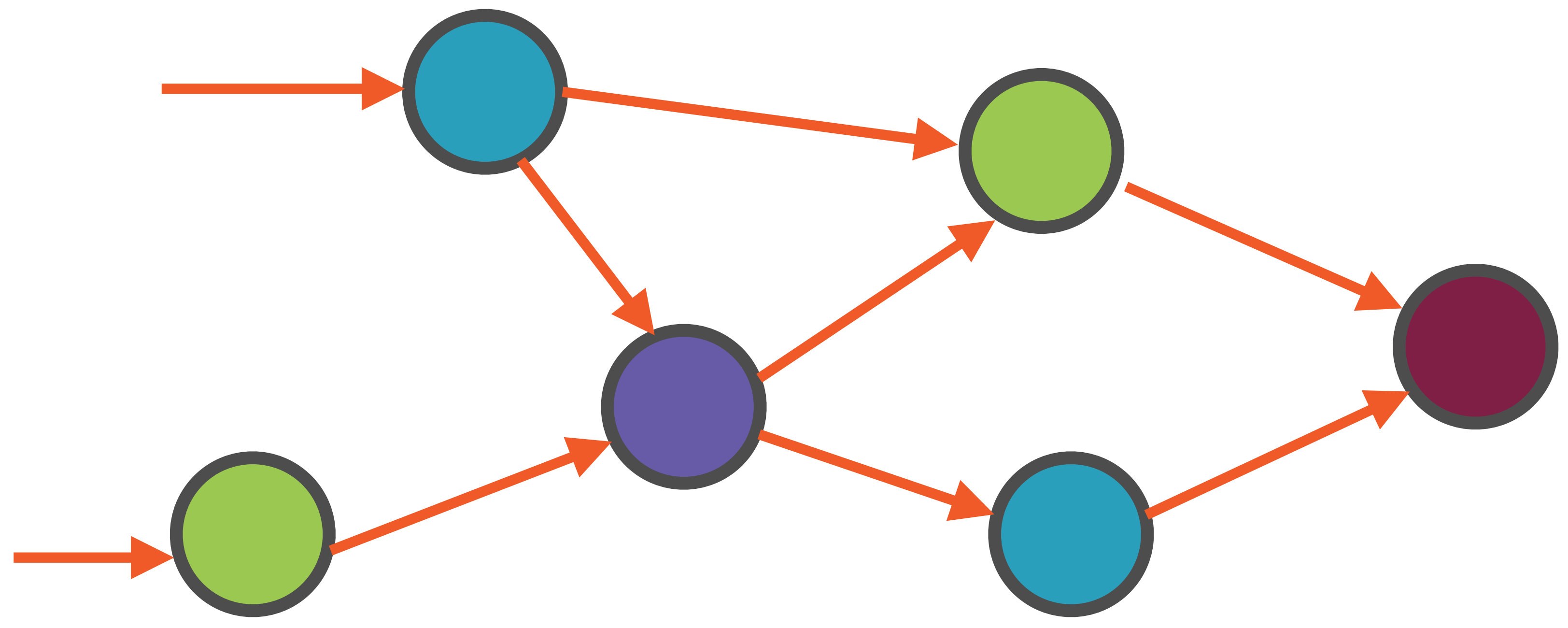
DataFrames

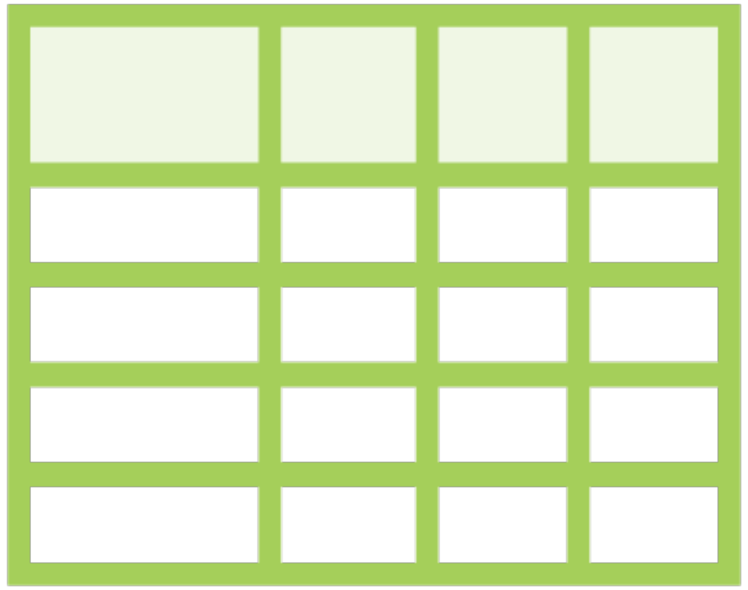
Transformers

Estimators

Pipeline

Parameter

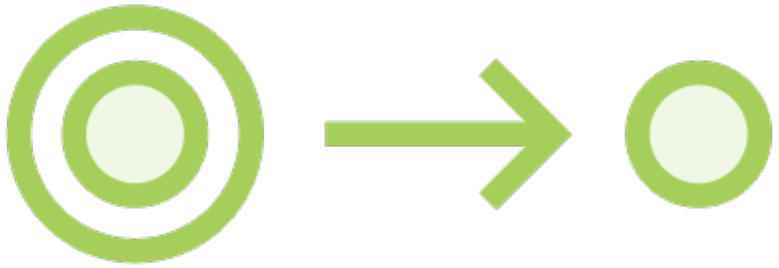




DataFrame

Use to represent the ML dataset for training

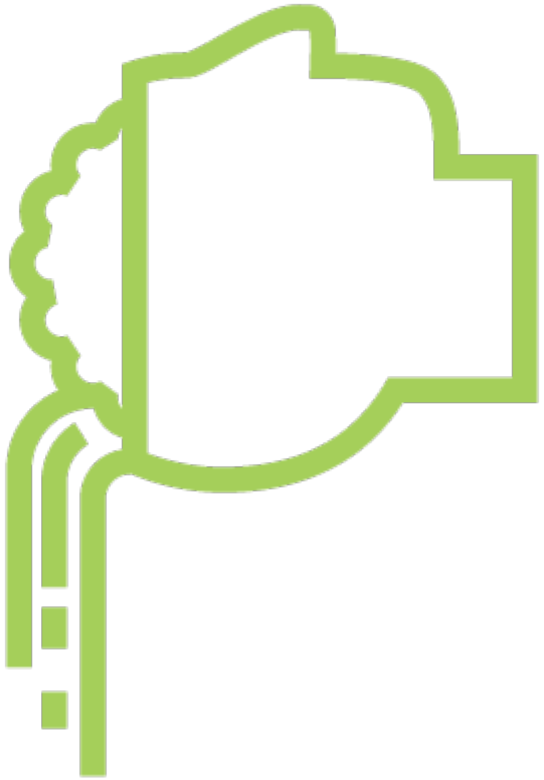
Different columns for features, labels, predictions



Transformer

Algorithm to convert one DataFrame to another

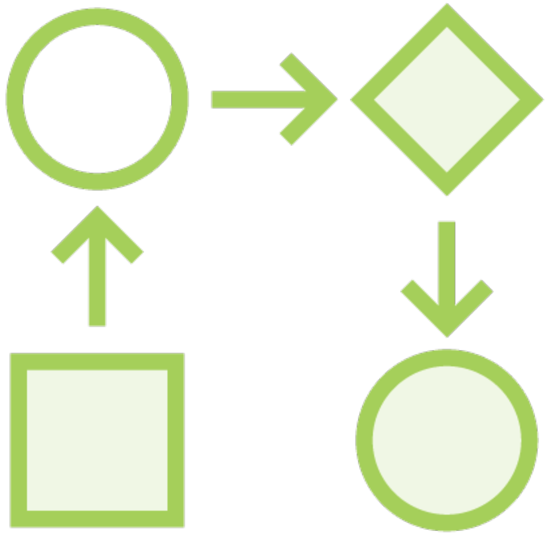
DataFrame with features -> DataFrame with predictions



Estimator

An algorithm that fits on a DataFrame to produce a Transformer

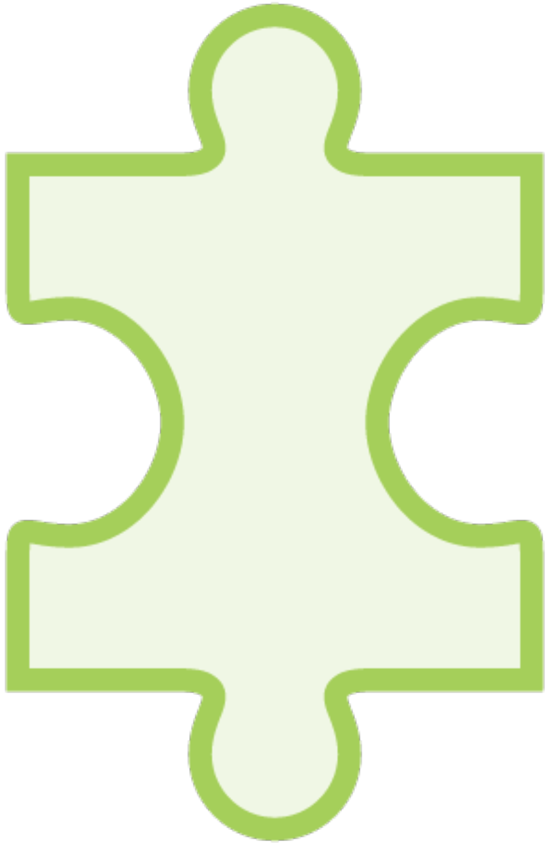
An ML algorithm -> trains on input data -> produces a model



Pipeline

Chains Estimators and Transformers to form a machine learning workflow

Chains a series of operations to be performed on DataFrames



Parameter

Design settings in ML algorithms that can be tuned

Transformers and Estimators have a common API for parameters

Pipeline Concepts

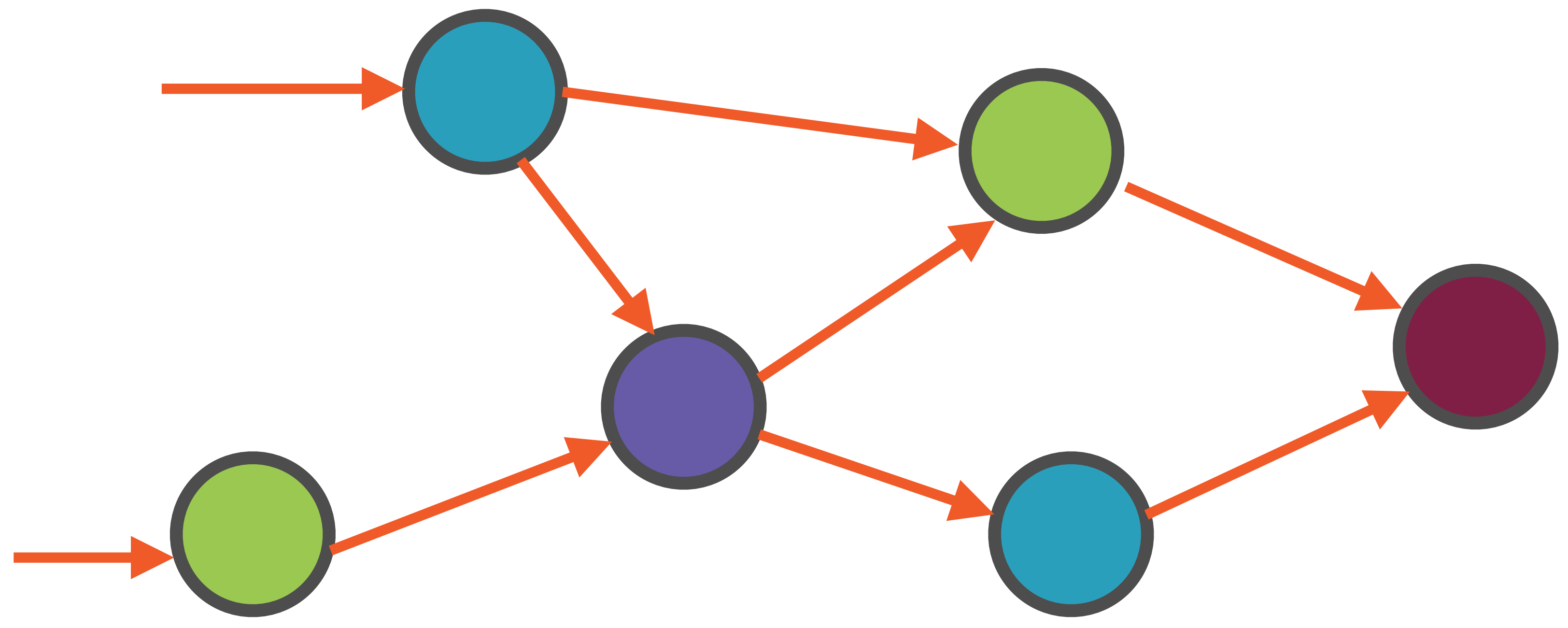
DataFrames

Transformers

Estimators

Pipeline

Parameter



Pipeline Stages

Estimator Stages

DataFrame in, Transformer out

Implement a fit() method

Obtain trained machine learning model by invoking fit()

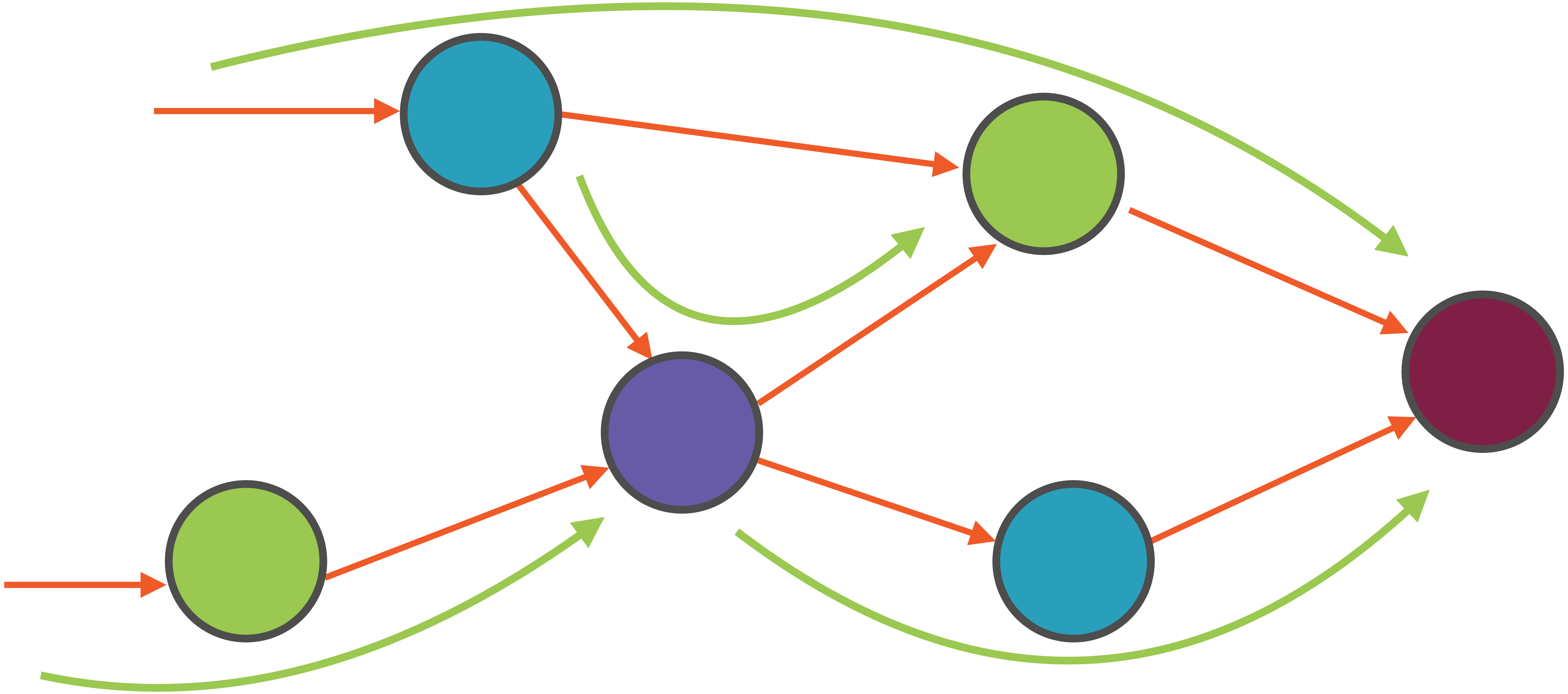
Transformer Stages

DataFrame in, DataFrame out

Implement a transform() method

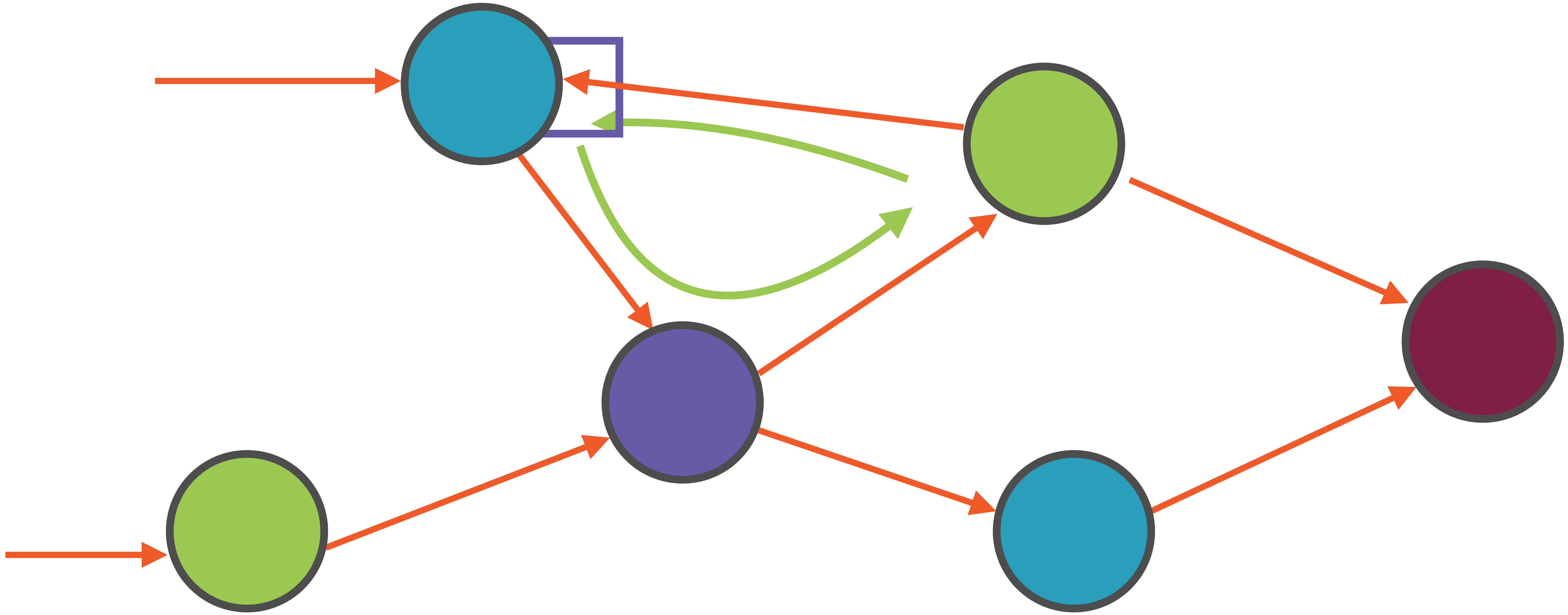
Transform features or carry out prediction using transform()

Pipelines Are DAGs



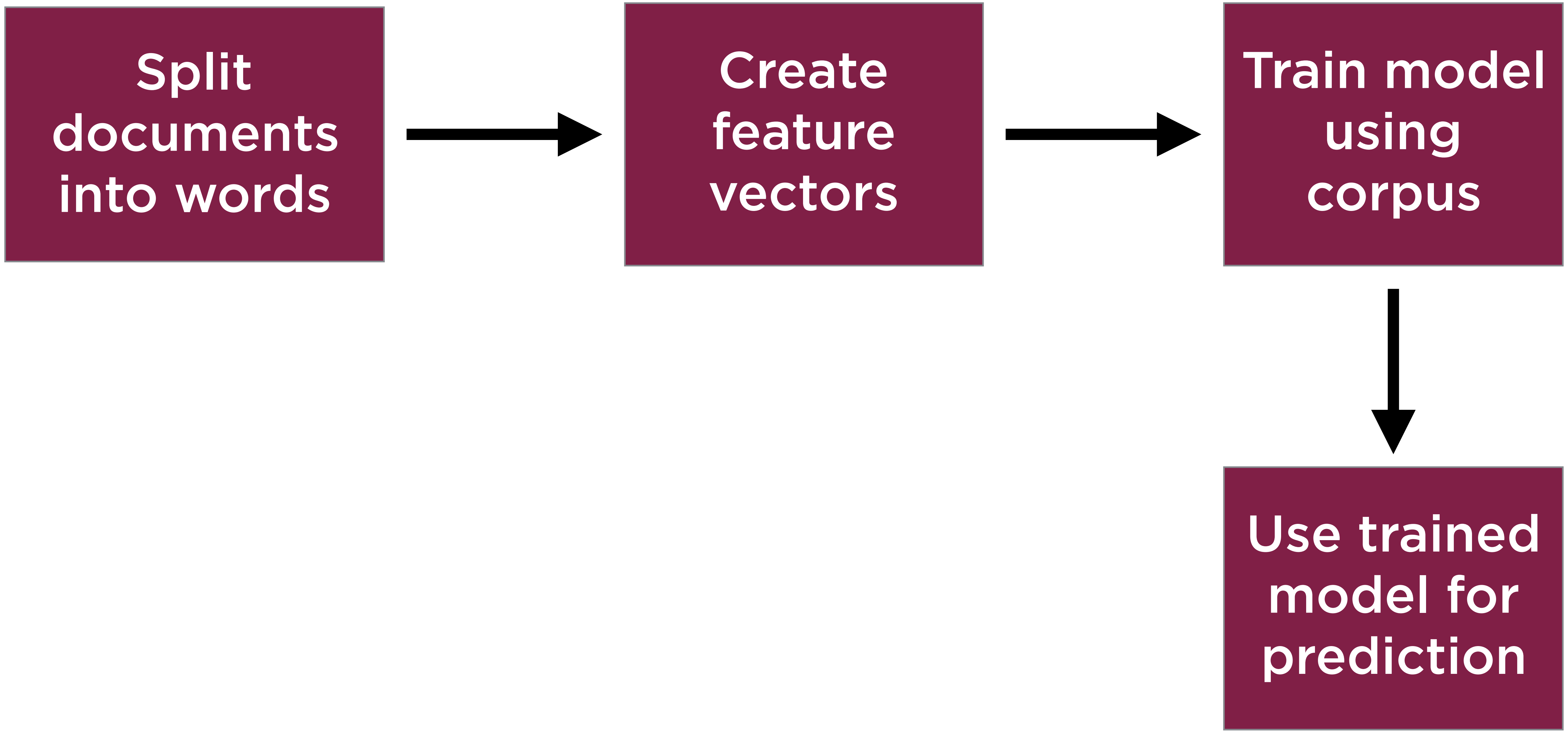
There are no cycles in the graph - acyclic

Pipelines Are DAGs



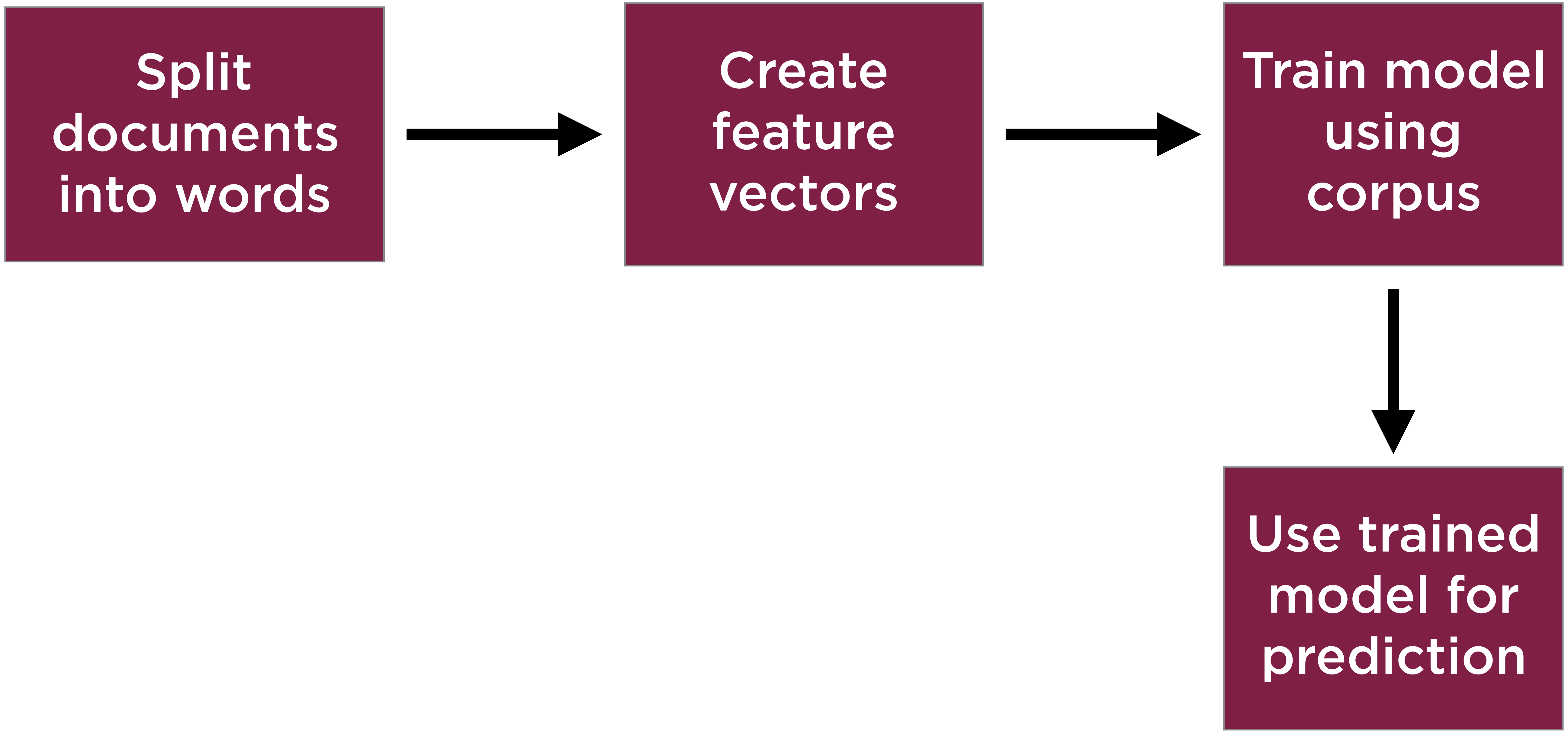
A graph with cycles will never finish computation

ML Pipeline for Sentiment Analysis

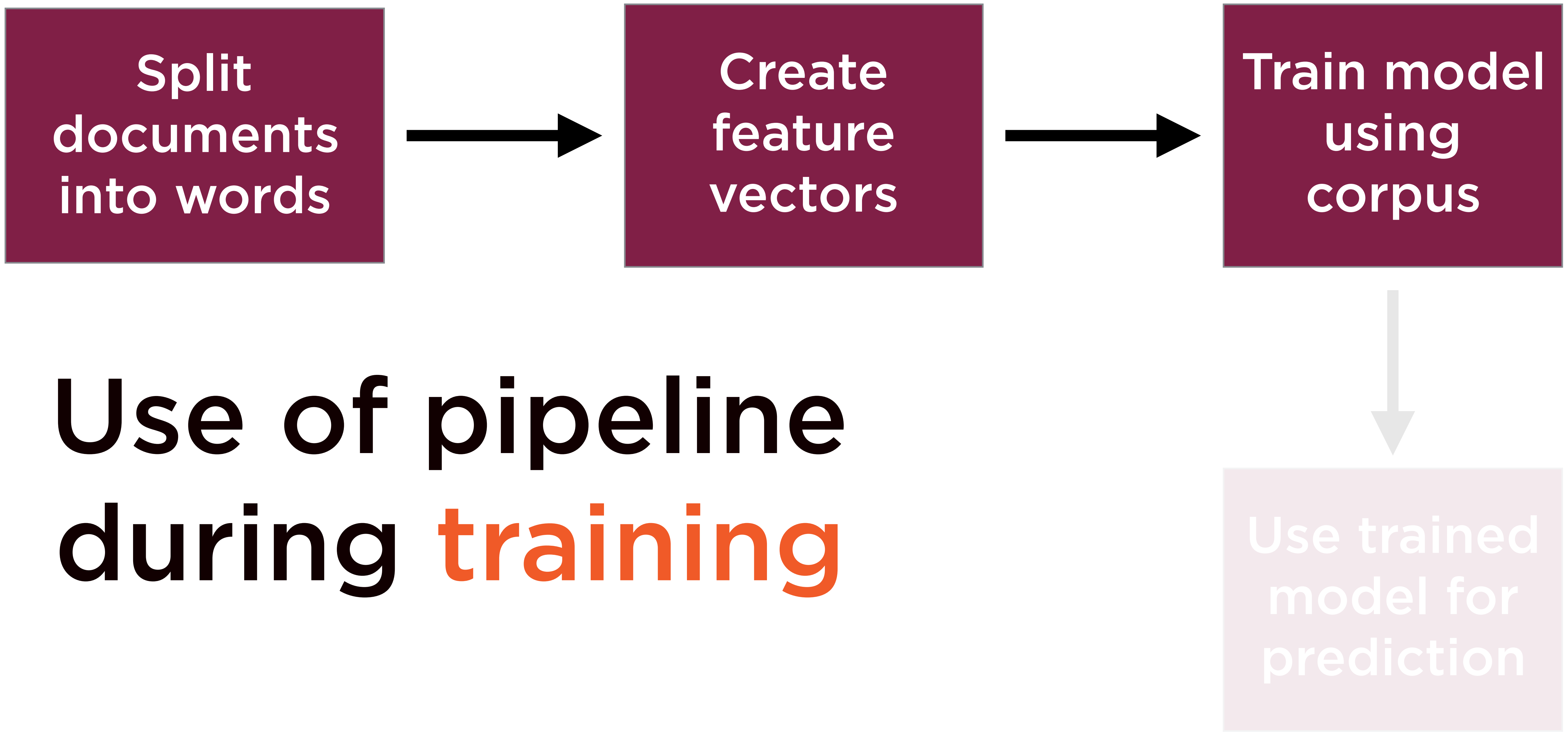


Pipeline Stages in Training and Prediction

ML Pipeline for Sentiment Analysis



ML Pipeline for Sentiment Analysis



Use of pipeline during training

ML Pipeline for Sentiment Analysis

Split documents into words

Create feature vectors

Train model using corpus

Use trained model for prediction

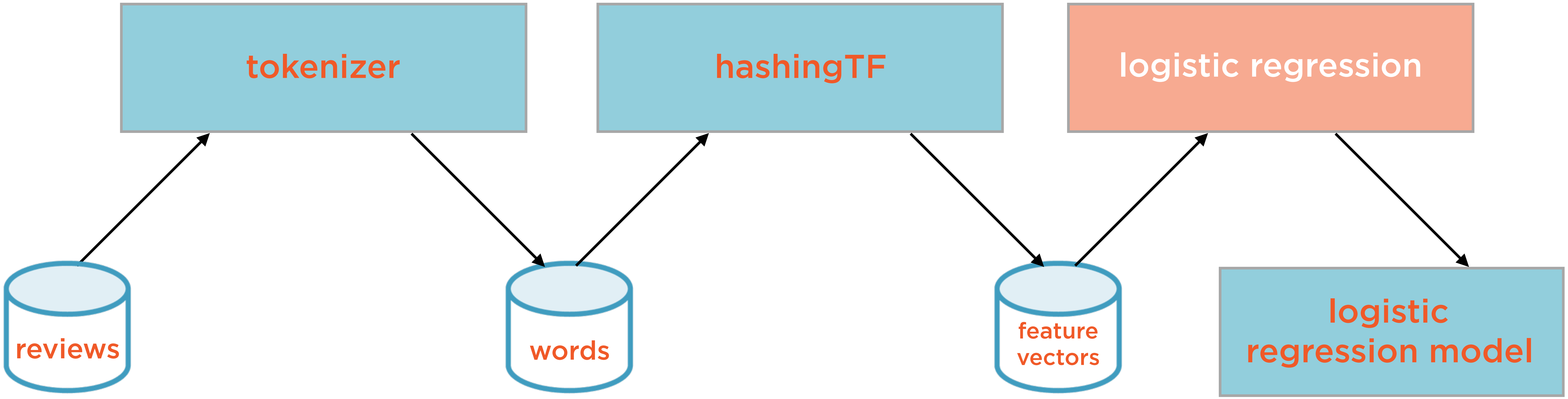
Use of pipeline during prediction

ML Pipeline for Sentiment Analysis

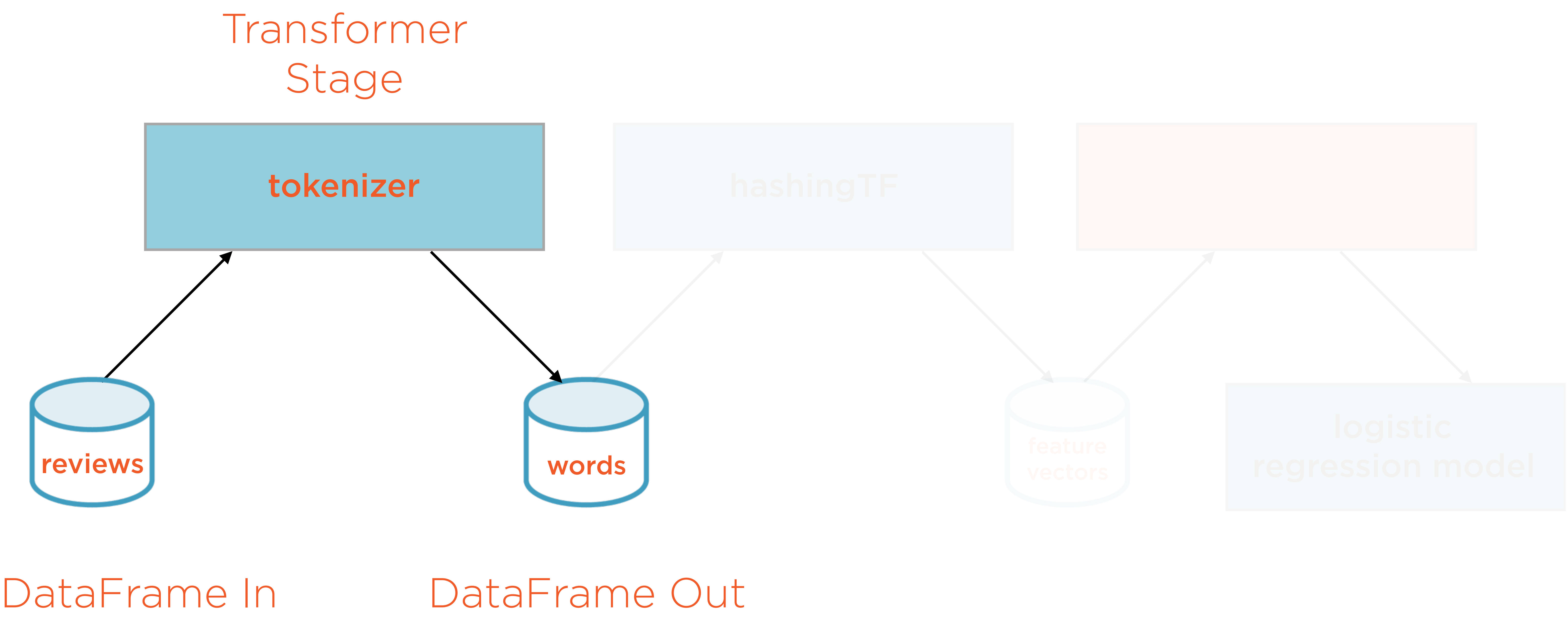
Split documents into words → Create feature vectors → Train model using corpus → Use trained model for prediction

Use of pipeline during training

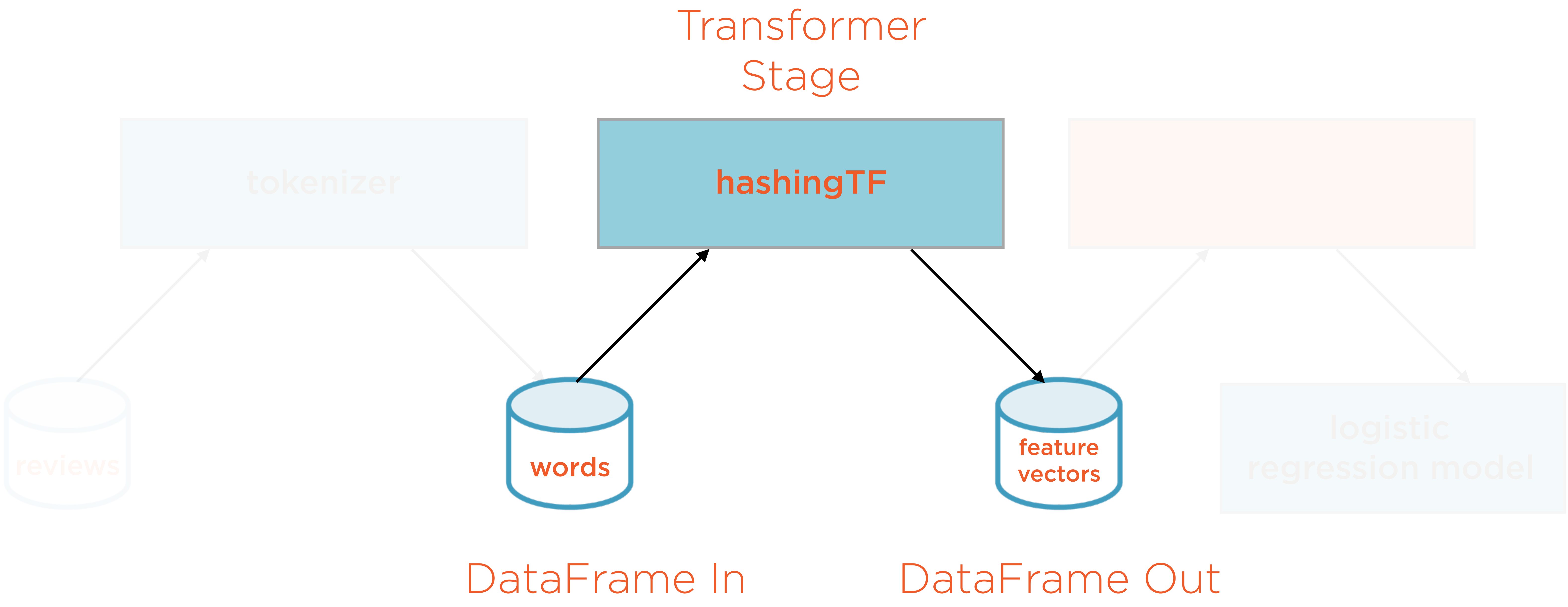
Training Pipeline



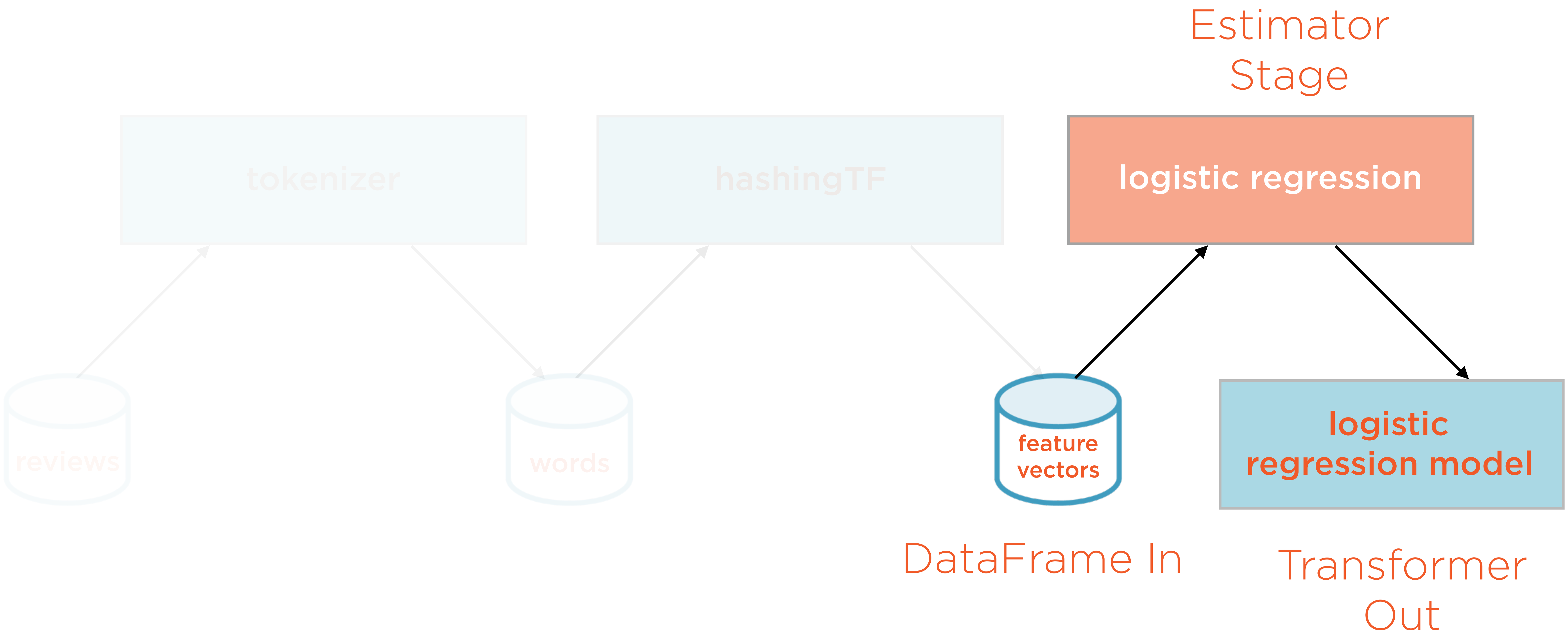
Training Pipeline



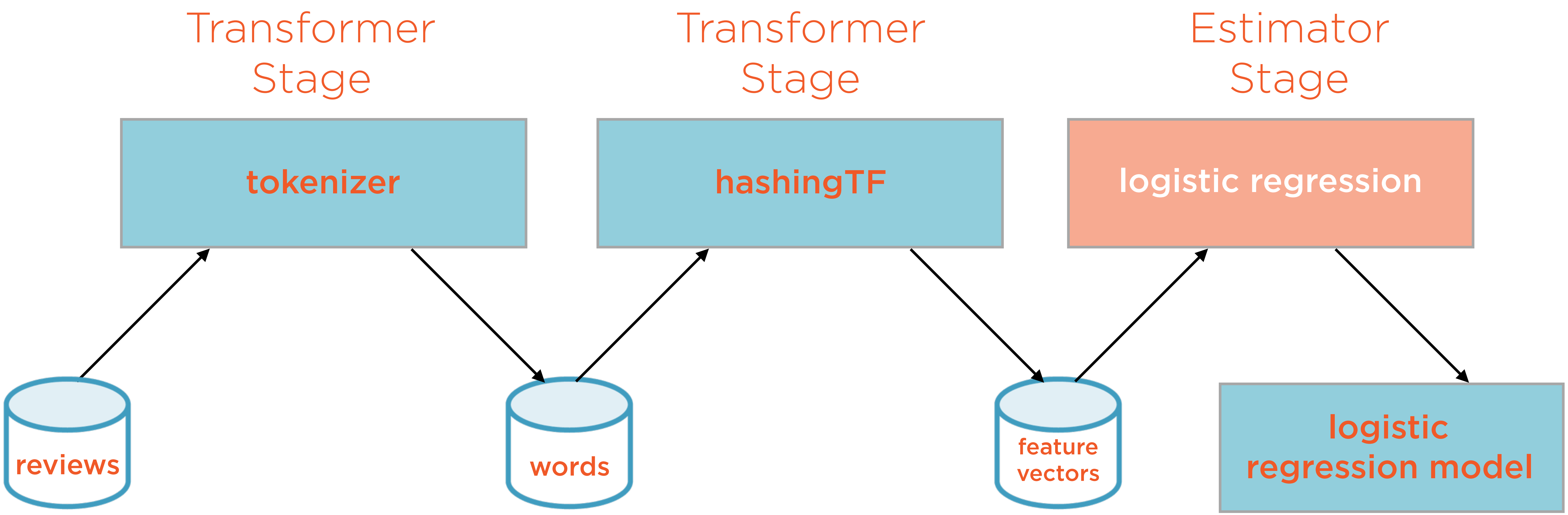
Training Pipeline



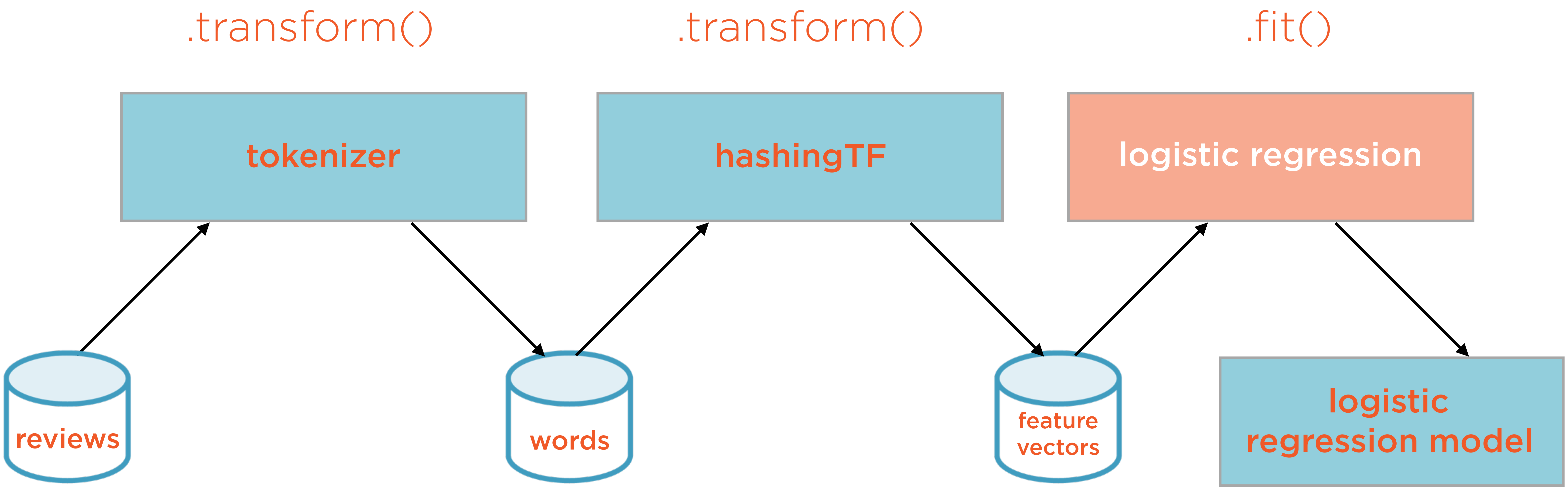
Training Pipeline



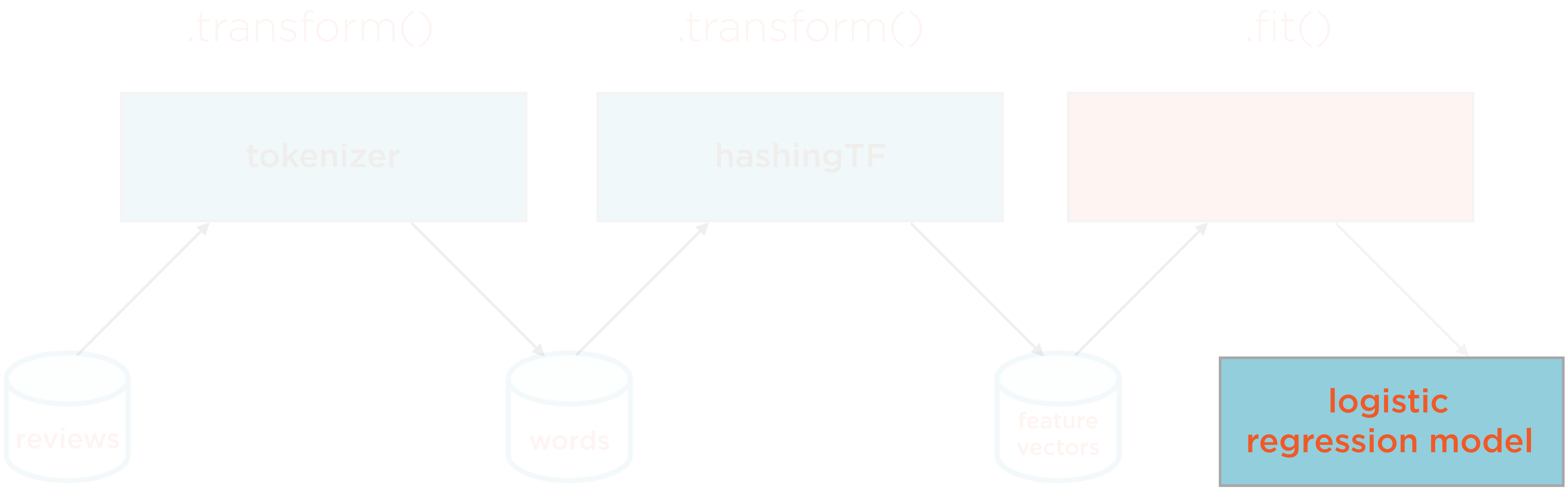
Training Pipeline



Training Pipeline



Training Pipeline



Transformer - use in prediction

ML Pipeline for Sentiment Analysis

Split documents into words

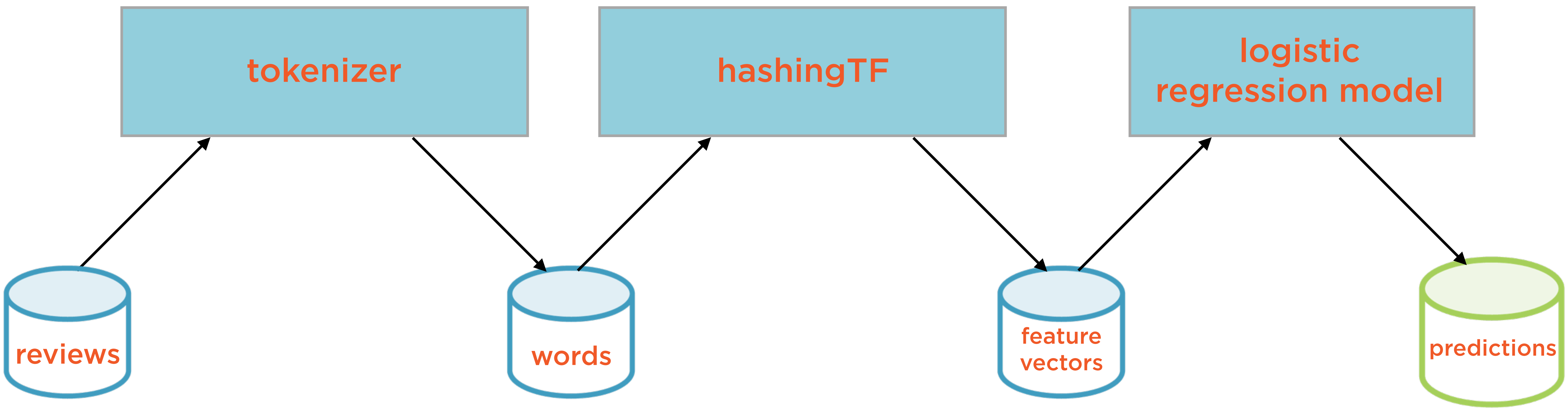
Create feature vectors

Train model using corpus

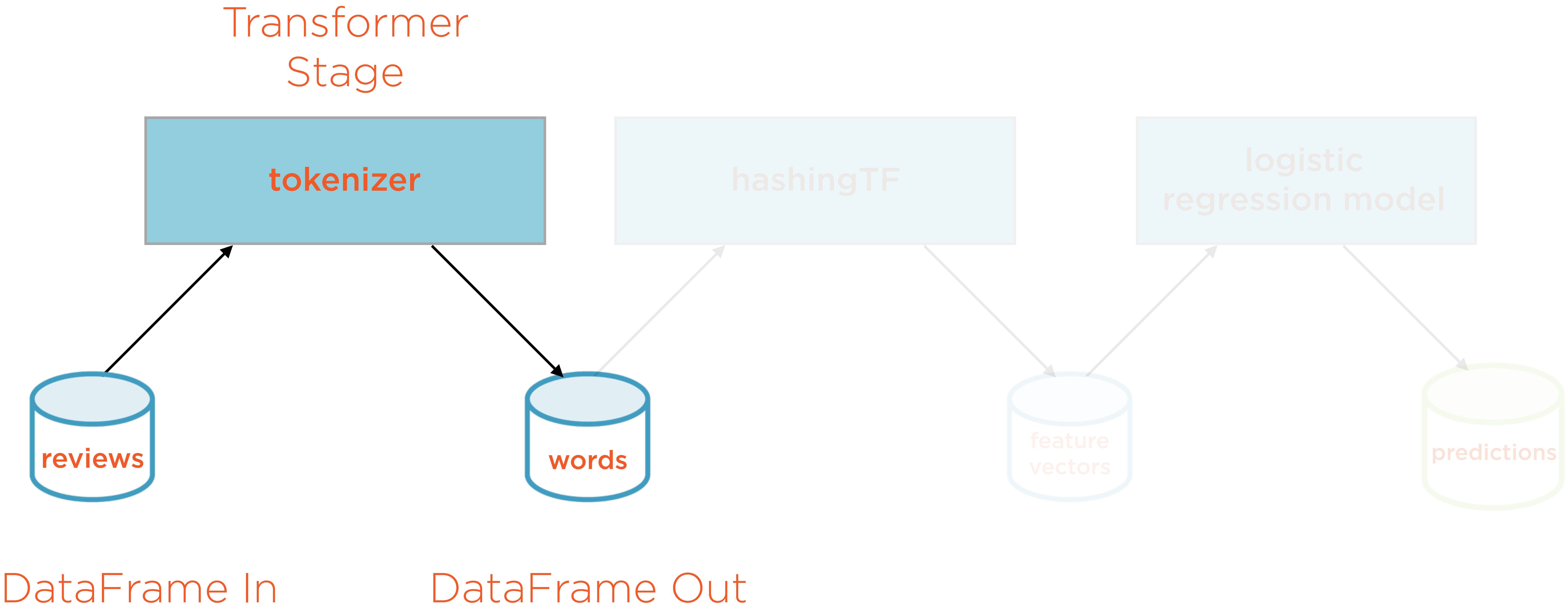
Use trained model for prediction

Use of pipeline during prediction

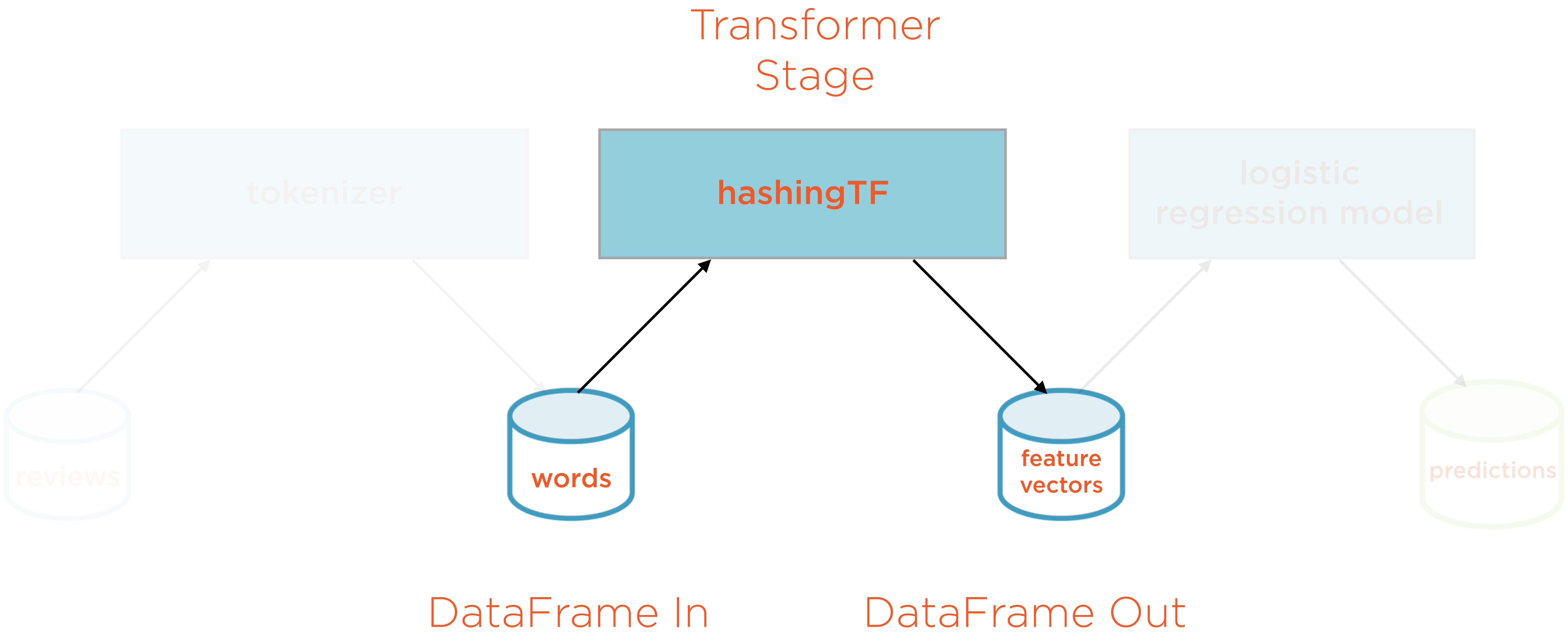
Prediction Pipeline



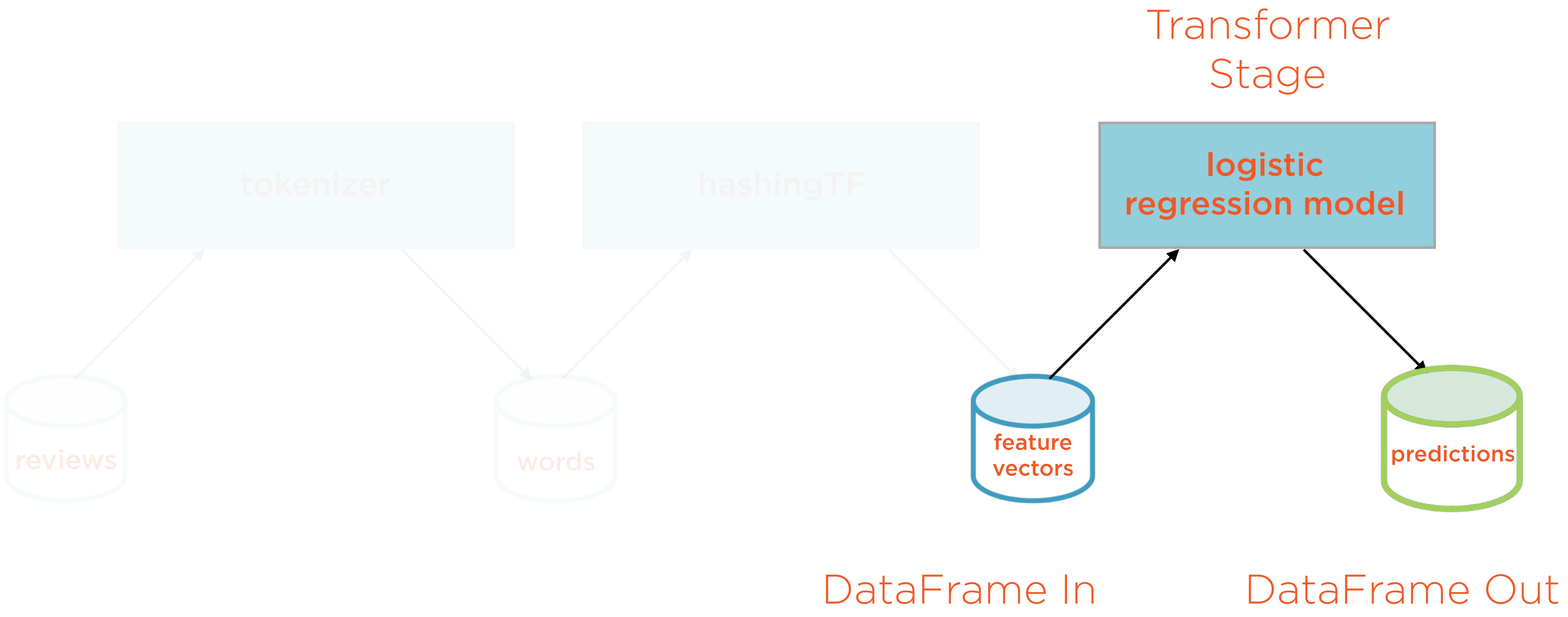
Prediction Pipeline



Prediction Pipeline



Prediction Pipeline



ML Pipeline for Sentiment Analysis

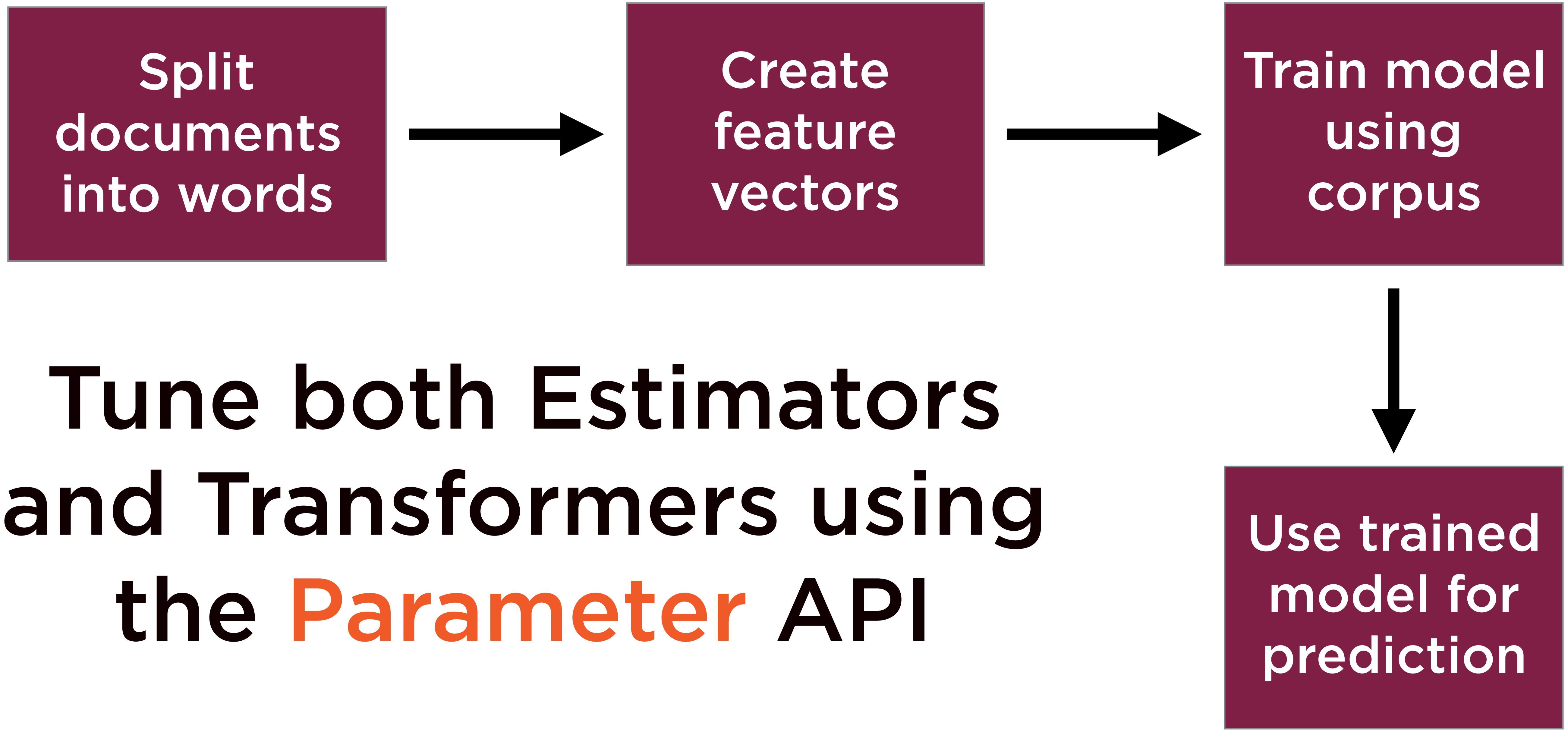
Split documents into words → Create feature vectors → Train model using corpus

↓

Use trained model for prediction

Use of pipeline during **prediction**

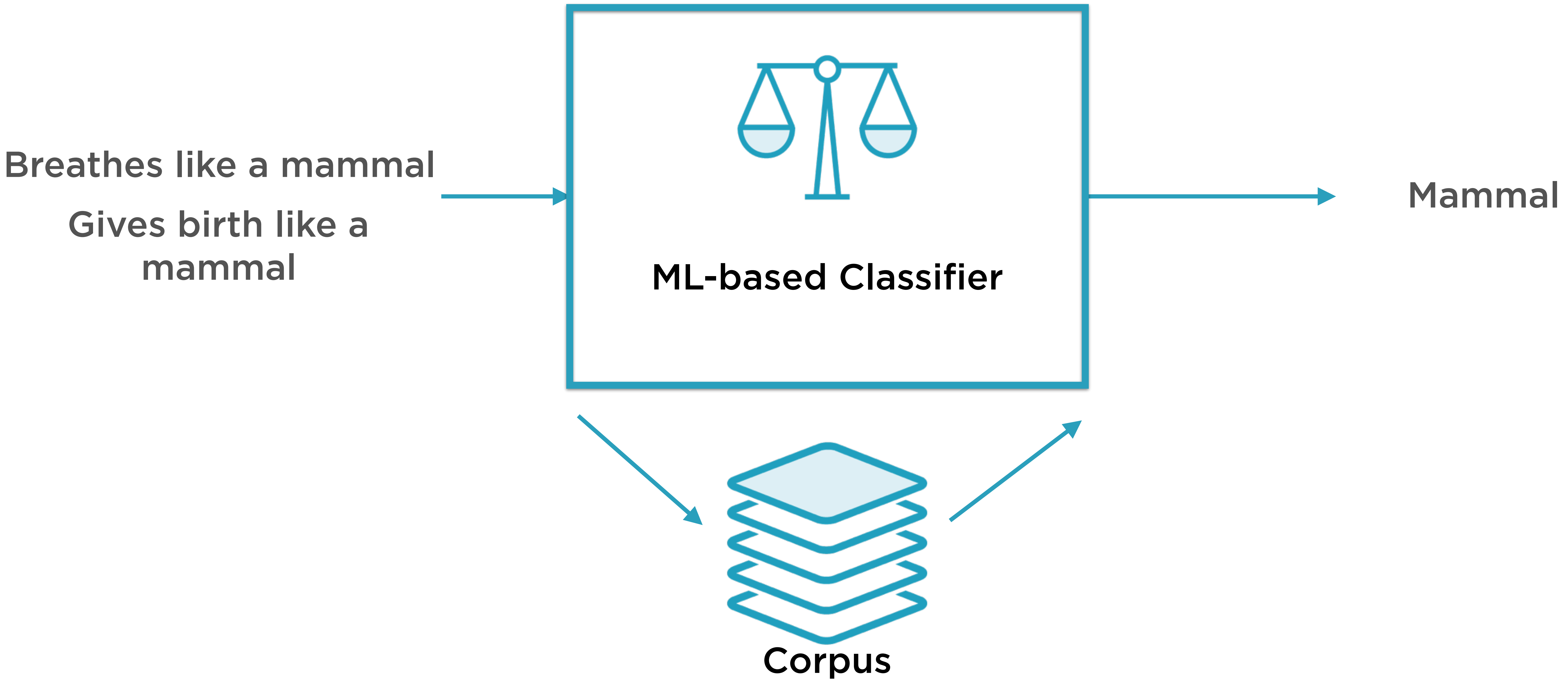
ML Pipeline for Sentiment Analysis



Tune both Estimators and Transformers using the Parameter API

Feature Engineering in Spark

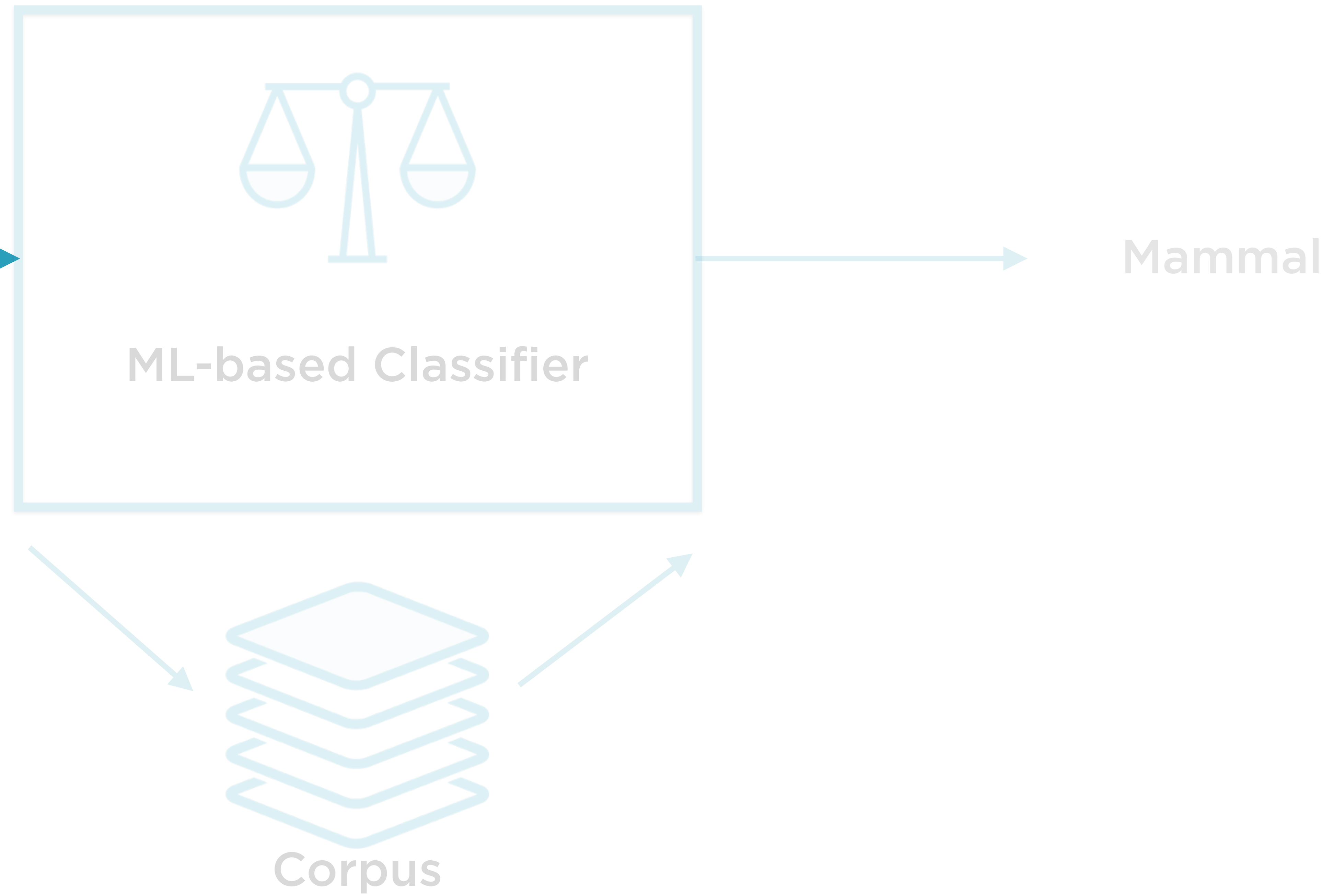
ML-based Binary Classifier



ML-based Binary Classifier

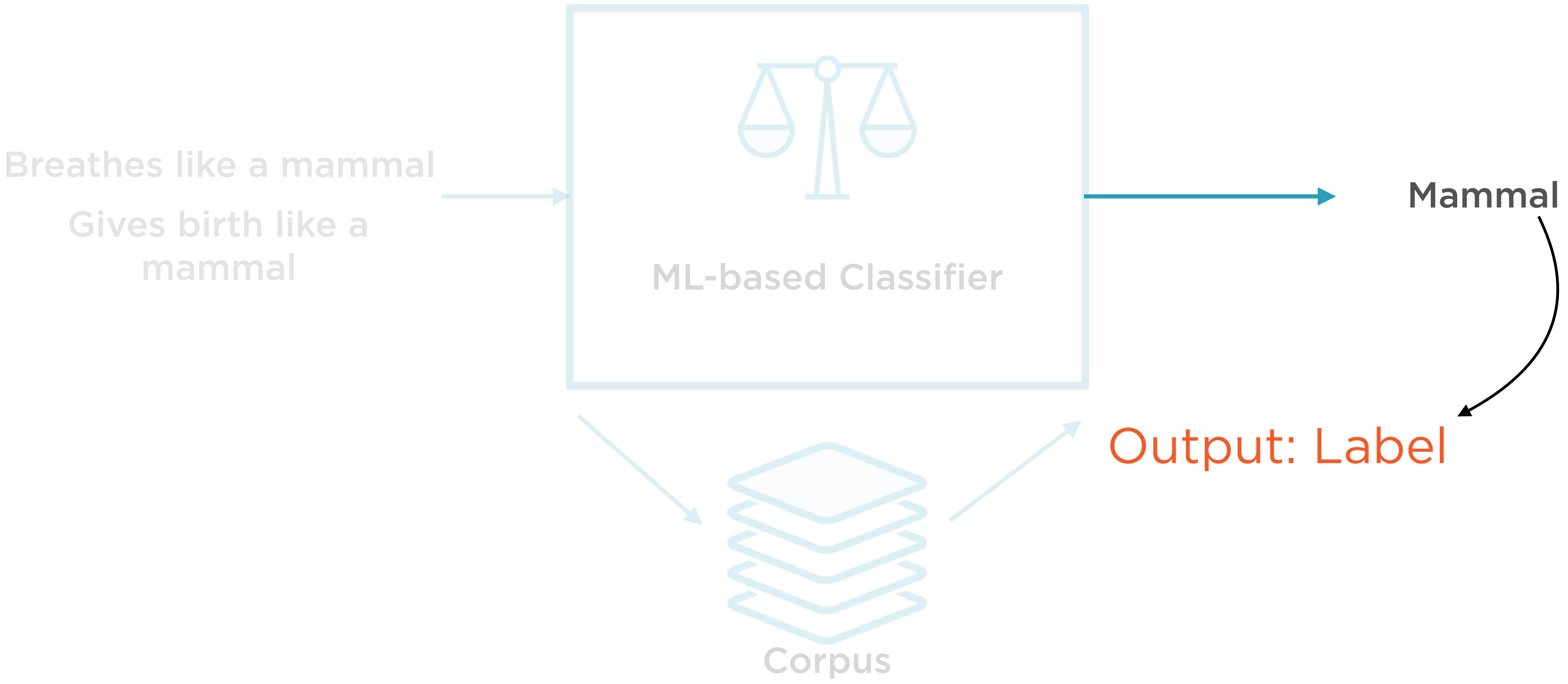
Breathes like a mammal

Gives birth like a mammal

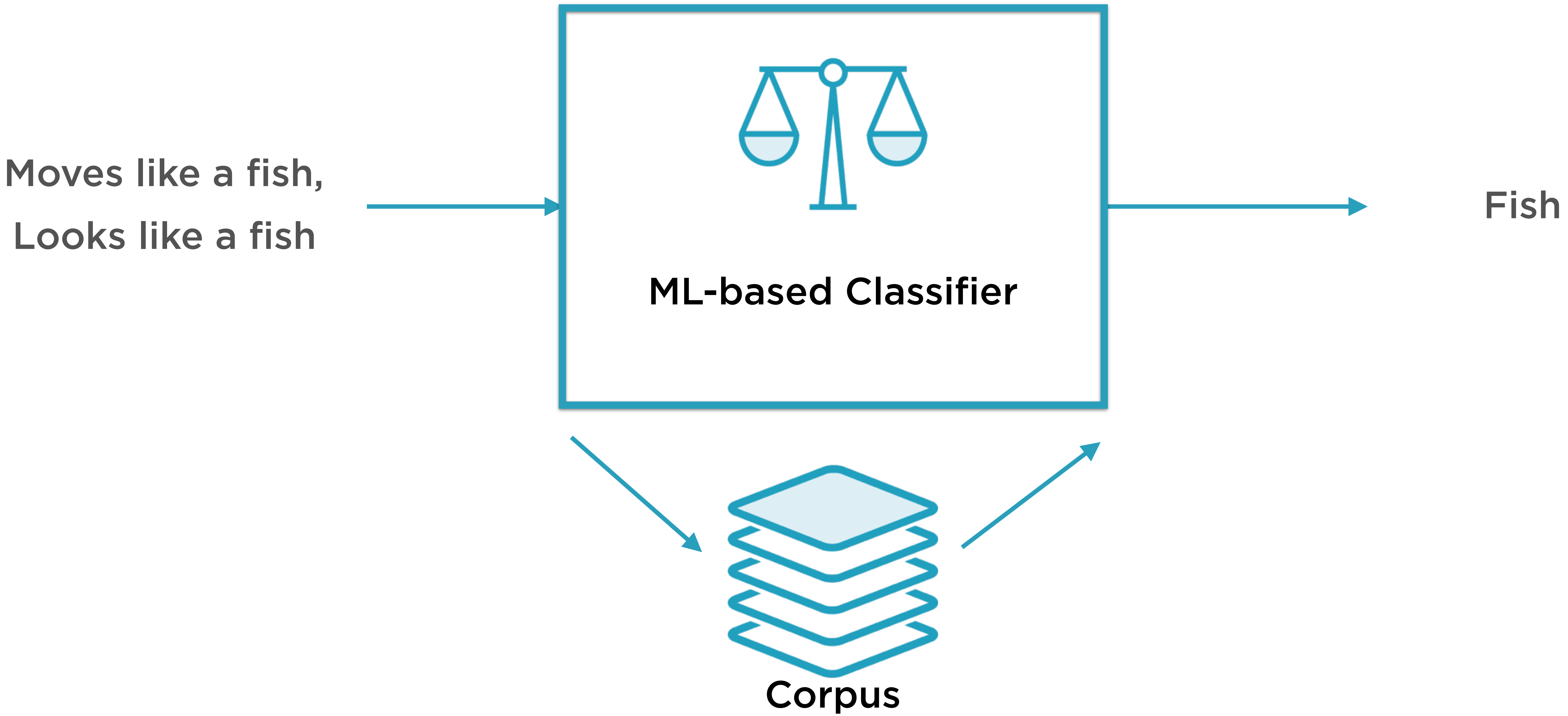


Input: Feature Vector

ML-based Binary Classifier



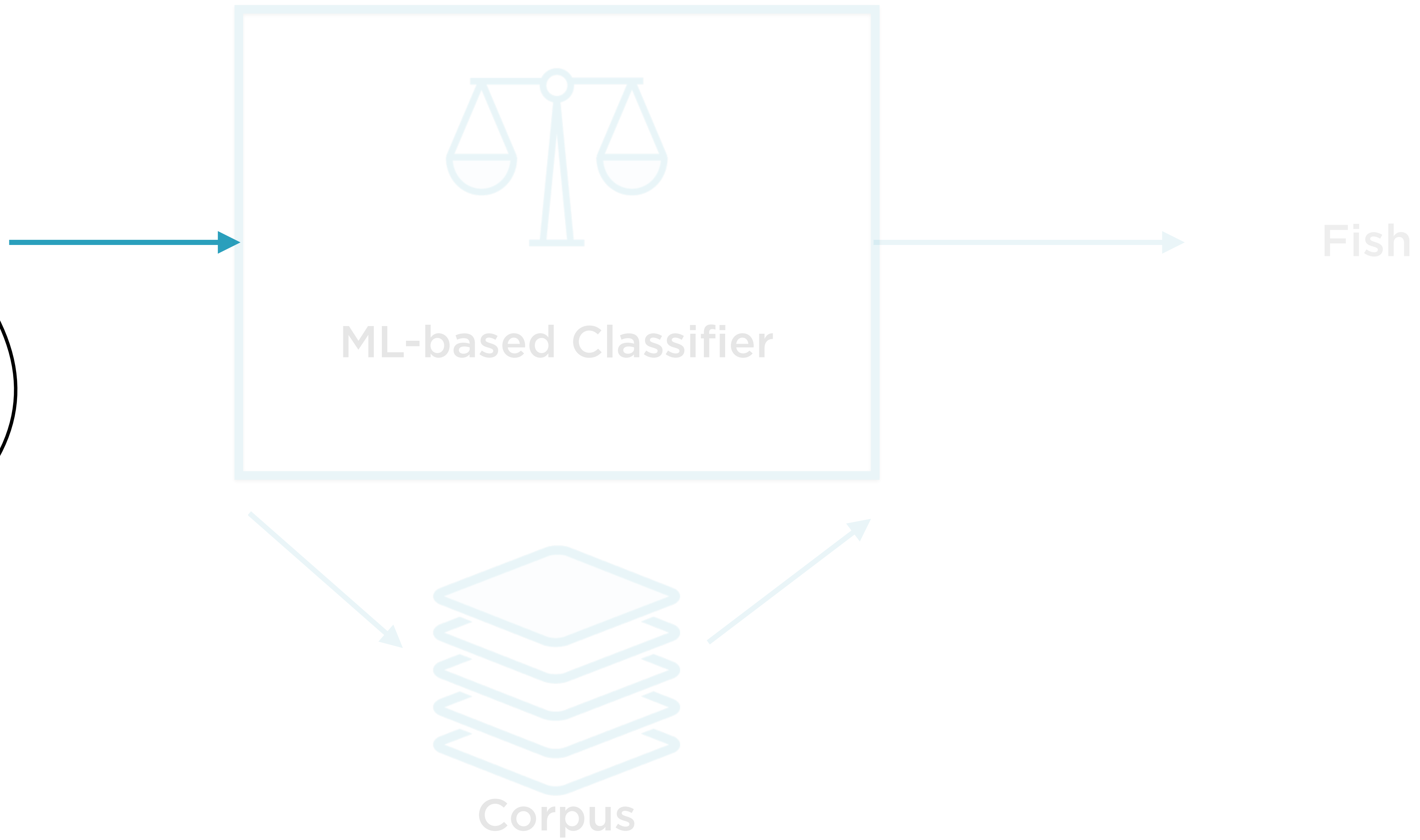
ML-based Binary Classifier



ML-based Binary Classifier

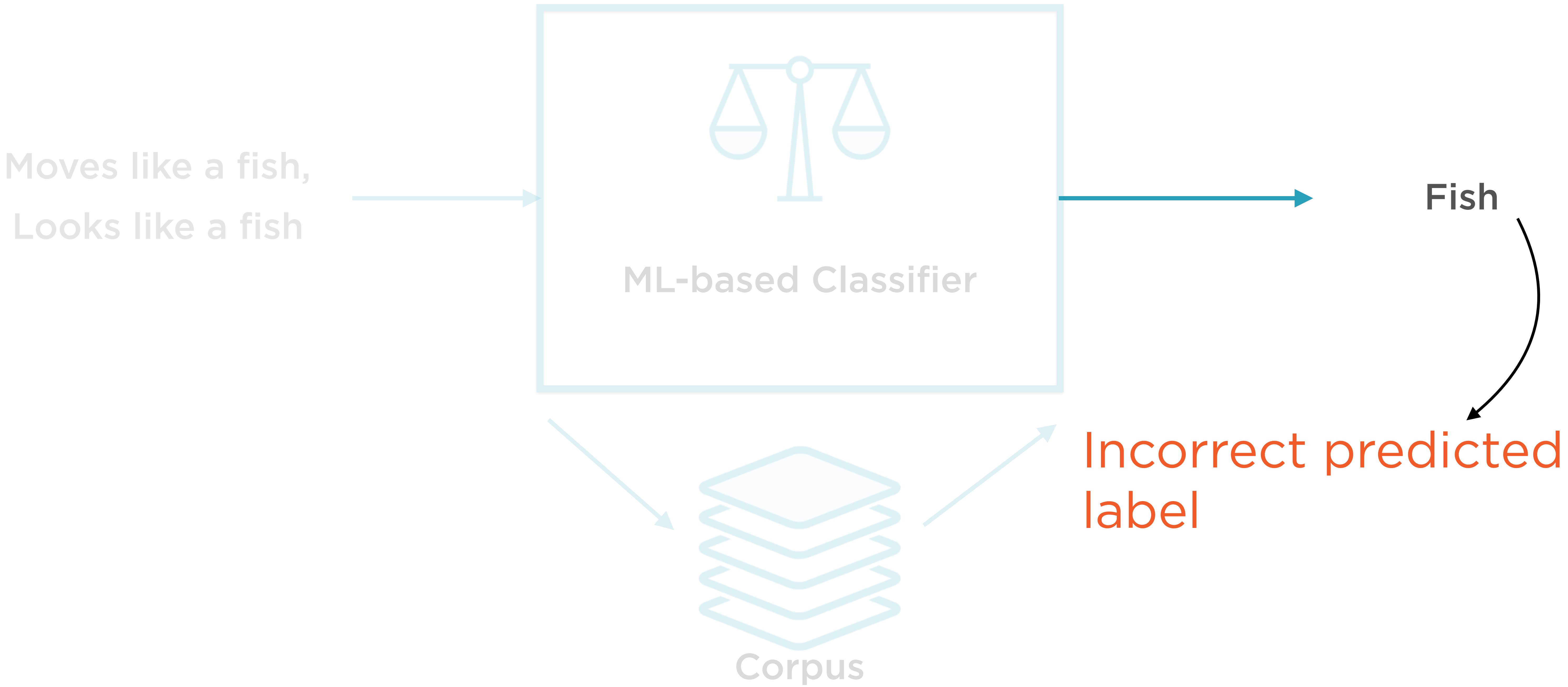
Moves like a fish,

Looks like a fish



Poor choice of features

ML-based Binary Classifier



```
y = f(x)
```

Machine Learning

Most machine learning algorithms seek to “learn” the function f that links the features and the labels

Feature Engineering

Manually designing the input x variables so that a machine learning algorithm can "learn" more effectively

"Spot the Millionaire"

Raw Feature

Latitude and longitude of address

Label

Binary classification - millionaire or not?

Raw feature too granular to hold much predictive power

"Spot the Millionaire"

Engineered Feature

Apartment complex of address of individual

Label

Binary classification - millionaire or not?

Engineered feature transforms raw features

"Predict Happy Hour"

Two Independent Features

Time of day, Day of week

Label

Happy Hour or not?

Independent features need to be combined to improve predictive power

“Predict Happy Hour”

One Crossed Feature

(Time of day, Day of week)

Label

Happy Hour or not?

Feature engineering to ensure individual features always considered together

Feature Engineering in Spark

Feature Extractors

Feature Transformers

Feature Selectors

Locality Sensitive Hashing

Feature Engineering in Spark

Feature Extractors

Feature Transformers

Feature Selectors

Locality Sensitive Hashing

TF-IDF

Word2Vec

CountVectorizer

...

Feature Extractors create features from ‘raw’ data

```
d = “This is not the worst restaurant in the metropolis,
not by a long way”
```

Document as Word Sequence

Model a document as an ordered sequence of words

```
d = “This is not the worst restaurant in the metropolis,
not by a long way”

(“This”, “is”,”not”,”the”,”worst”,”restaurant”,”in”,”the”,
“metropolis”, “not”,”by”,”a”,”long”,”way”)
```

Document as Word Sequence

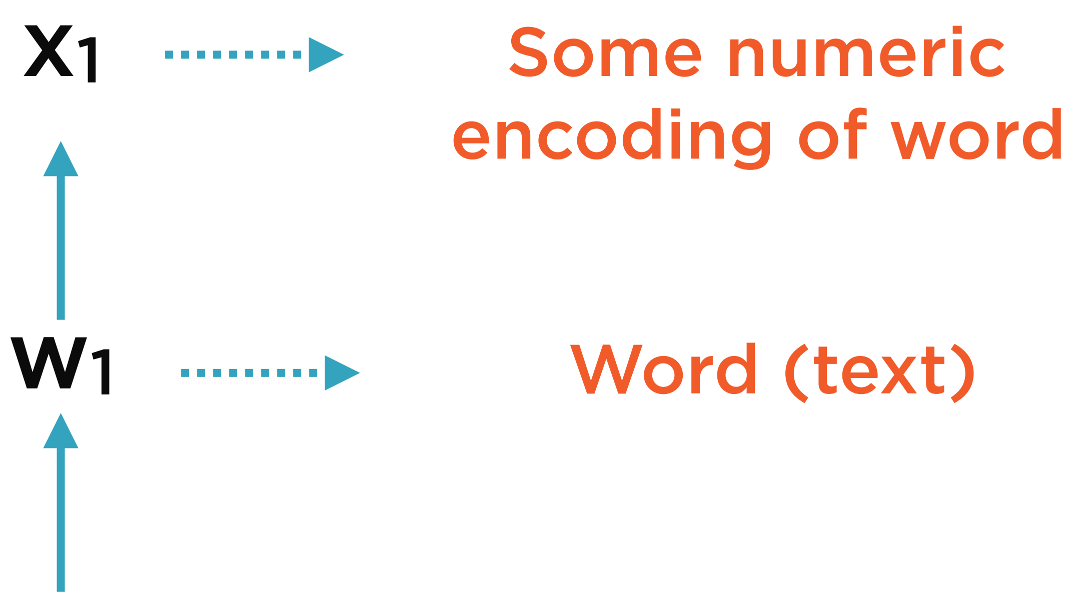
Tokenize document into individual words

x_0 → Some numeric encoding of word

w_0 → Word (text)

("This", "is","not","the","worst","restaurant","in","the", "metropolis", "not","by","a","long","way")

Represent Each Word as a Number



("This", "is","not","the","worst","restaurant","in","the", "metropolis", "not","by","a","long","way")

Represent Each Word as a Number

$$d = [x_0,\ x_1,\ \ldots\ x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor

```
d = [[?], [?], … [?]]
```

The Big Question

How best can words be represented as numeric data?

Word embeddings are used to represent text in numeric form to feed into ML algorithms

Frequency-based Embeddings

Count

TF-IDF

Frequency-based Embeddings

Count

TF-IDF

Count Vector Encoding

Reviews
The movie was bad
The actors were bad, sets were bad

All Words
the
movie
was
bad
actors
were
sets

Create a set of all words (all across the corpus)

Count Vector Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
the	1	1
movie	1	1
was	1	1
bad	1	2
actors	0	1
were	0	1
sets	0	1

Express each review as a frequency of the words which appear in that review

Flaws of Count Vectors

Large vocabulary - enormous feature vectors

Unordered - lost all context

Semantics and word relationships lost

Flaws of Count Vectors

Large vocabulary - enormous feature vectors

Unordered - lost all context

Semantics and word relationships lost

Hash words to buckets to have a fixed vocabulary size

Choose enough buckets so that collisions are rare

Hash Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
the	1	1
movie	1	0
was	1	0
bad	1	2
$\boxed{\text{actors}}$	0	1
were	0	2
$\boxed{\text{sets}}$	0	1

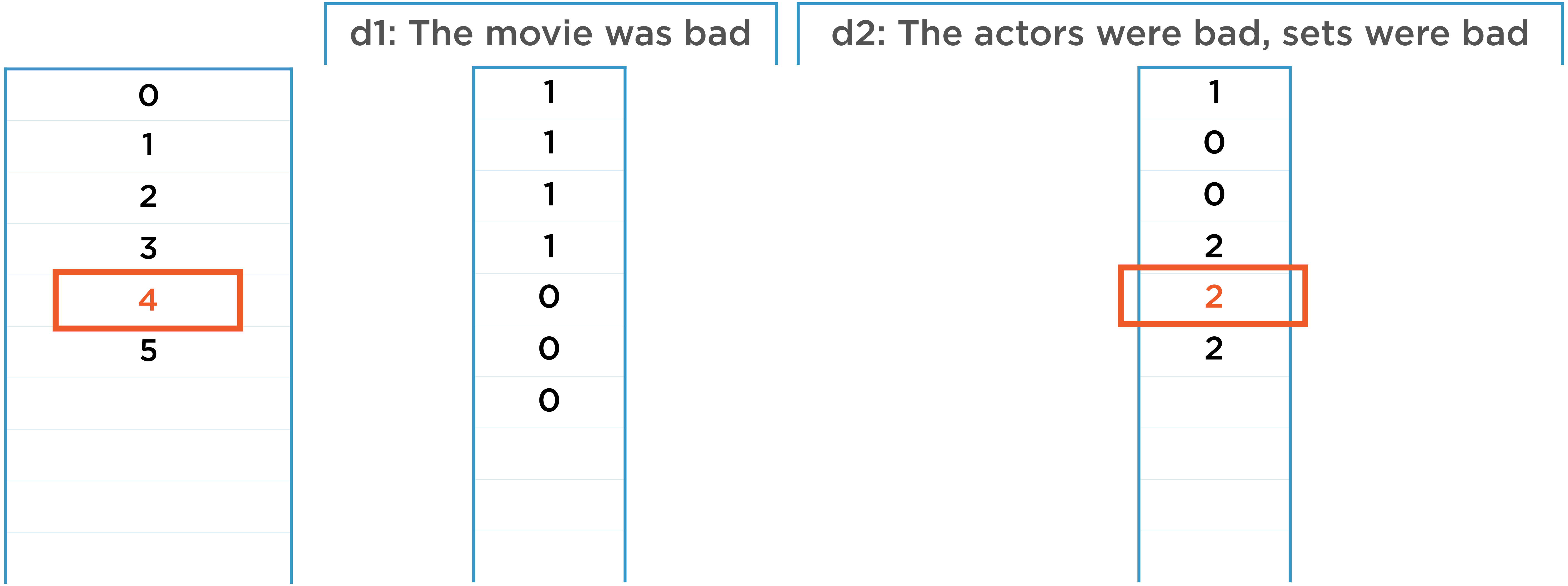
Suppose the words “actors” and “sets” hashed to the same bucket (represented by an integer)

Hash Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
0	1	1
1	1	0
2	1	0
3	1	2
4	0	1
5	0	2
4	0	1

Suppose the words “actors” and “sets” hashed to the same bucket (represented by an integer)

Hash Encoding



Suppose the words “actors” and “sets” hashed to the same bucket (represented by an integer)

Frequency-based Embeddings

Count

TF-IDF

Captures how often a word occurs in a **document** as well as the **entire corpus**

$$d = [x_0,\ x_1,\ \dots\ x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor

$$x_i = tf(w_i) \times idf(w_i)$$

Tf-Idf

Tf = Term Frequency; Idf = Inverse Document Frequency

Tf-Idf

Frequently in a single document

Might be important

Frequently in the corpus

Probably a common word like "a", "an", "the"

Evaluating Tf-Idf

Important advantages

- Feature vector much more tractable in size
- Frequency and relevance captured

One big drawback

- Context still not captured

Feature Engineering in Spark

Feature Extractors

Feature Transformers

Feature Selectors

Locality Sensitive Hashing

Tokenizer

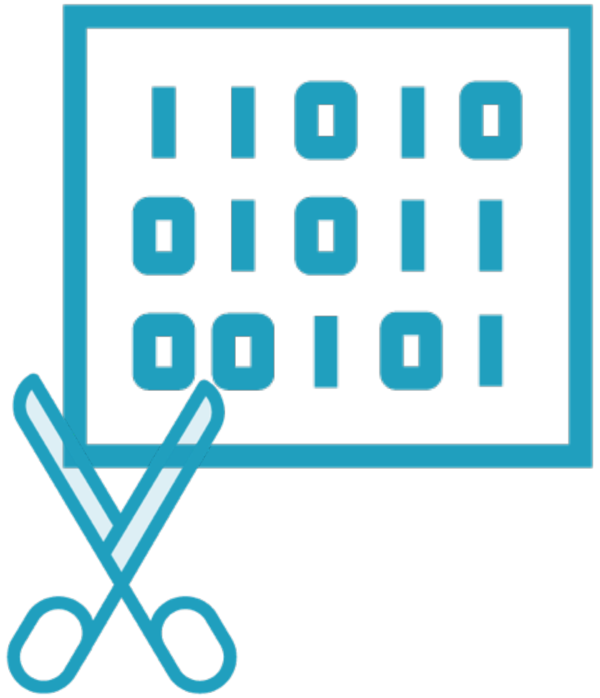
PCA

StandardScaler

OneHotEncoder

...

Feature Transformers refine existing features



Split text fragment into words

Tokenizer

Split documents into paragraphs

Split paragraphs into sentences

Split sentences into words

Simple library functions available

One-hot Encoding

Continuous data can be ordered, categorical data can not

ML algorithms only operate on numbers

Categorical data need to be encoded as numbers

Numerical encodings of categorical data should never be ordered

One-hot Encoding

Continuous data can be ordered, categorical data can not

ML algorithms only operate on numbers

Categorical data need to be encoded as numbers

Numerical encodings of categorical data should never be ordered

One-hot Encoding

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

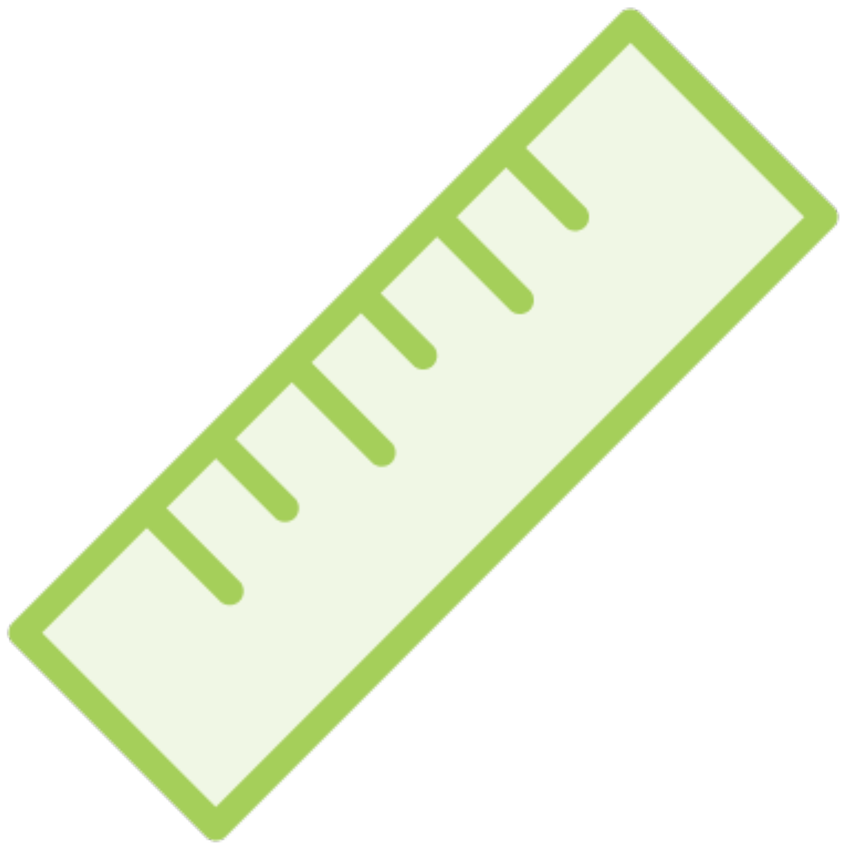
Saturday

One-hot Encoding

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Monday	0	1	0	0	0	0	0
Thursday	0	0	0	0	1	0	0
Saturday	0	0	0	0	0	0	1

One-hot Encoding

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Monday	0	1	0	0	0	0	0
Thursday	0	0	0	0	1	0	0
Saturday	0	0	0	0	0	0	1



Standardized Data

Many techniques work best on standardized data

Standardization prevents some (high-variance) data series from dominating

Examples:

- Principal Components Analysis
- Lasso/Ridge Regression

Standardizing Data

$$
\begin{bmatrix}
\boxed{\begin{matrix} x_{11} \\ x_{21} \\ x_{31} \\ \dots \\ x_{n1} \end{matrix}} & \dots & \boxed{\begin{matrix} x_{1k} \\ x_{2k} \\ x_{3k} \\ \dots \\ x_{nk} \end{matrix}}
\end{bmatrix}
$$

$$
\begin{array}{ccc}
\mathrm{avg}(X_1) & \dots & \mathrm{avg}(X_k) \\
\mathrm{stdev}(X_1) & \dots & \mathrm{stdev}(X_k)
\end{array}
$$

Standardizing Data

$$\begin{bmatrix} \dfrac{x_{11} - \text{avg}(X_1)}{\text{stdev}(X_1)} & \cdots & \dfrac{x_{1k} - \text{avg}(X_k)}{\text{stdev}(X_k)} \\ \cdots & & \cdots \\ \dfrac{x_{n1} - \text{avg}(X_1)}{\text{stdev}(X_1)} & & \dfrac{x_{nk} - \text{avg}(X_k)}{\text{stdev}(X_k)} \end{bmatrix}$$

Each column of the standardized data has mean 0 and variance 1

Principal Components Analysis

A technique to re-express complex data in terms of a few, well-chosen vectors (Principal Components) that most efficiently capture the variation in that data

Feature Engineering in Spark

Feature Extractors

Feature Transformers

Feature Selectors

Locality Sensitive Hashing

VectorSlicer

RFormula

ChiSqSelector

Feature Selectors choose a subset of features

VectorSlicer

DATE	OPEN	…	PRICE
2016-12-01	772	…	779
2016-11-01	758	…	747
2006-01-01	302	…	309

Select

[“Date”, “Price”]

DATE	PRICE
2016-12-01	779
2016-11-01	747
2006-01-01	309

Feature Engineering in Spark

Feature Extractors

Feature Transformers

Feature Selectors

Locality Sensitive Hashing

Nearest Neighbor Search

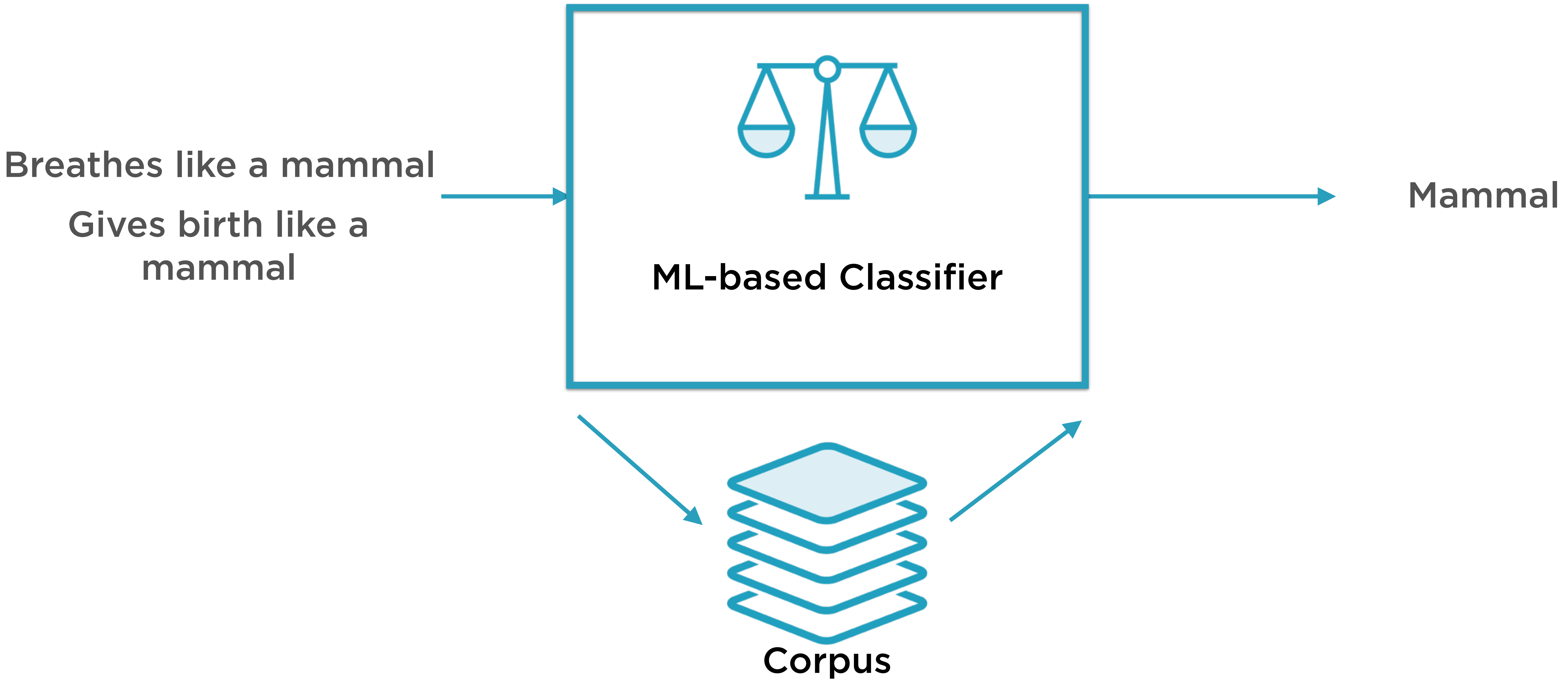
Similarity Join

Used in clustering

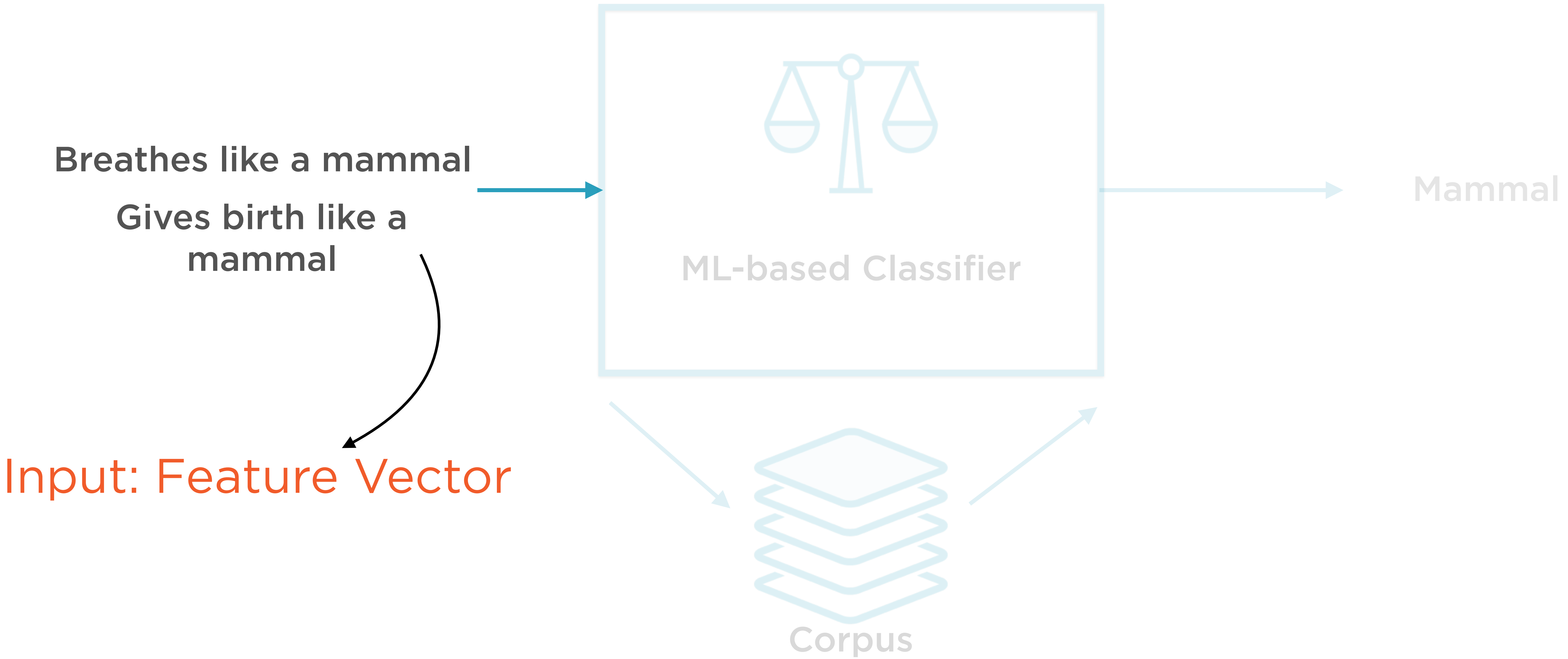
Locality Sensitive Hashing (LSH)

Evaluating Classifiers - The Confusion Matrix

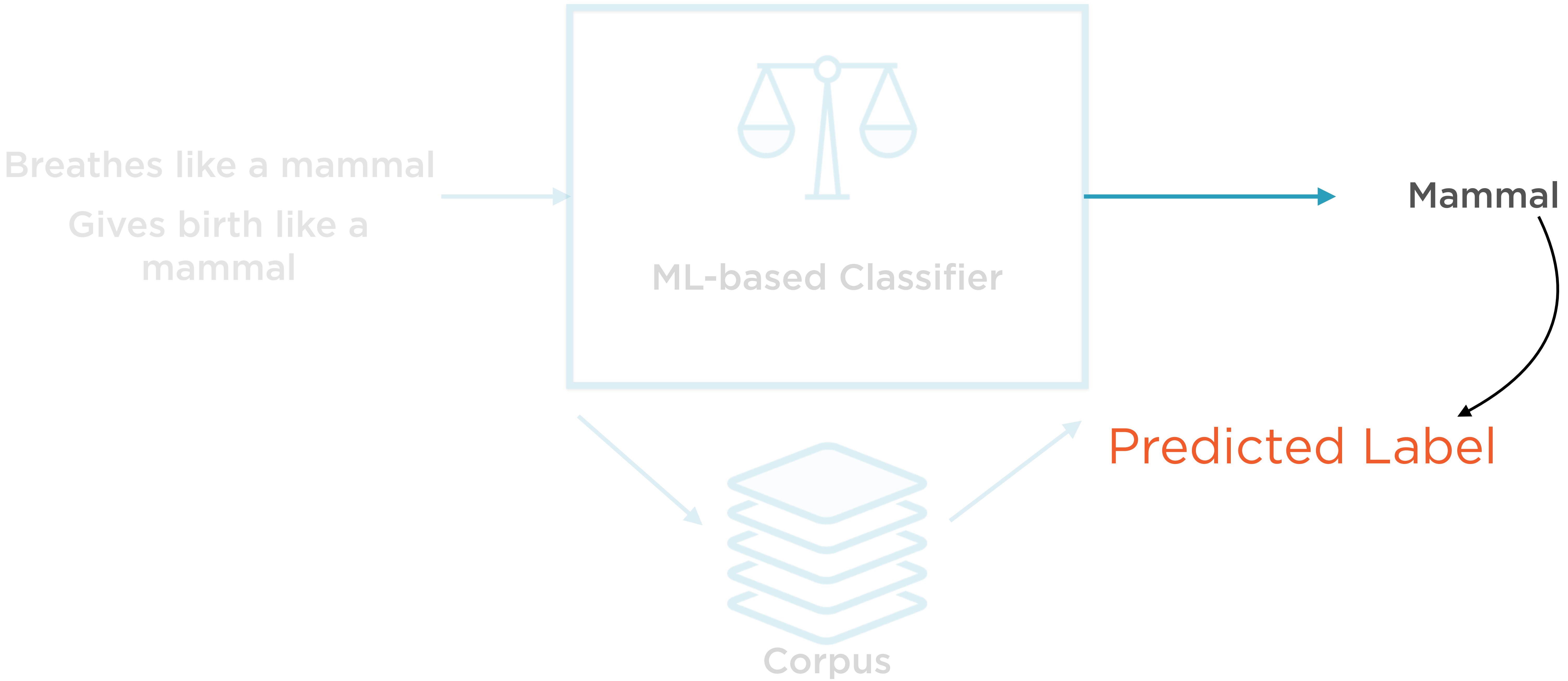
Binary Classifier



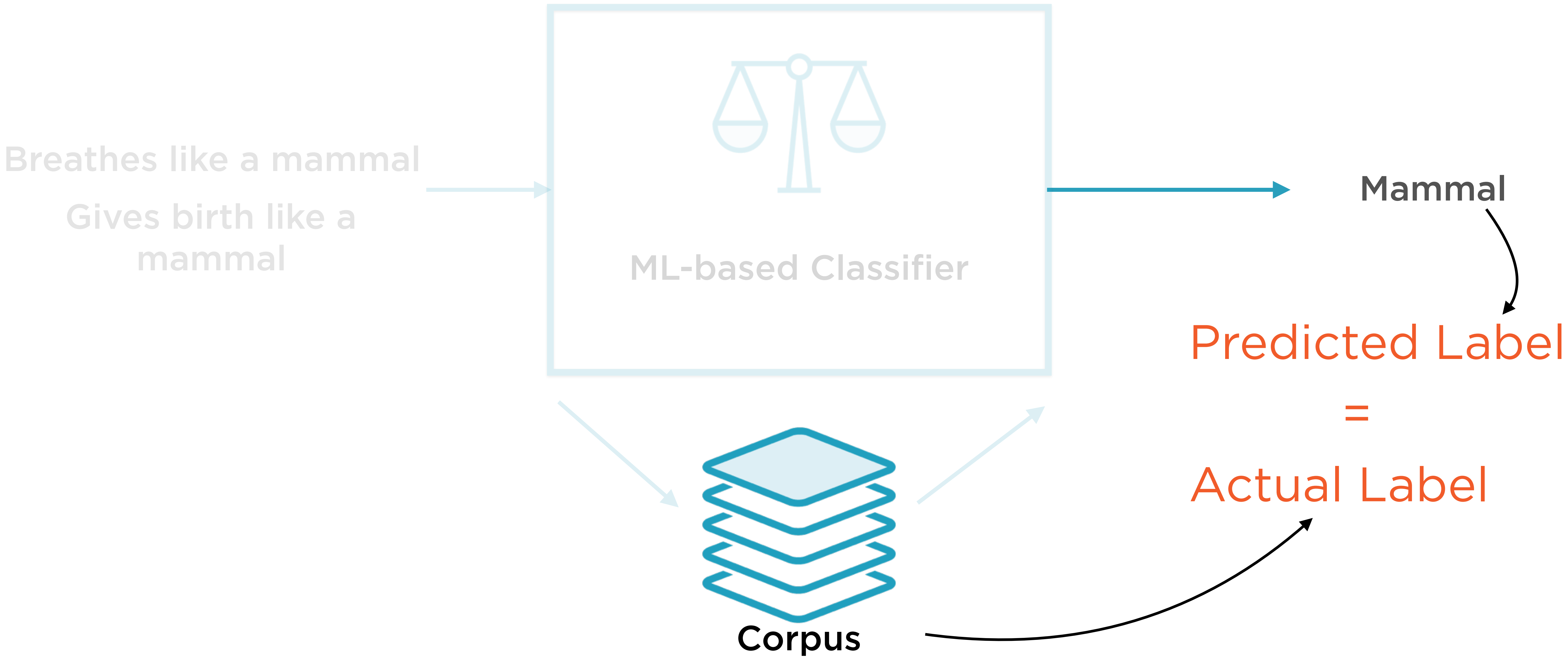
Binary Classifier



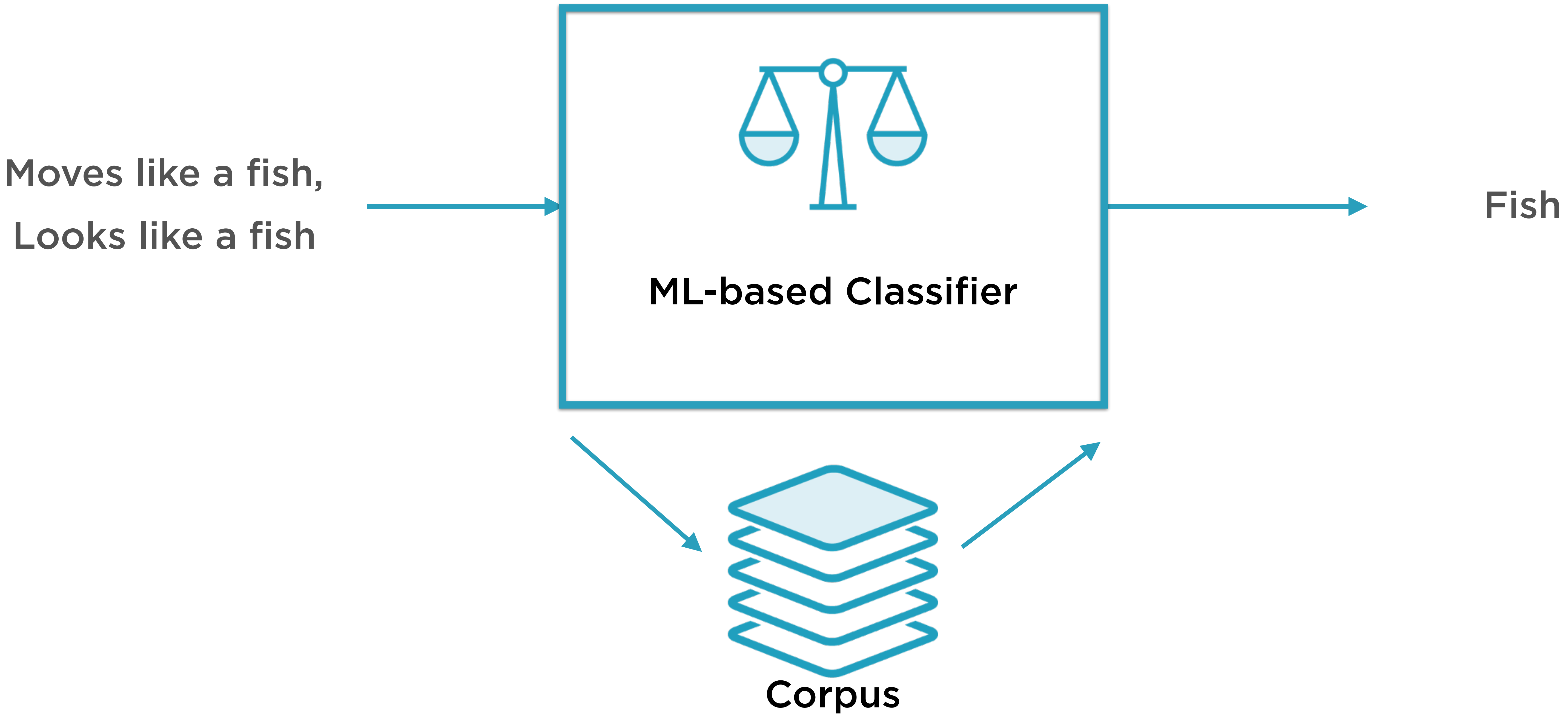
Binary Classifier



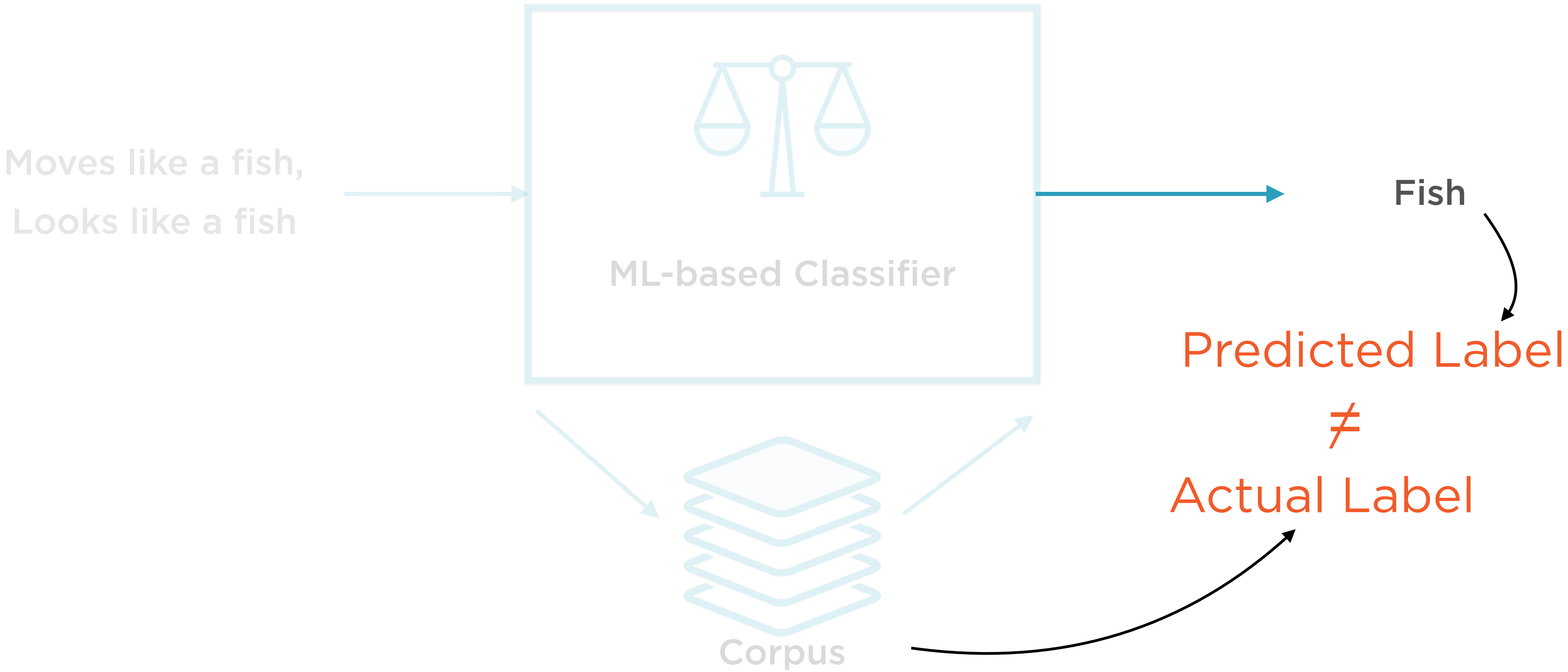
Binary Classifier



Binary Classifier



Binary Classifier



Accuracy

Compare predicted and actual labels

High accuracy is good, but...

All-is-well Binary Classifier

Here, accuracy for rare cancer may be 99.9999%, but…

Accuracy

Some labels maybe much more common/rare than others

Such a dataset is said to be skewed

Accuracy is a poor evaluation metric here

Confusion Matrix

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	**10 instances**	**4 instances**
No Cancer	**5 instances**	**1000 instances**

Confusion Matrix

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	10	4
No Cancer	5	1000

True Positive

Predicted Labels

Actual Label	Cancer	No Cancer
Cancer	TP 10	4
No Cancer	5	1000

Actual Label = Predicted Label

False Positive

Actual Label	Predicted Labels: Cancer	Predicted Labels: No Cancer
Cancer	10	4
No Cancer	FP 5	1000

Actual Label ≠ Predicted Label

True Negative

Predicted Labels

Actual Label	Cancer	No Cancer
Cancer	10	4
No Cancer	5	TN 1000

Actual Label = Predicted Label

False Negative

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	10	FN 4
No Cancer	5	1000

Actual Label ≠ Predicted Label

Confusion Matrix

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Accuracy

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Accuracy

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Actual Label = Predicted Label

Accuracy

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Num Instances}} = \frac{1010}{1019} = 99.12\%$$

Accuracy

Accuracy = 99.12%

Classifier gets it right 99.12% of the time

But...

Precision

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Precision

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Precision = Accuracy when classifier flags cancer

Precision

Predicted Labels

Actual Label	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{10}{15} = 66.67\%$$

Precision

Precision = 66.67%

- **1 in 3 cancer diagnoses is incorrect**

Recall

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Recall

Actual Label \ Predicted Labels	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

Recall = Accuracy when cancer actually present

Recall

Predicted Labels

Actual Label	Cancer	No Cancer
Cancer	TP 10	FN 4
No Cancer	FP 5	TN 1000

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{10}{14} = 71.42\%$$

Recall

Recall = 71.42%

- **2 in 7 cancer cases missed**

"Always Positive"

$P_{threshold} = 0$

Actual Label	Predicted Labels: Cancer	Predicted Labels: No Cancer
Cancer	TP 14	FN 0
No Cancer	FP 1005	TN 0

- Recall = 100%
- Precision = 14/1005 = 13.9%
- Classifier not conservative enough

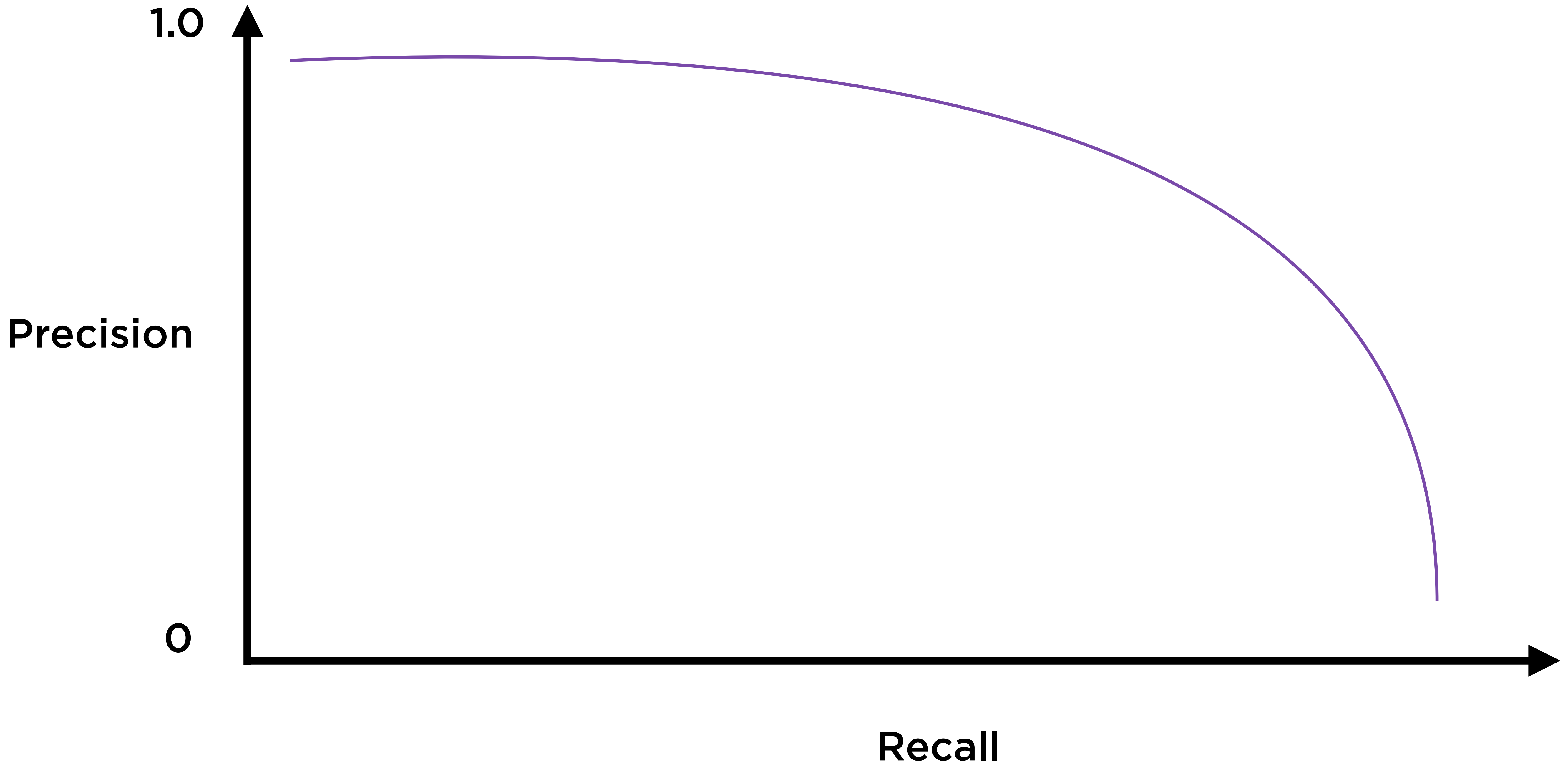
"Always Negative"

$P_{threshold} = 1$

Actual Label	Predicted Labels: Cancer	Predicted Labels: No Cancer
Cancer	TP 14	FN 0
No Cancer	FP 1005	TN 0

- Recall = 0%
- Precision = Infinite
- Classifier too conservative

Precision-Recall Tradeoff



F_1 Score

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Harmonic mean of precision, recall**
- **Closer to lower of two**
- **Favors even tradeoff**

Demo

Implement classification using Decision Trees in spark.ml

Evaluate the model using the F1 score

Random Forests

Jockey or Basketball Player?

Jockeys

Tend to be light to meet horse carrying limits

Basketball Players

Tend to be tall, strong and heavy

Jockey or Basketball Player?

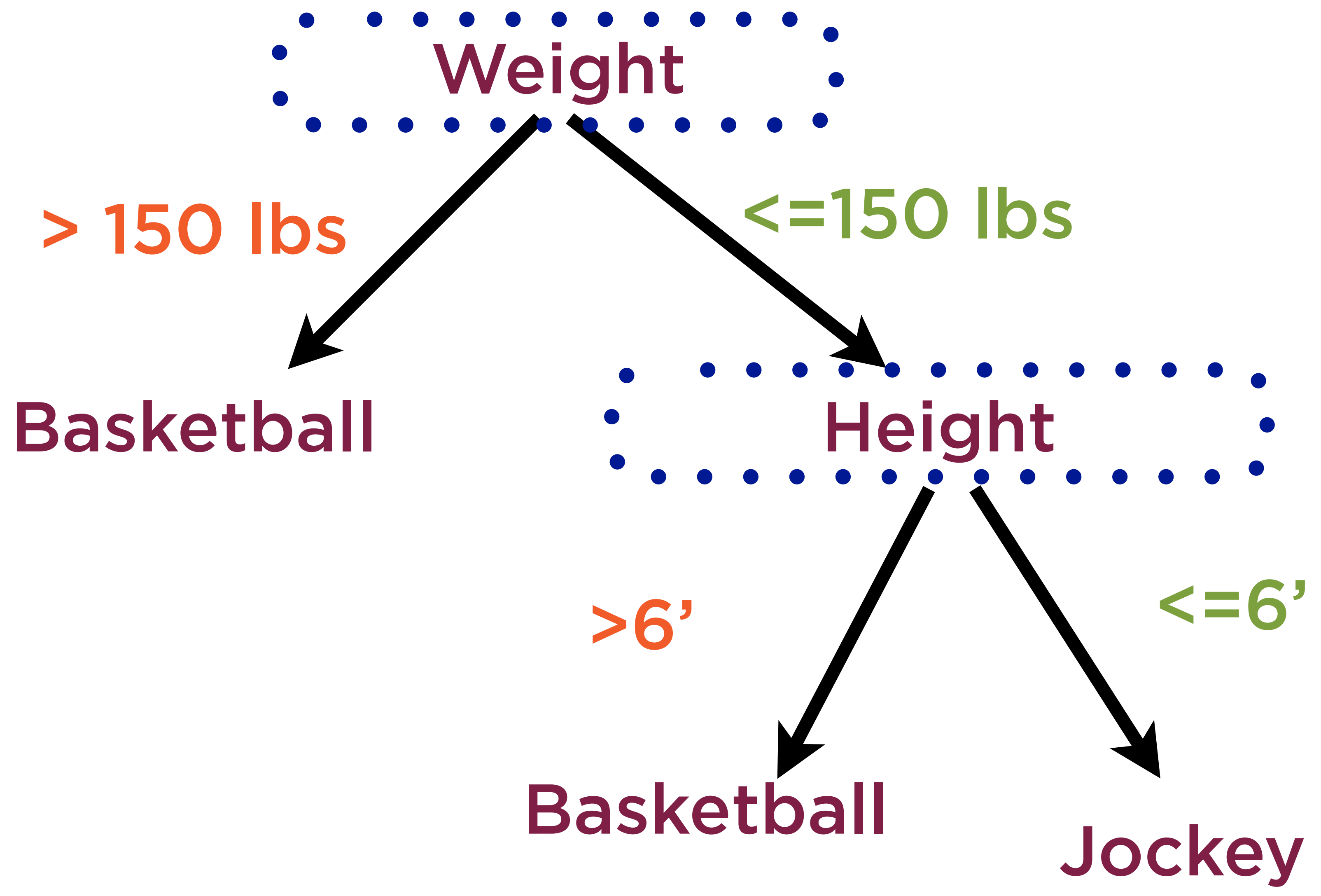
Intuitively know

- jockeys tend to be light
- ...and not very tall
- basketball players tend to be tall
- ...and also quite heavy

Order of decision variables matters

Rules and order found using ML

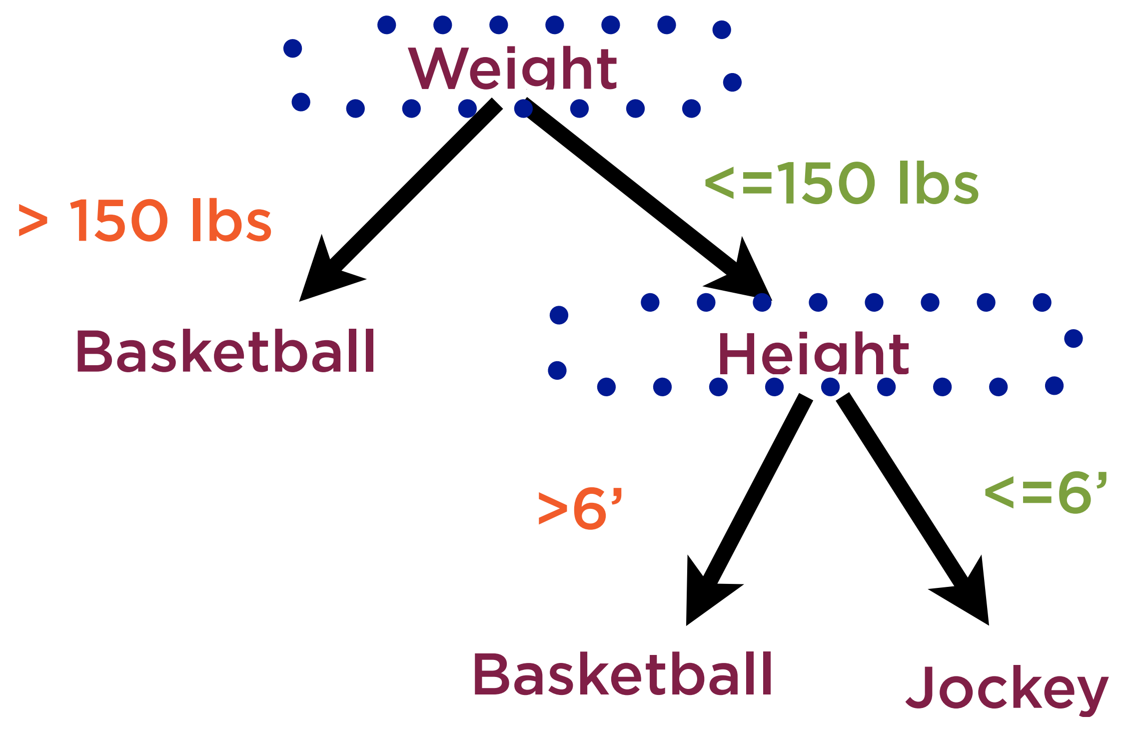
Decision Tree



Decision trees are prone to overfitting on the training data

“If everyone in the room is thinking the same thing, then somebody isn’t thinking.”

General Patton



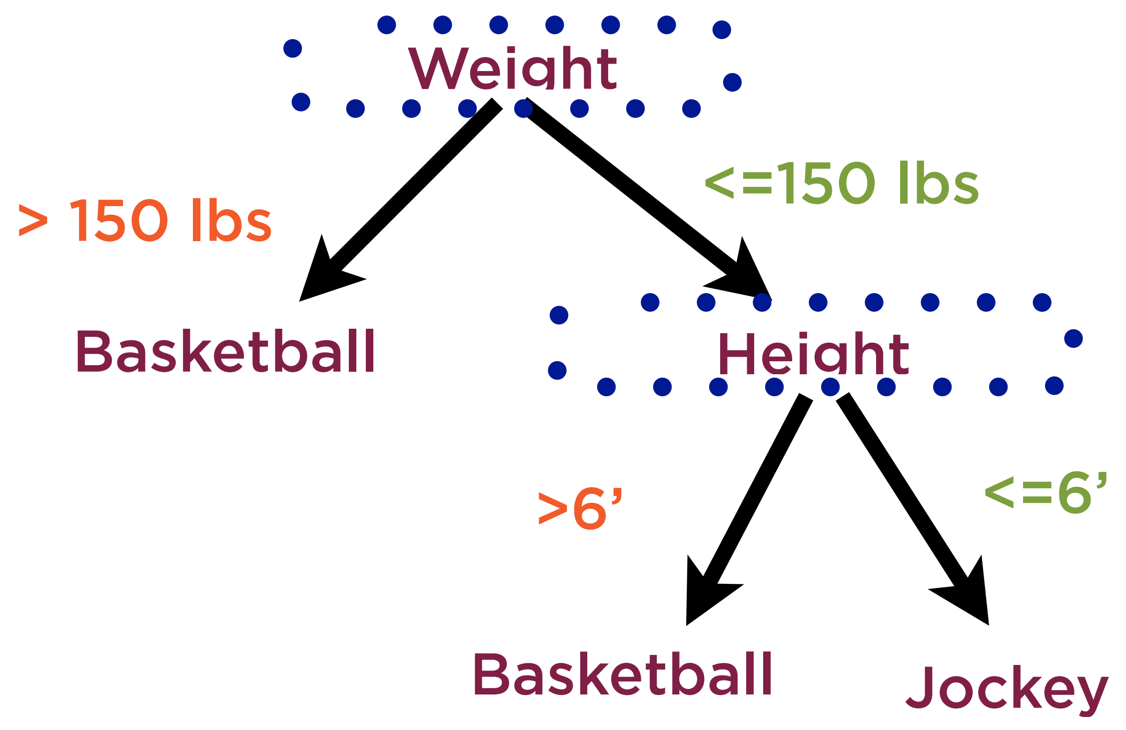
Random Forests

Train many decision trees

- each on random sample of data

Combine their output

- averaging for regression
- mode for classification



Random Forests

Extremely powerful technique

Example of ensemble learning

Individual trees should be as different as possible

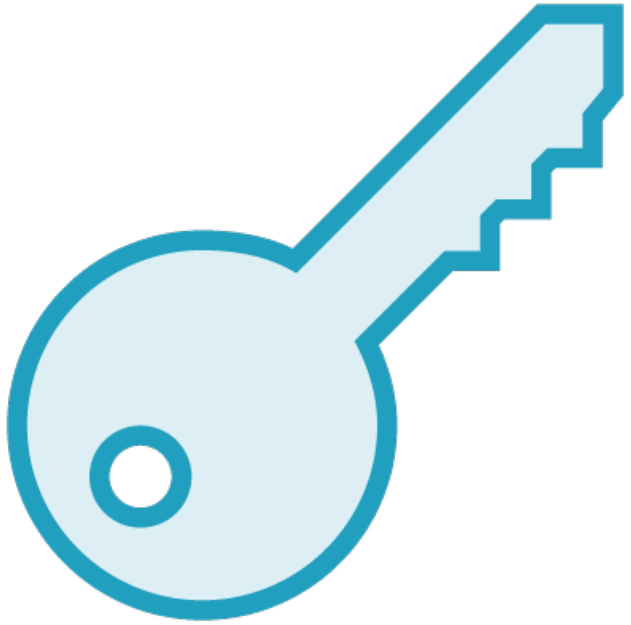
Demo

Implement classification using Random Forests in spark.ml

Build an ML pipeline to chain transformers and estimators

Setting up the Regression Problem

X Causes Y



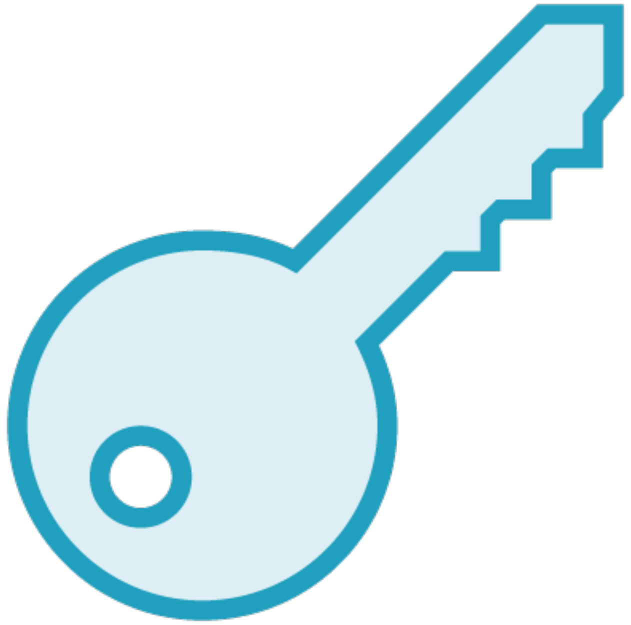
Cause

Independent variable

Effect

Dependent variable

X Causes Y



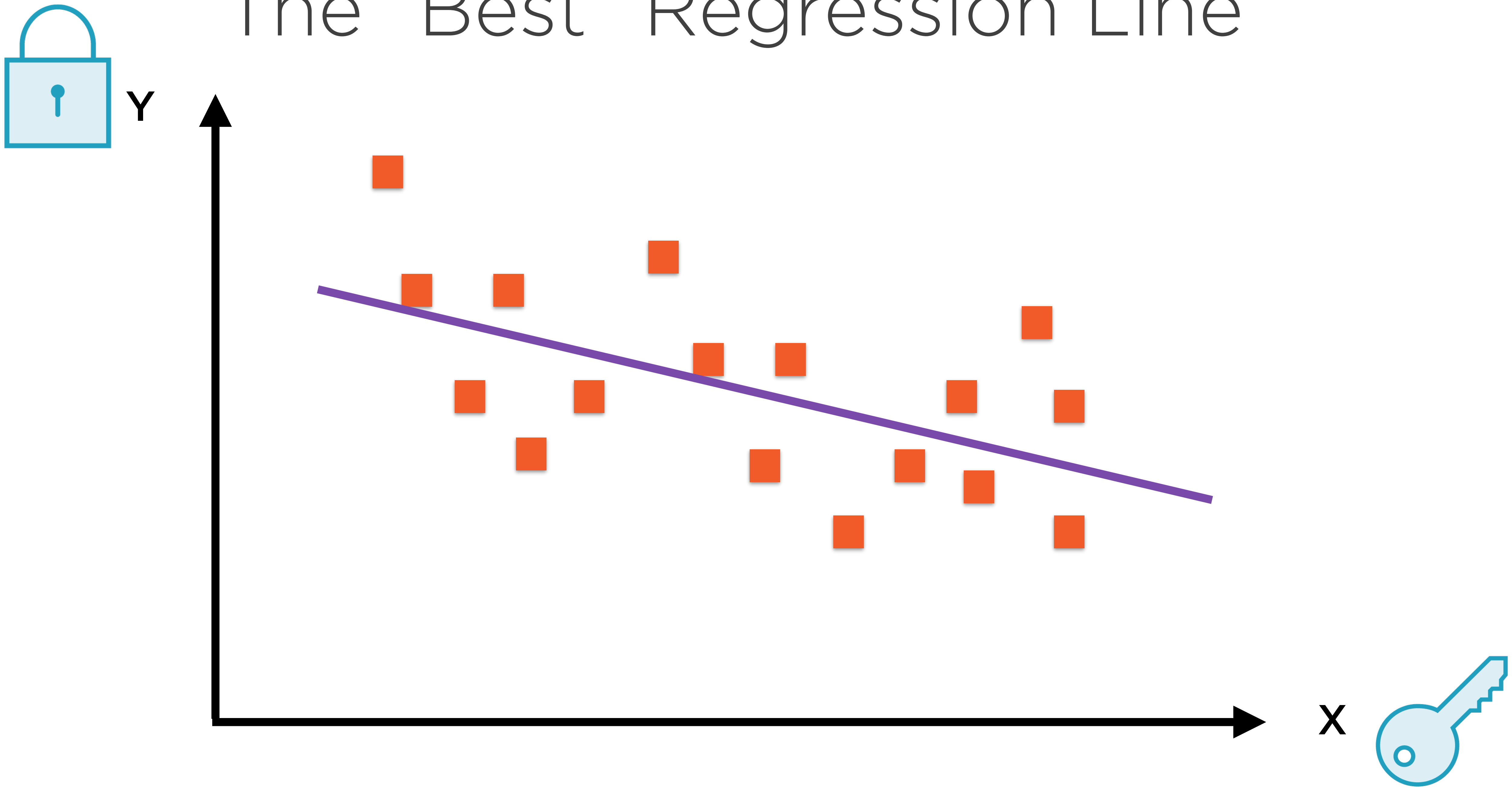
Cause

Explanatory variable

Effect

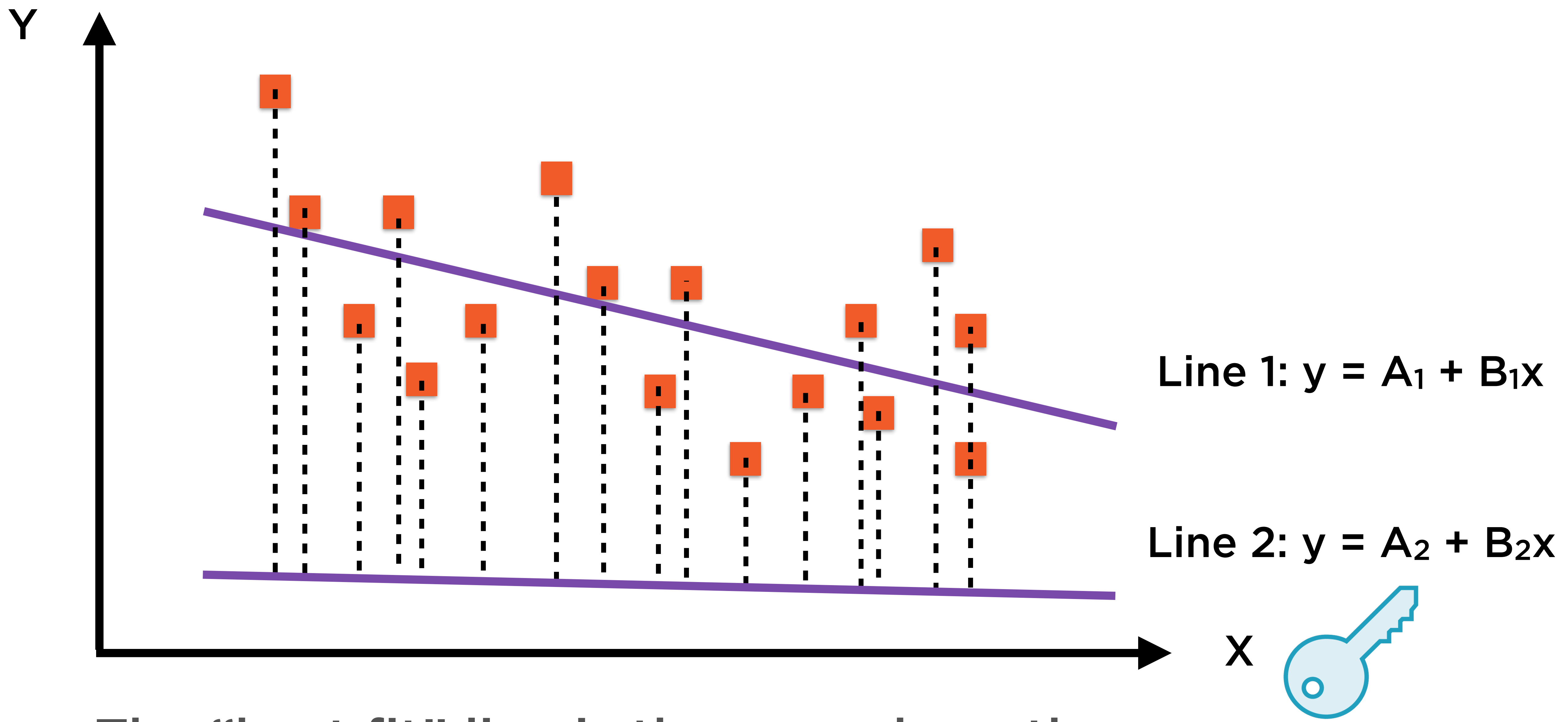
Dependent variable

The "Best" Regression Line



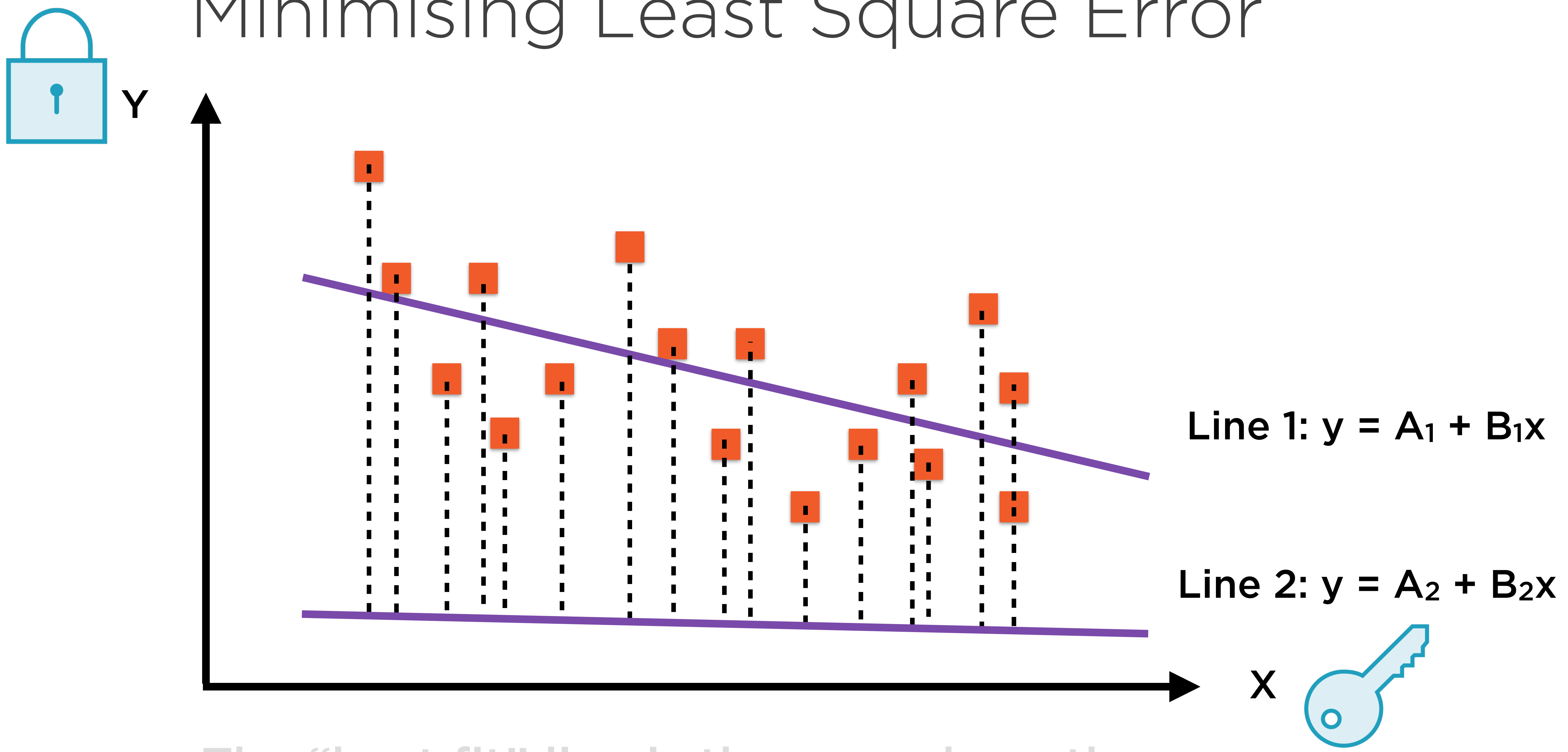
Linear Regression involves finding the "best fit" line

Minimising Least Square Error



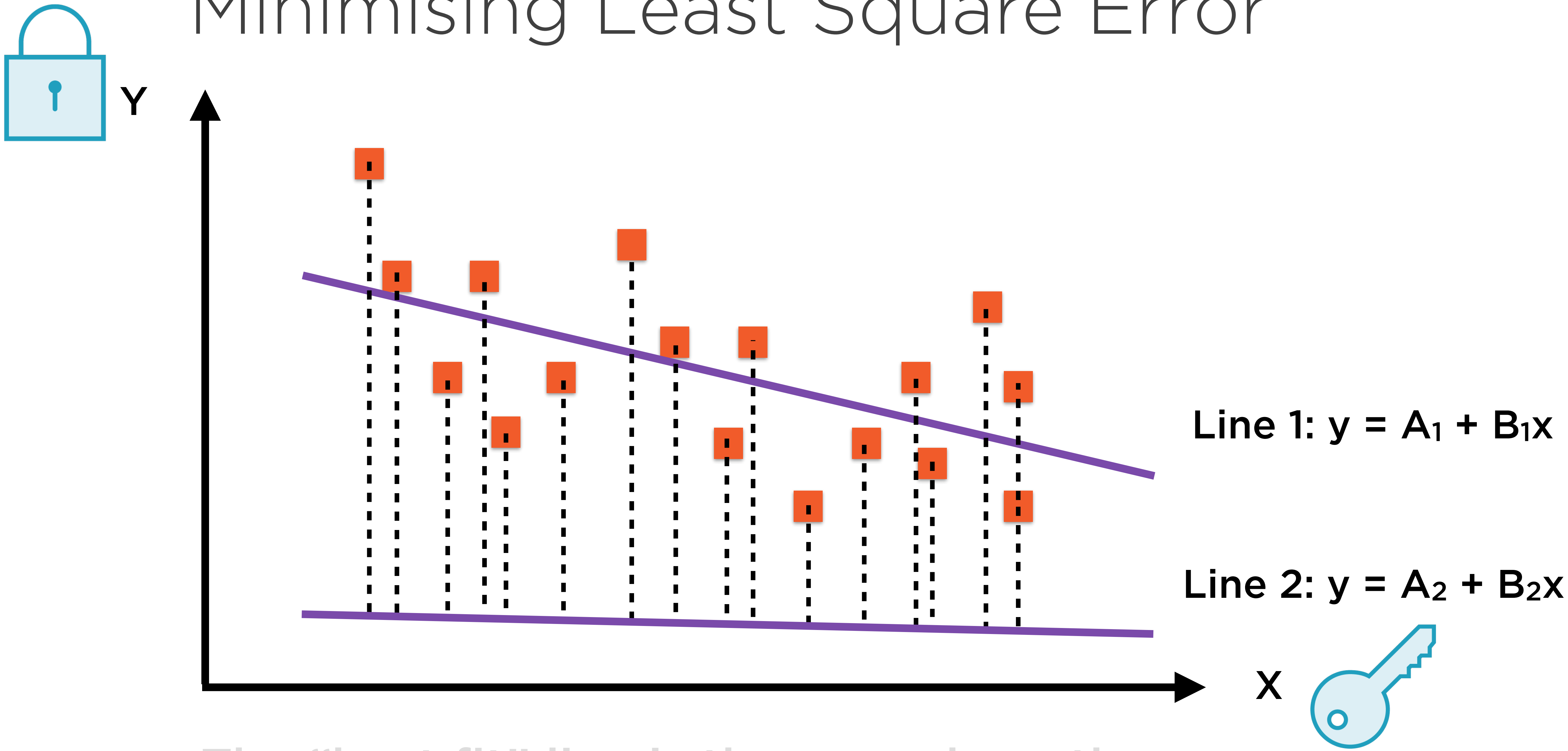
The “best fit” line is the one where the sum of the squares of the lengths of these dotted lines is minimum

Minimising Least Square Error



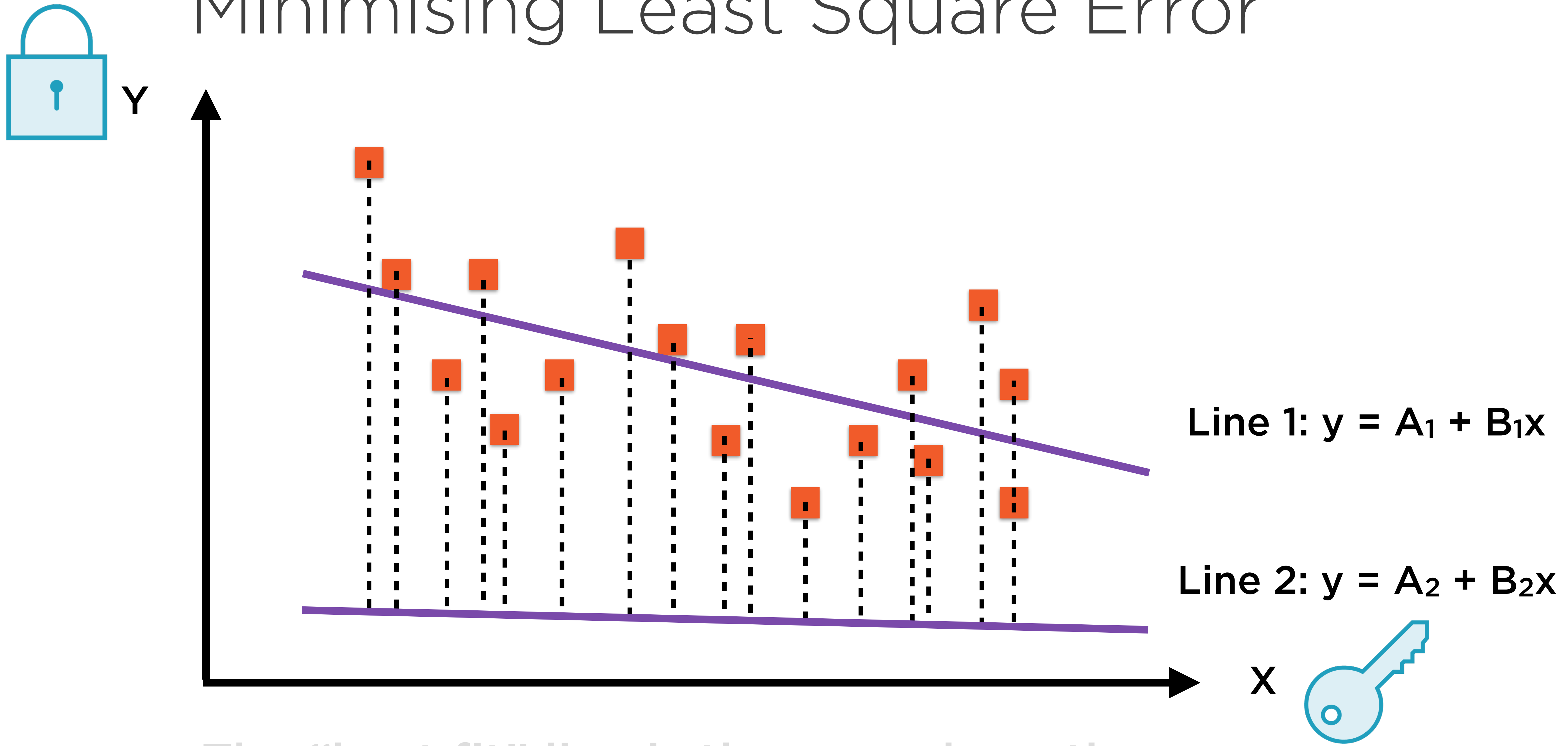
The "best fit" line is the one where the sum of the squares of the **lengths of these dotted lines** is minimum

Minimising Least Square Error



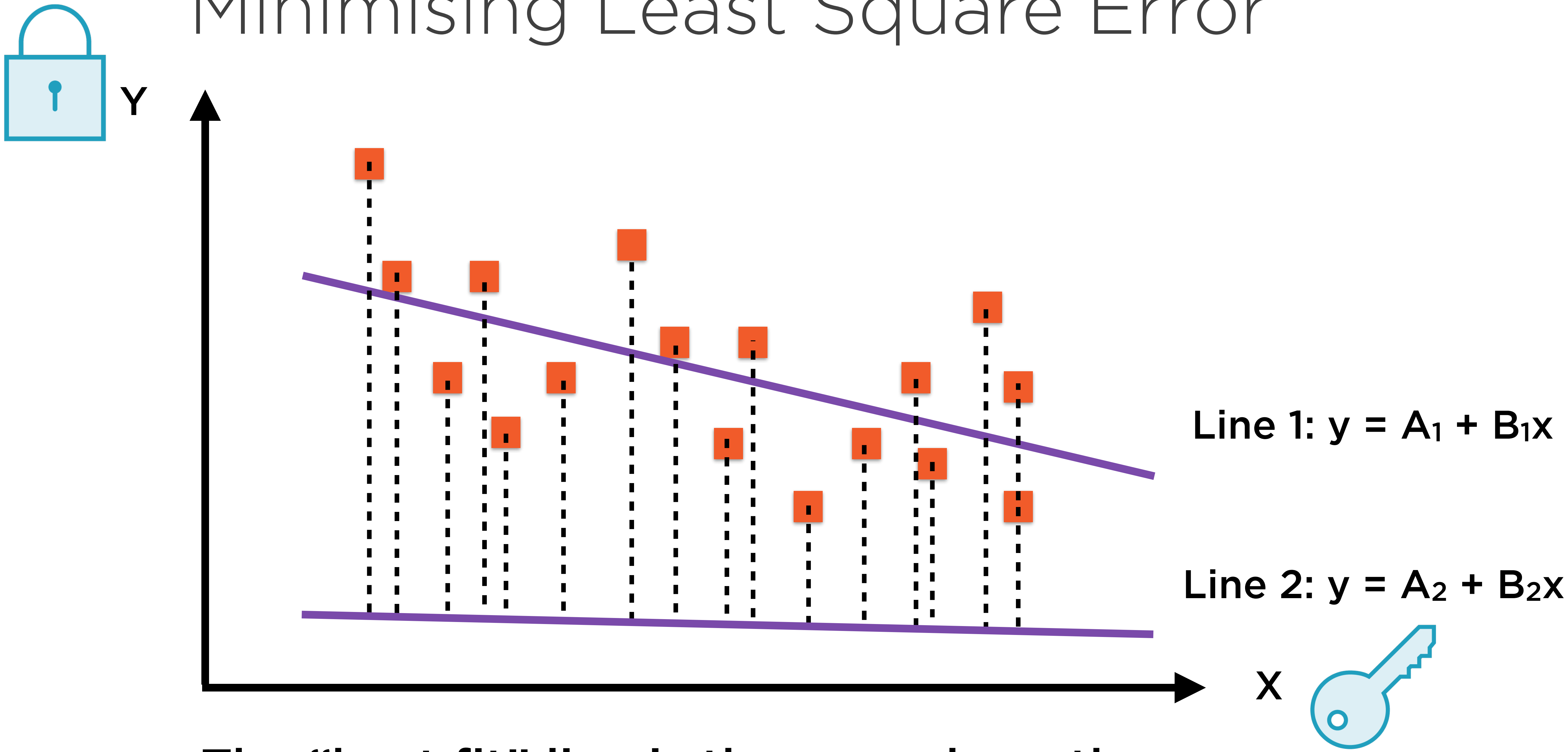
The “best fit” line is the one where the **sum of the squares** of the lengths of these dotted lines is minimum

Minimising Least Square Error



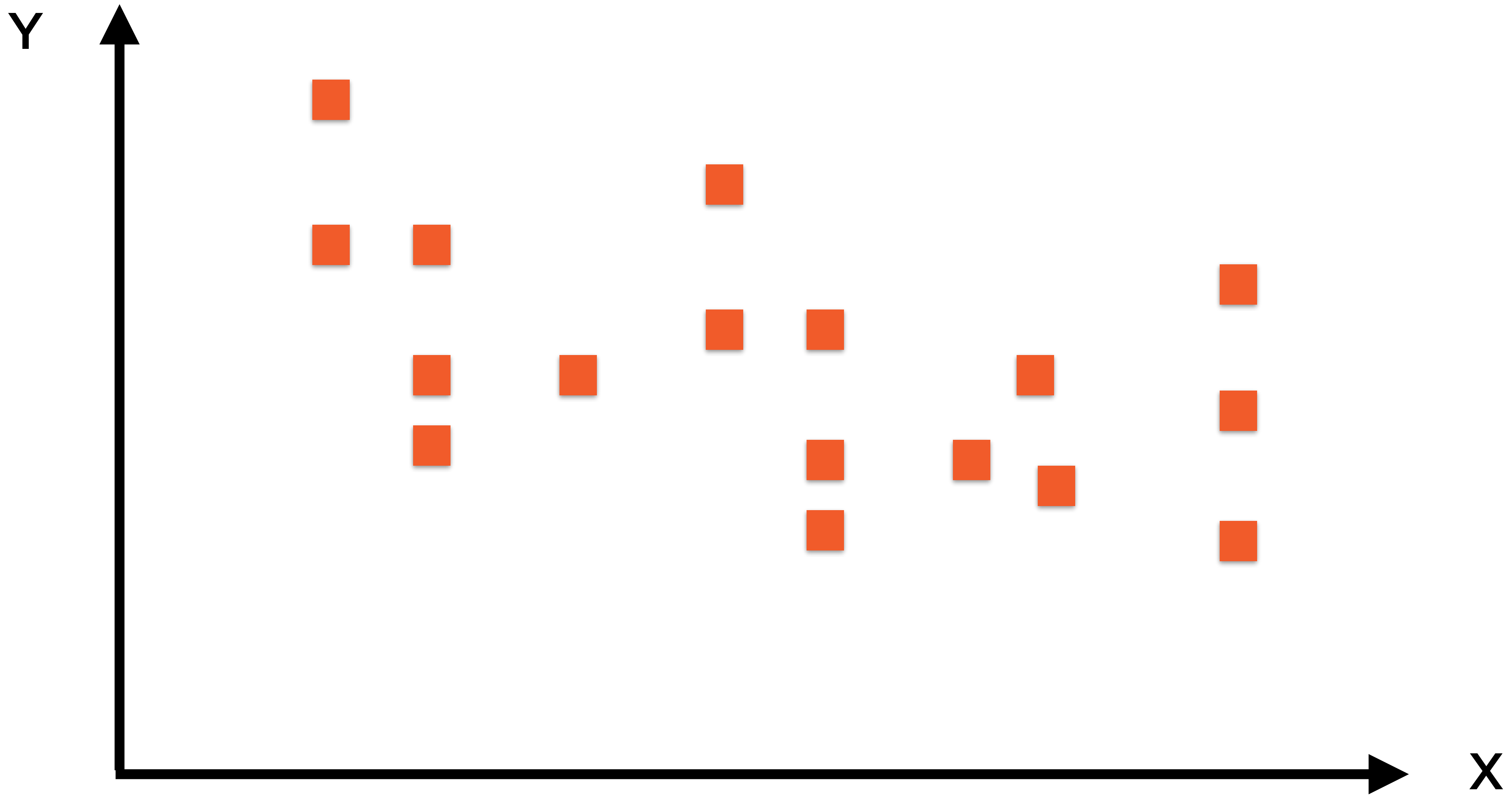
The "best fit" line is the one where the sum of the squares of the lengths of **the errors is minimum**

Minimising Least Square Error



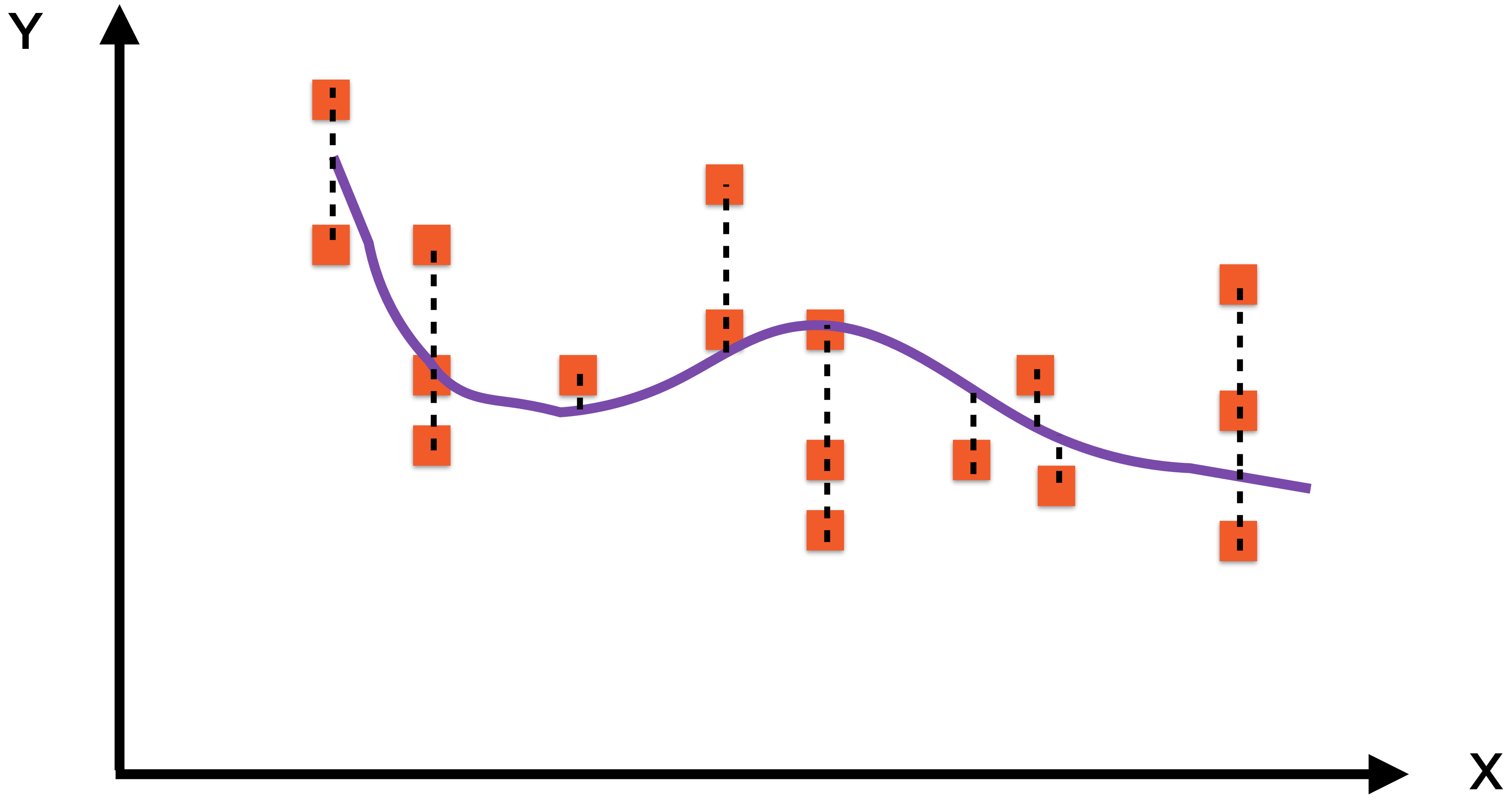
The "best fit" line is the one where the sum of the squares of the lengths of the errors is minimum

Connecting the Dots



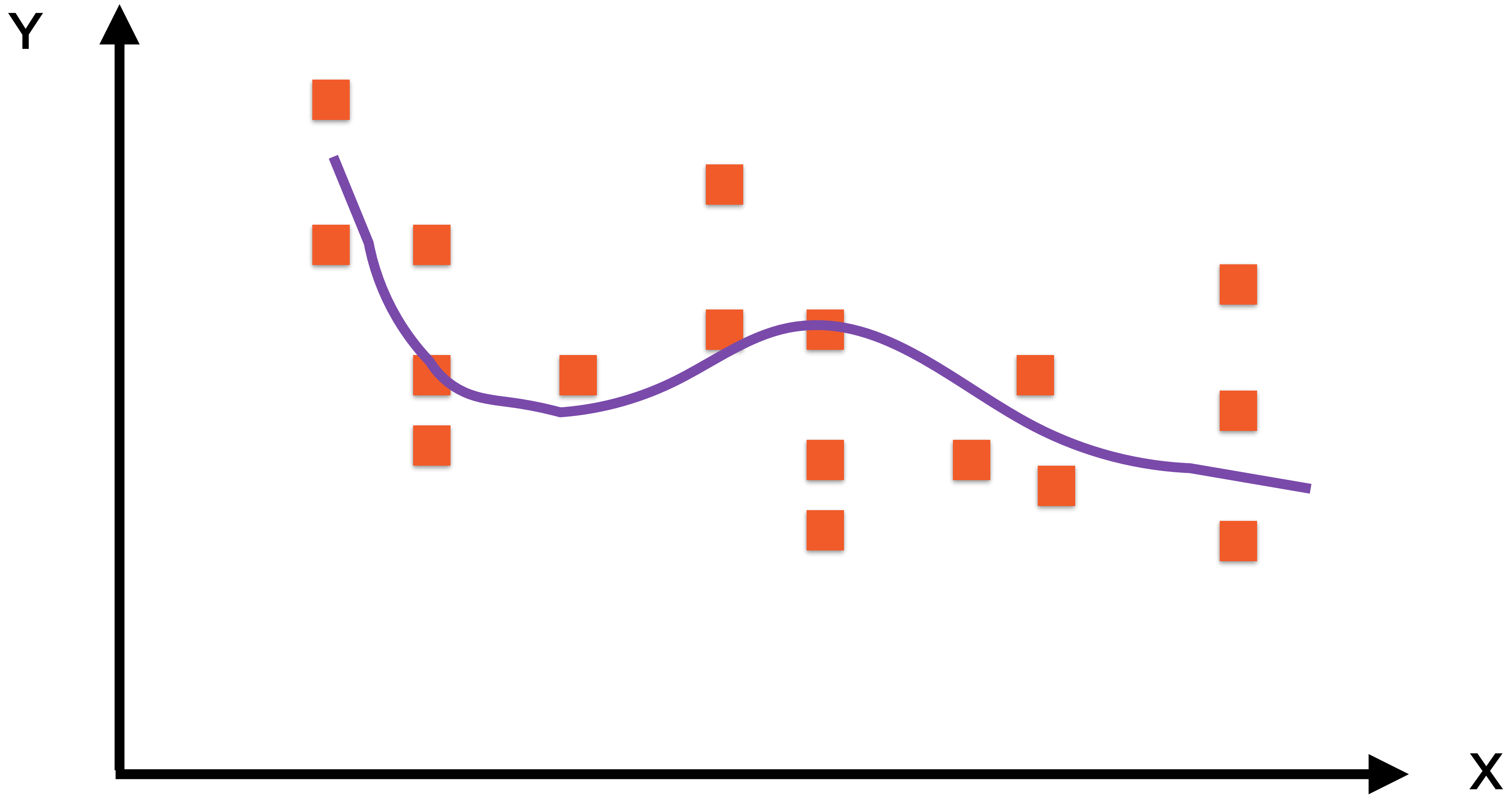
Challenge: Fit the "best" curve through these points

Good Fit?



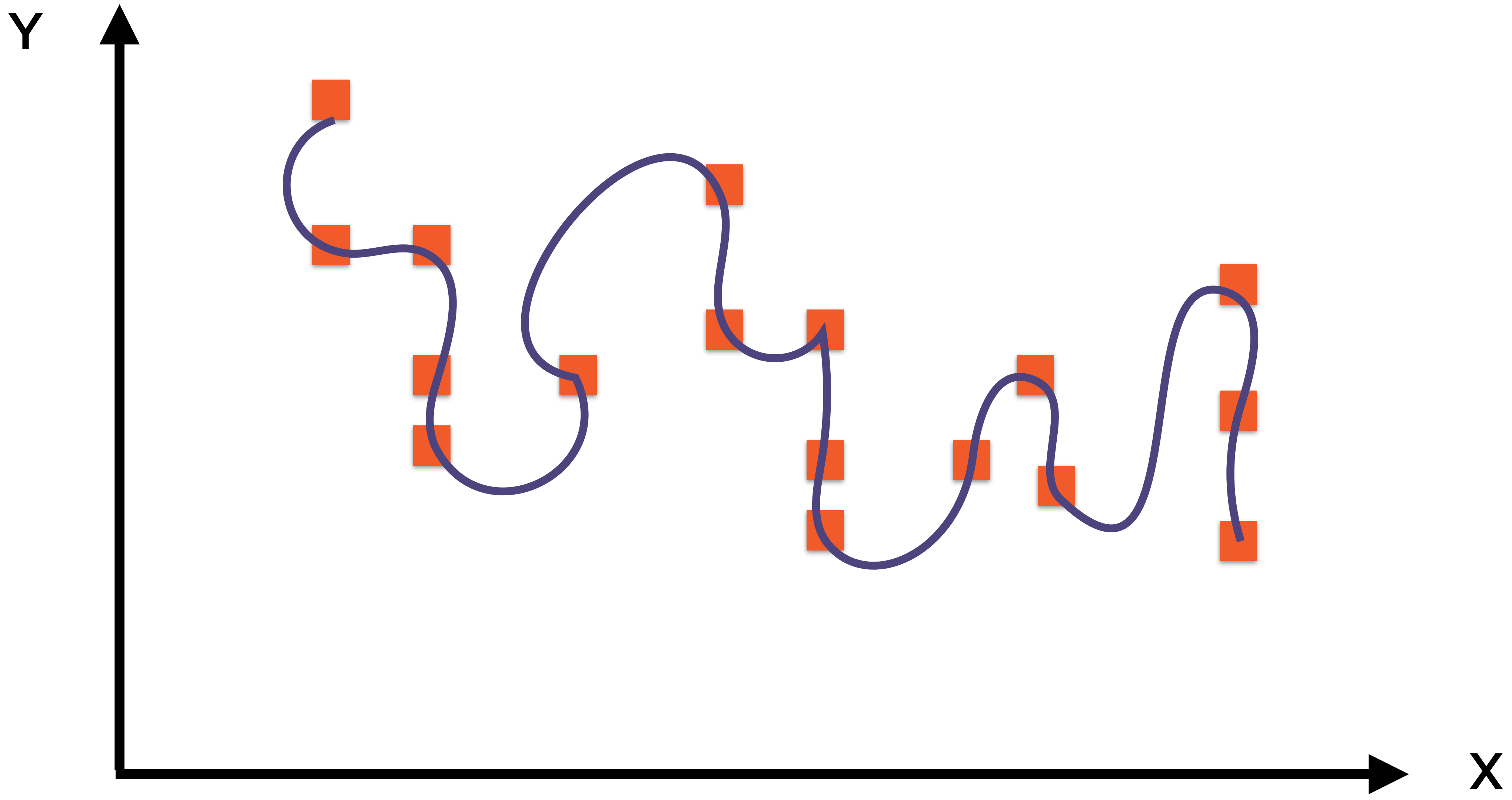
A curve has a "good fit" if the distances of points from the curve are small

Connecting the Dots



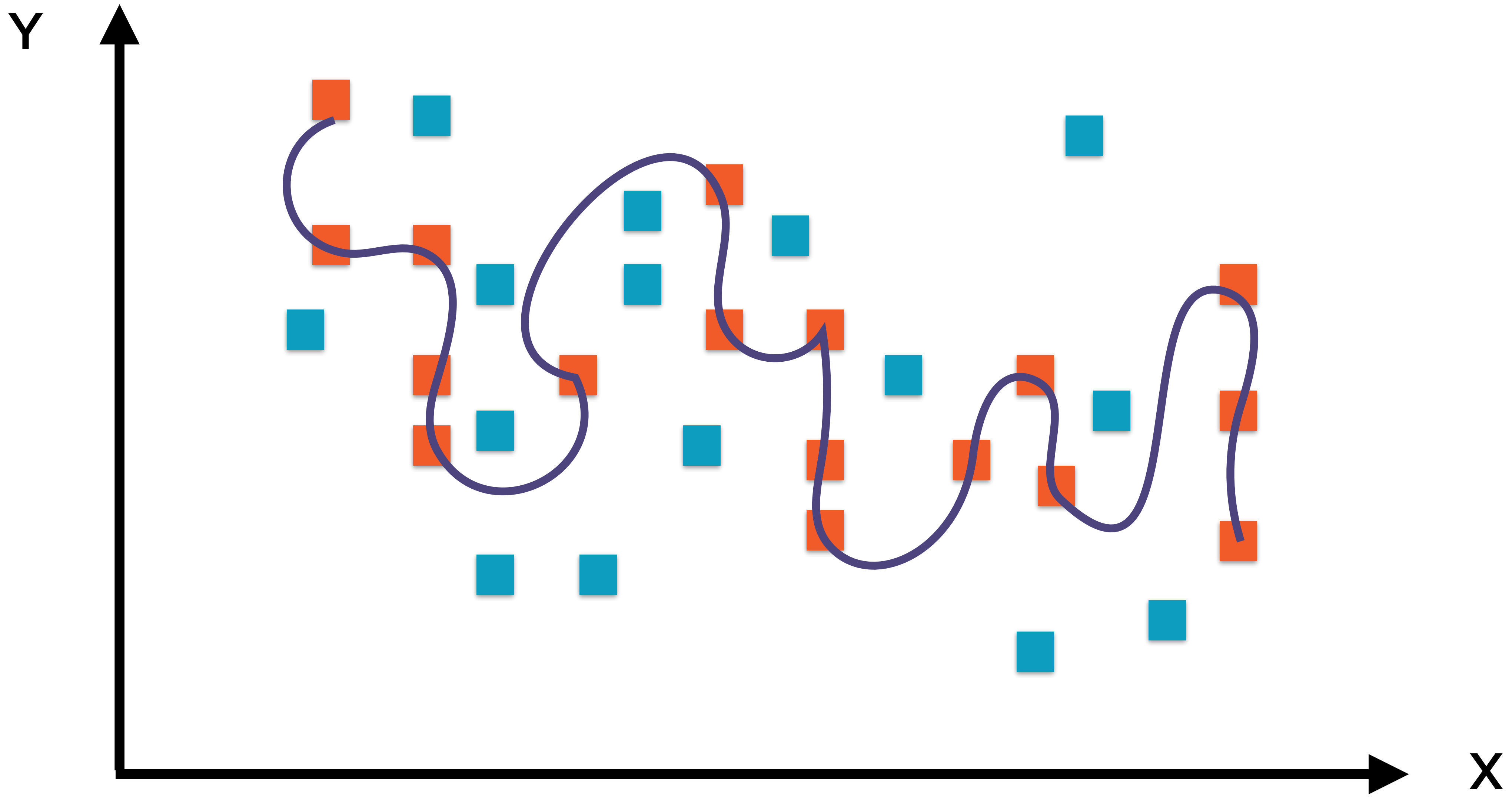
We could draw a pretty complex curve

Connecting the Dots



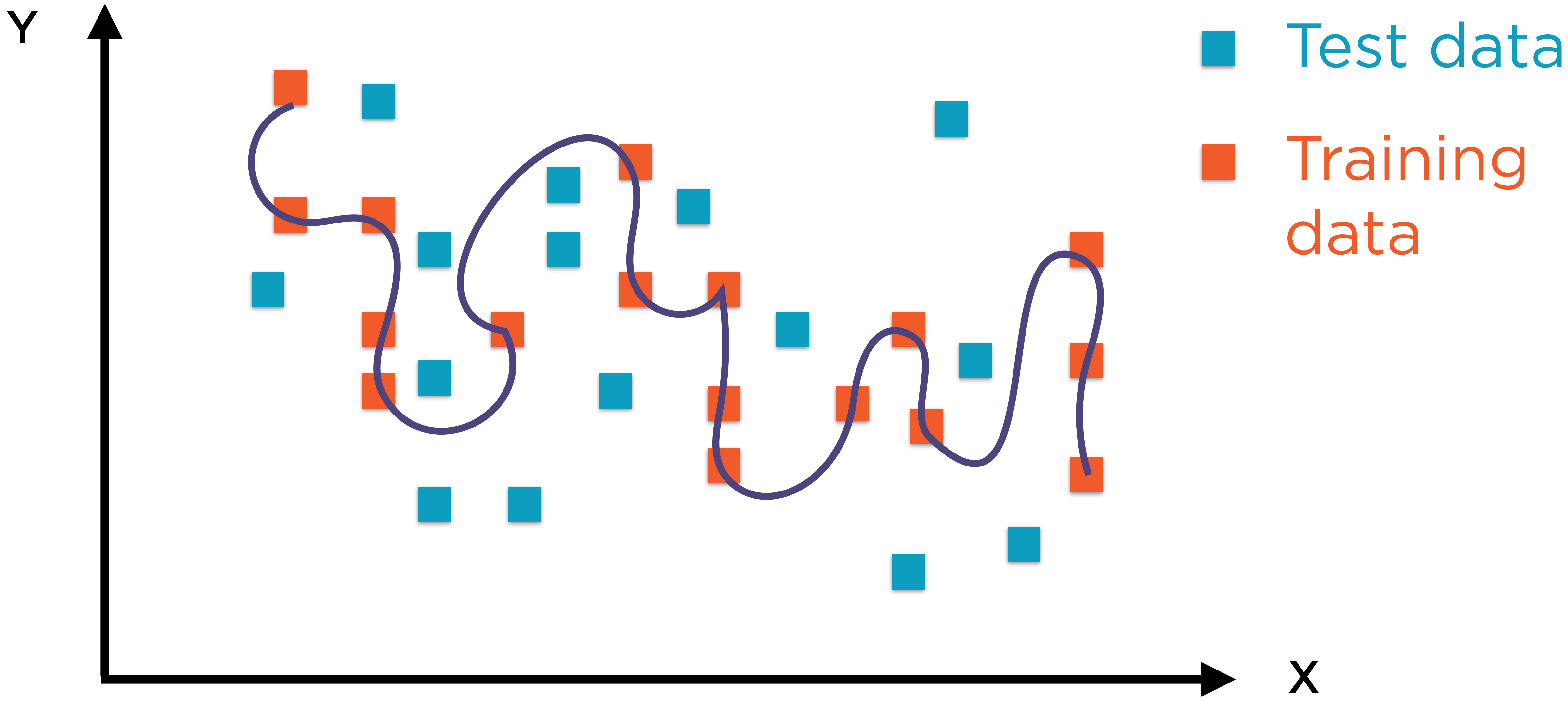
We can even make it pass through every single point

Connecting the Dots



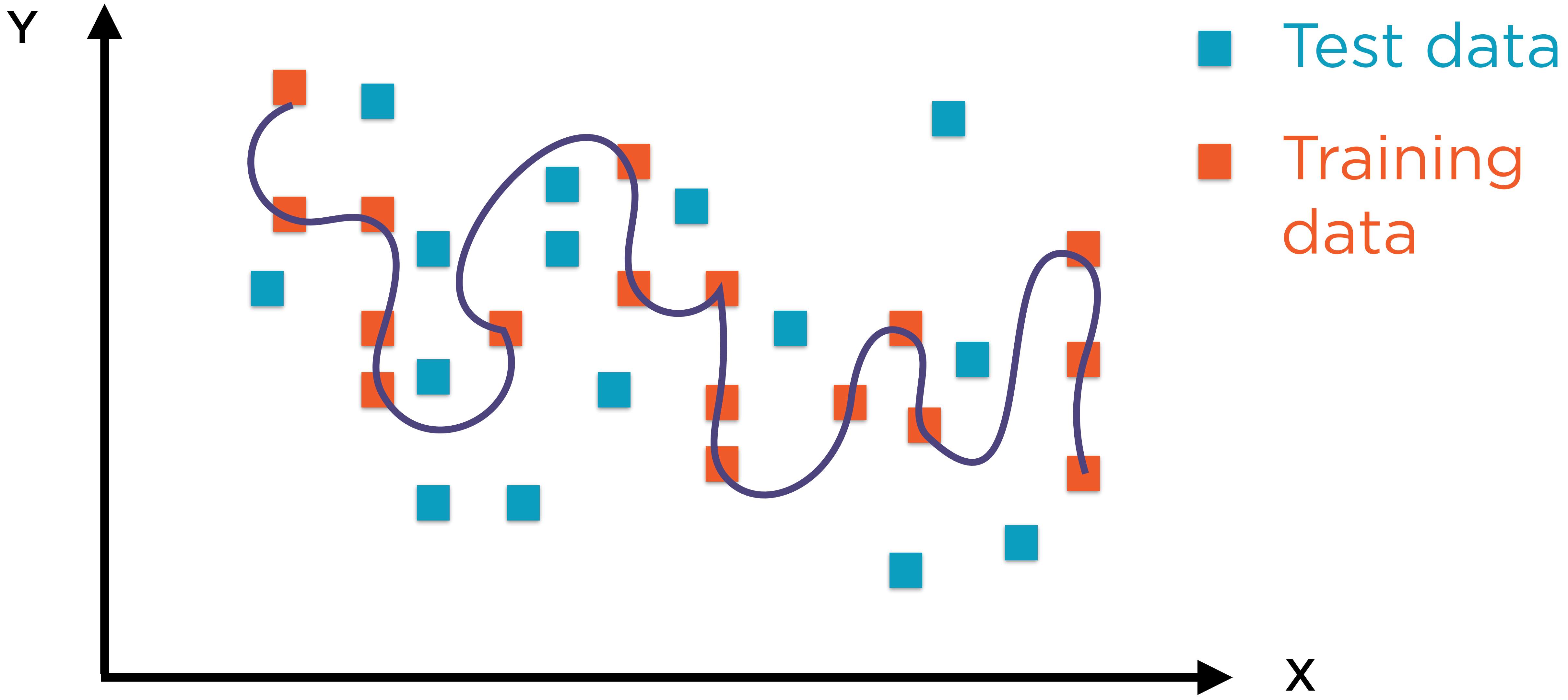
But given a new set of points, this curve might perform quite poorly

Connecting the Dots



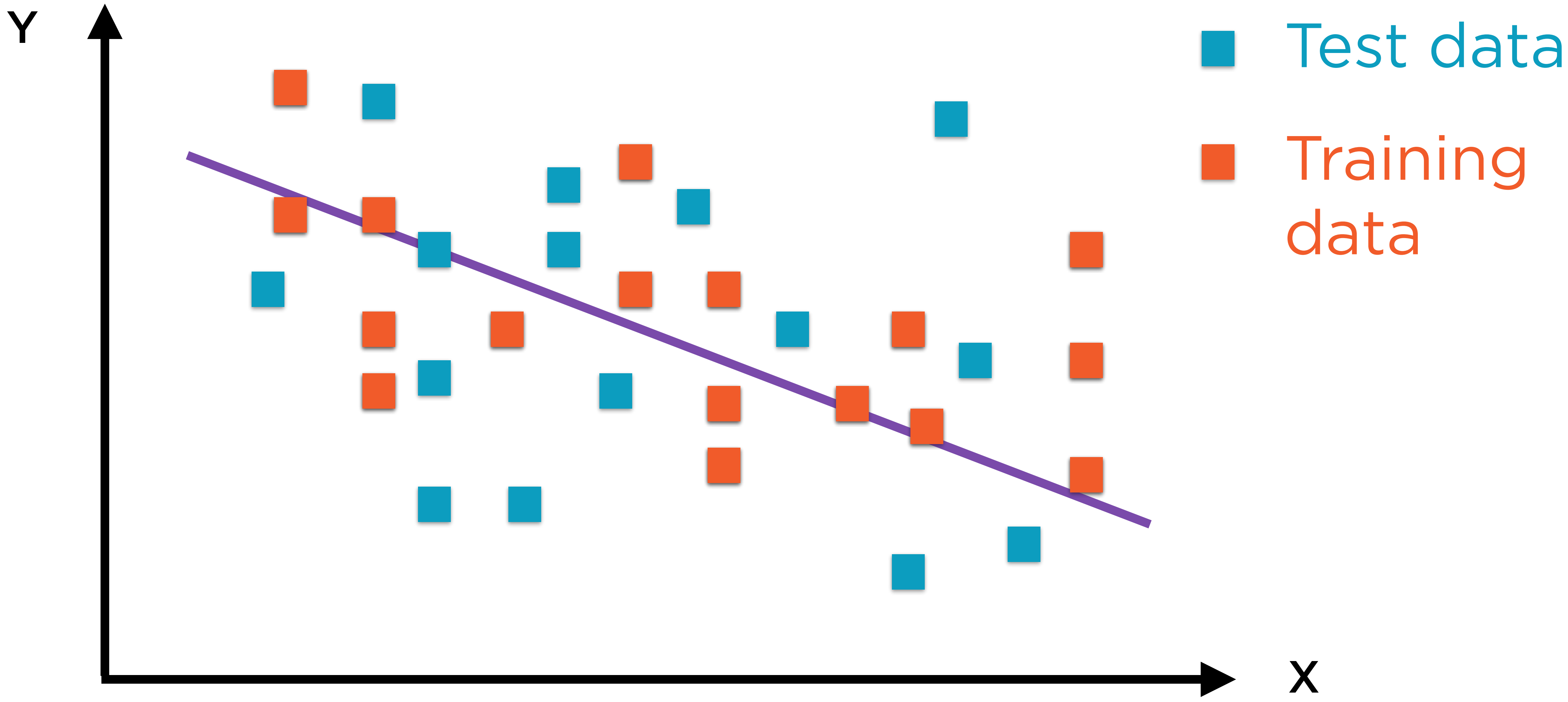
The original points were “training data”, the new points are “test data”

Overfitting



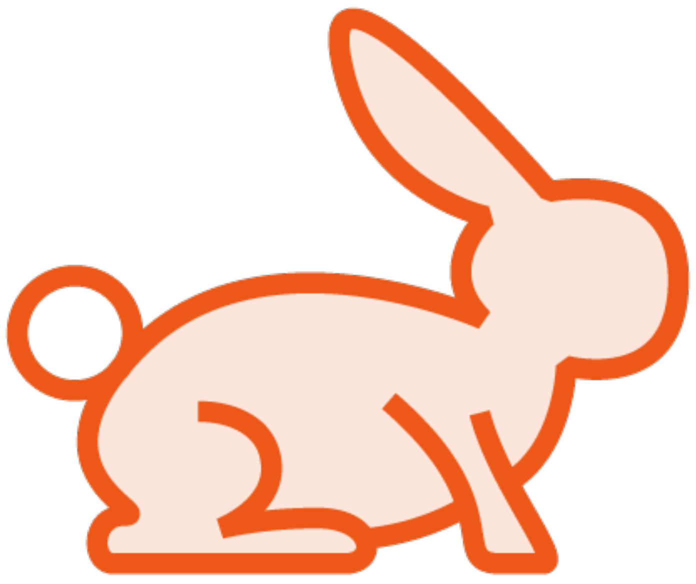
Great performance in training, poor performance in real usage

Connecting the Dots



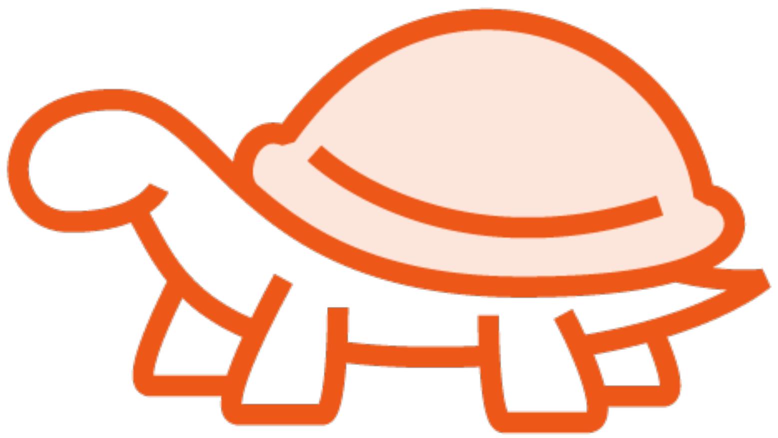
A simple straight line performs worse in training, but better with test data

Overfitting



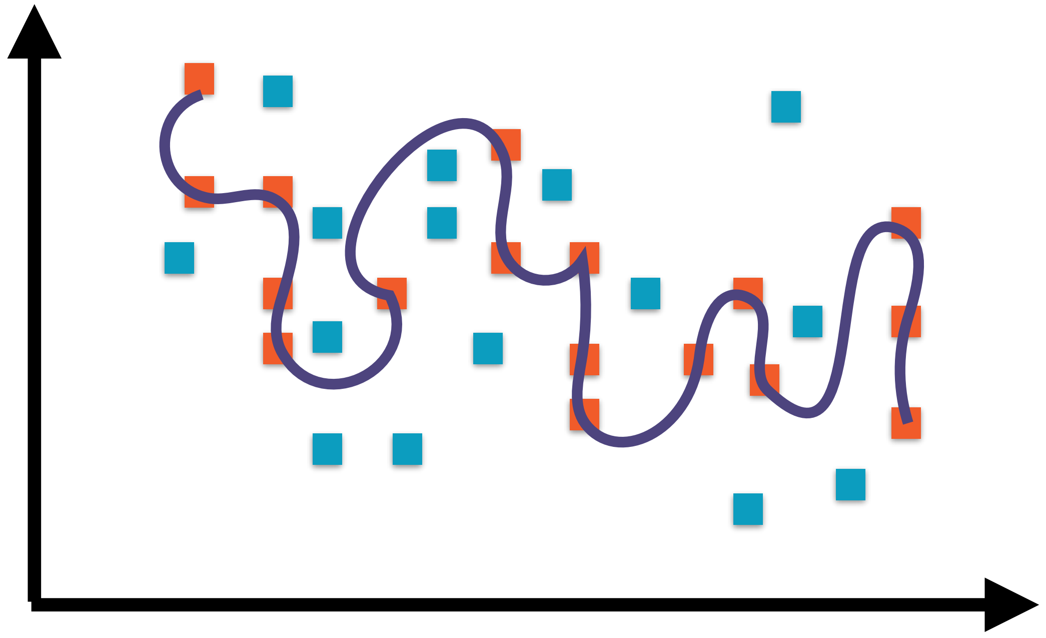
Low Training Error

Model does very well in training…



High Test Error

…but poorly with real data



Overfitting in Regression

Multi-collinearity in regression leads to overfitting

Model performs well in training, poorly in prediction

Various techniques to improve regression algorithm

Regularized Regression Models

Lasso Regression

Penalizes large regression coefficients

Ridge Regression

Also penalizes large regression coefficients

Elastic Net Regression

Simply combines lasso and ridge

Regularization

Penalize complex models

Add penalty to objective function

Penalty as function of regression coefficients

Forces optimizer to keep it simple

Regularized Regression Models

Lasso Regression

Penalizes large regression coefficients

Ridge Regression

Also penalizes large regression coefficients

Elastic Net Regression

Simply combines lasso and ridge

Ordinary MSE Regression

Minimize

$$\sqrt{(y^{actual} - y^{predicted})^2}$$

To find

A, B

The value of A and B define the "best fit" line

$y = A + Bx$

Lasso Regression

Minimize $\sqrt{(y^{actual} - y^{predicted})^2}$ $+ \alpha\ (|A| + |B|)$

To find

A, B

α is a hyperparameter

The value of A and B still define the "best fit" line

$$y = A + Bx$$

Lasso Regression

Minimize $\sqrt{(y^{actual} - y^{predicted})^2} + \alpha\,(|A| + |B|)$

To find

A, B

α is a hyperparameter

The value of A and B still define the "best fit" line

$$y = A + Bx$$

Lasso Regression

Minimize

$$\sqrt{(y^{actual} - y^{predicted})^2} + \alpha\,(|A| + |B|)$$

L-1 Norm of regression coefficients

To find

A, B

α is a hyperparameter

The value of A and B still define the "best fit" line

$y = A + Bx$

Ridge Regression

Minimize

$$\sqrt{(y^{actual} - y^{predicted})^2} + \alpha (|A|^2 + |B|^2)$$

L-2 Norm of regression coefficients

To find

A, B

α is a hyperparameter

The value of A and B still define the “best fit” line

$$y = A + Bx$$

Lasso Regression

Add penalty for large coefficients

Penalty term is L-1 norm of coefficients

Penalty weighted by hyperparameter α

Lasso Regression

$\alpha = 0$ ~ Regular (MSE regression)

$\alpha \to \infty$ ~ Force small coefficients to zero

Model selection by tuning α

Eliminates unimportant features

Lasso Regression

"Lasso" ~ *Least Absolute Shrinkage and Selection Operator*

Math is complex

No closed form, needs numeric solution

Ridge Regression

Minimize $\sqrt{(y^{actual} - y^{predicted})^2}$ $+ \alpha (|A|^2 + |B|^2)$

L-2 Norm of regression coefficients

To find

A, B

α is a hyperparameter

The value of A and B still define the "best fit" line

$y = A + Bx$

Ridge Regression

Add penalty for large coefficients

Penalty term is L-2 norm of coefficients

Penalty weighted by hyperparameter α

Ridge Regression

Unlike lasso, ridge regression has closed-form solution

Unlike lasso, ridge regression will not force coefficients to 0

- Does not perform model selection

Demo

Implement linear regression in spark.ml

Use the Elastic Net parameter to range between Lasso and Ridge regression

Perform hyperparameter tuning to find the best possible model

Summary

Estimators, Transformers chained to form an ML pipeline

Evaluating classifiers using the confusion matrix, accuracy, precision and recall

Decision trees and random forests for classification

Specialized regression models such as Lasso and Ridge regression