CS4225/CS5425 Big Data Systems for Data Science

Spark II: Advanced Topics

Ai Xin
School of Computing
National University of Singapore
aixin@comp.nus.edu.sg

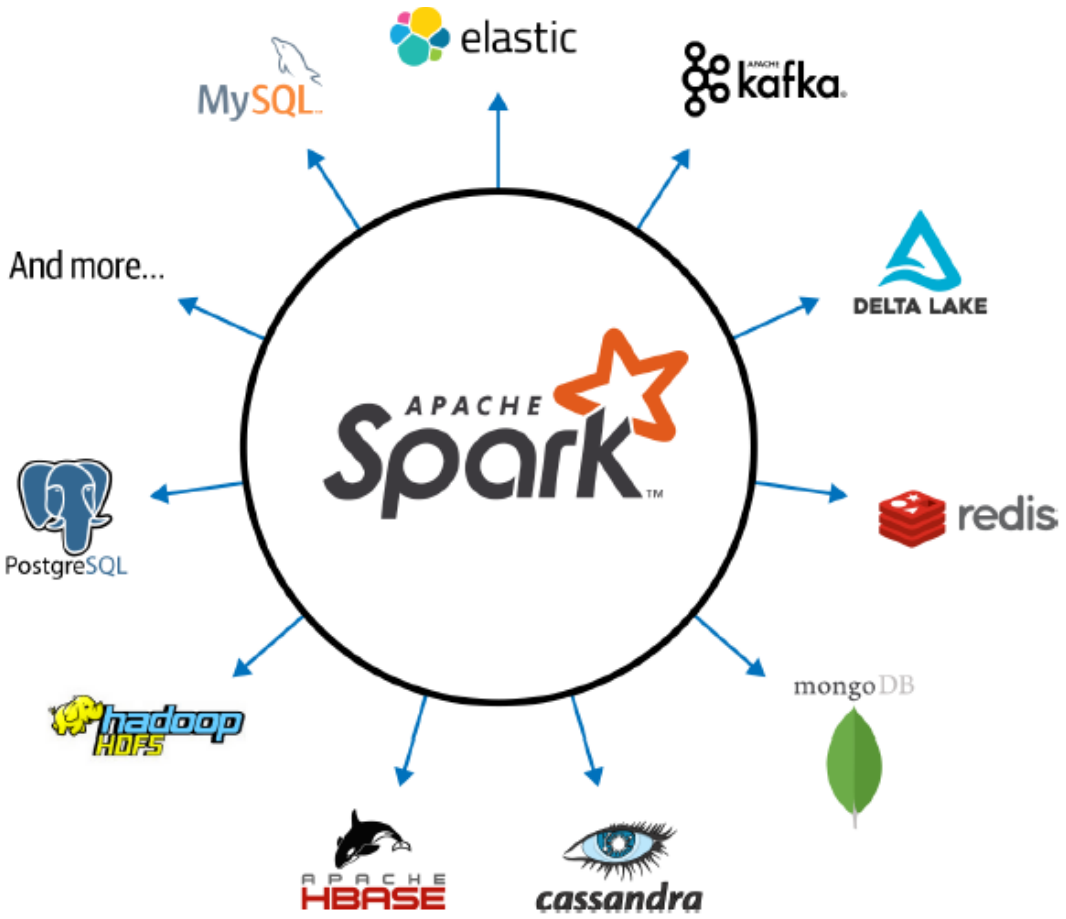
School of Computing

Today's Plan

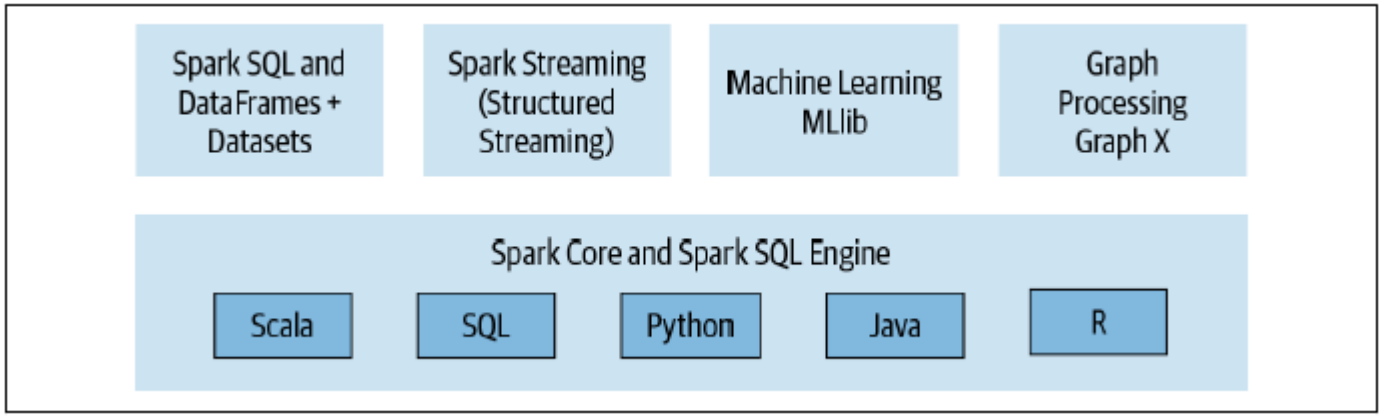
- Spark SQL
- Machine Learning with Spark ML

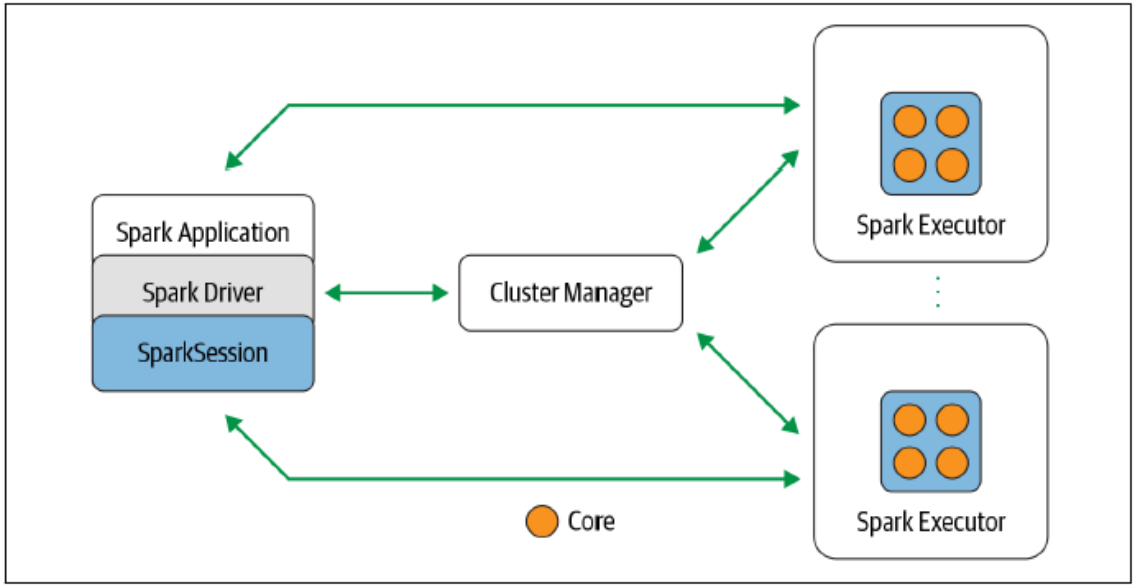
Spark Design Philosophy

- Speed
- Ease of use
- Modularity
- Extensibility



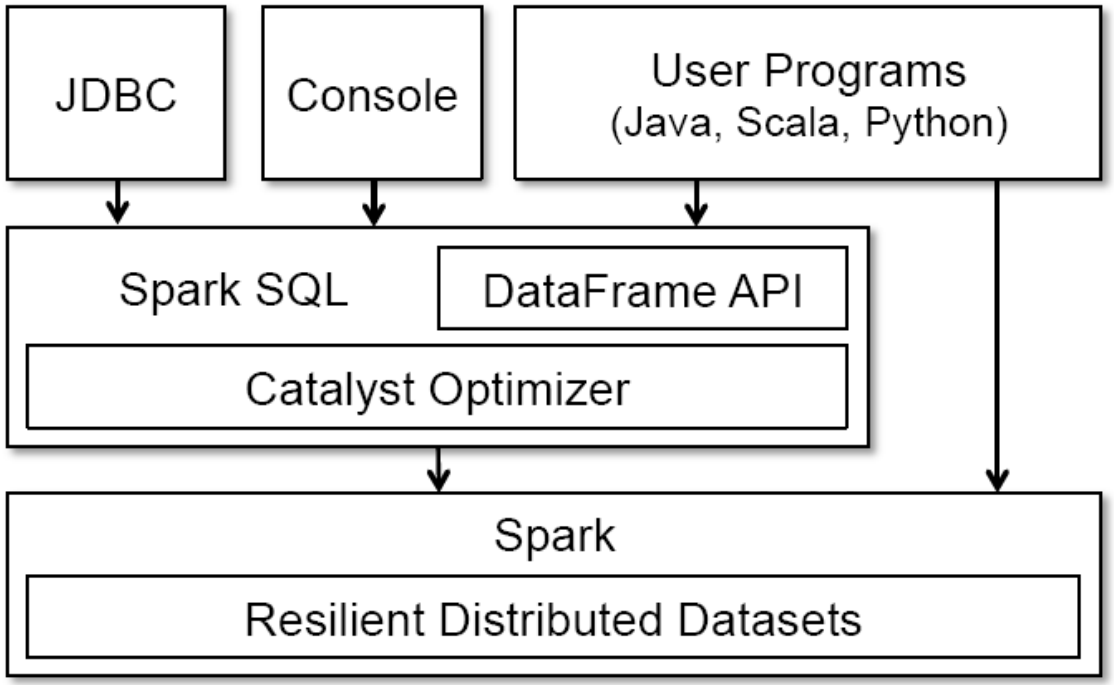
Spark: a unified stack for distributed execution





Spark SQL

- Unifies Spark components and permits abstraction to DataFrames/Datasets in Java, Scala, Python, and R
- Keep track of schema and support optimized relational operations



RDD vs. DataFrame

- RDD

```
# Create an RDD of tuples (name, age)
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30),
  ("TD", 35), ("Brooke", 25)])
# Use map and reduceByKey transformations with their lambda
# expressions to aggregate and then compute average

agesRDD = (dataRDD
  .map(lambda x: (x[0], (x[1], 1)))
  .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
  .map(lambda x: (x[0], x[1][0]/x[1][1])))
```

- DataFrame

```
# Create a DataFrame
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31), ("Jules", 30),

  ("TD", 35), ("Brooke", 25)], ["name", "age"])
# Group the same names together, aggregate their ages, and compute an average
avg_df = data_df.groupBy("name").agg(avg("age"))
# Show the results of the final execution
avg_df.show()
```

```
+------+--------+
|  name|avg(age)|
+------+--------+
|Brooke|    22.5|
| Jules|    30.0|
|    TD|    35.0|
| Denny|    31.0|
+------+--------+
```

RDD vs. DataFrame

- RDD
  - Instruct Spark how to compute the query
  - The intention is completely opaque to Spark
  - Spark also does not understand the structure of the data in RDDs (which is arbitrary Python objects) or the semantics of user functions (which contain arbitrary code)
- DataFrame
  - Tell Spark what to do, instead of How to do
  - The code is far more expressive as well as simpler
    - Using a domain specific language (DSL) similar to python pandas
    - Use high-level DSL operators to compose the query
  - Spark can inspect or parse this query and understand our intention, it can then optimize or arrange the operations for efficient execution

Spark SQL

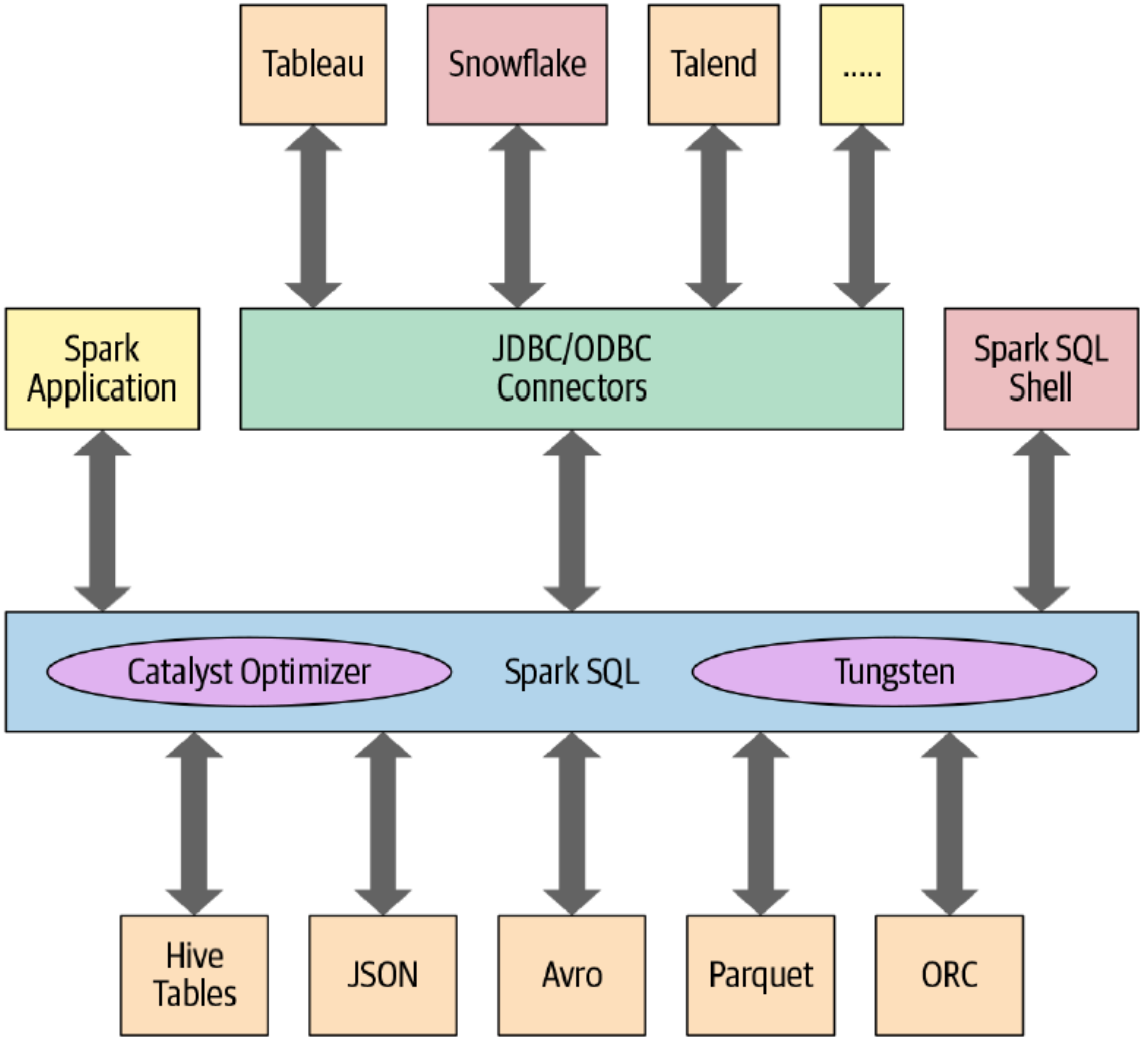
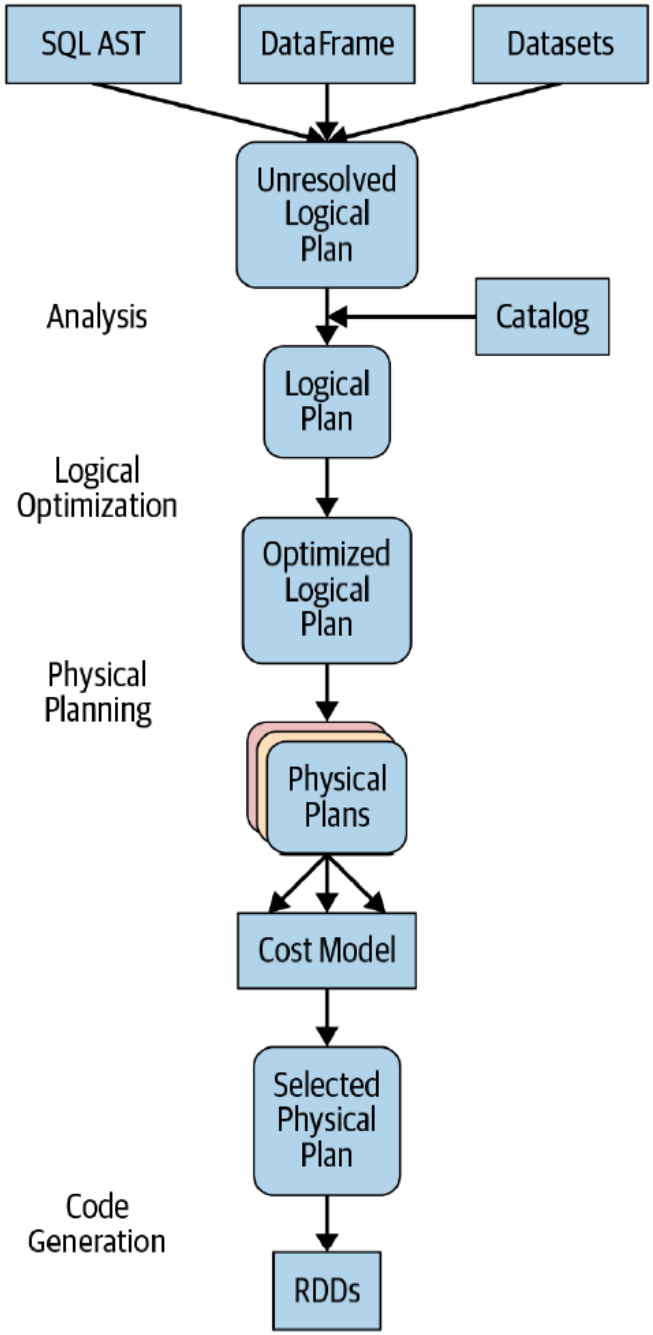


Tableau
Snowflake
Talend
.....
Spark Application
JDBC/ODBC Connectors
Spark SQL Shell
Catalyst Optimizer
Spark SQL
Tungsten
Hive Tables
JSON
Avro
Parquet
ORC

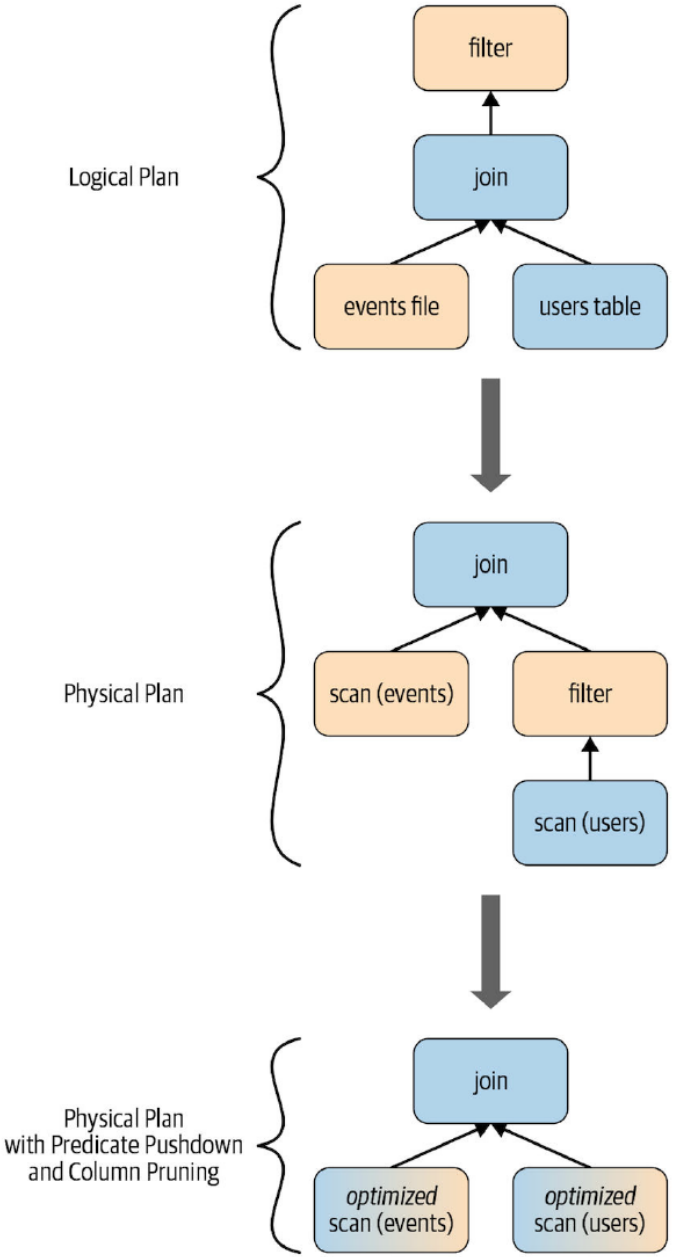
The Catalyst Optimizer

- Takes a computational query and converts it into an execution plan through four transformational phases:
  1. Analysis
  2. Logical optimization
  3. Physical planning
  4. Code generation

A Spark computation's four-phase journey

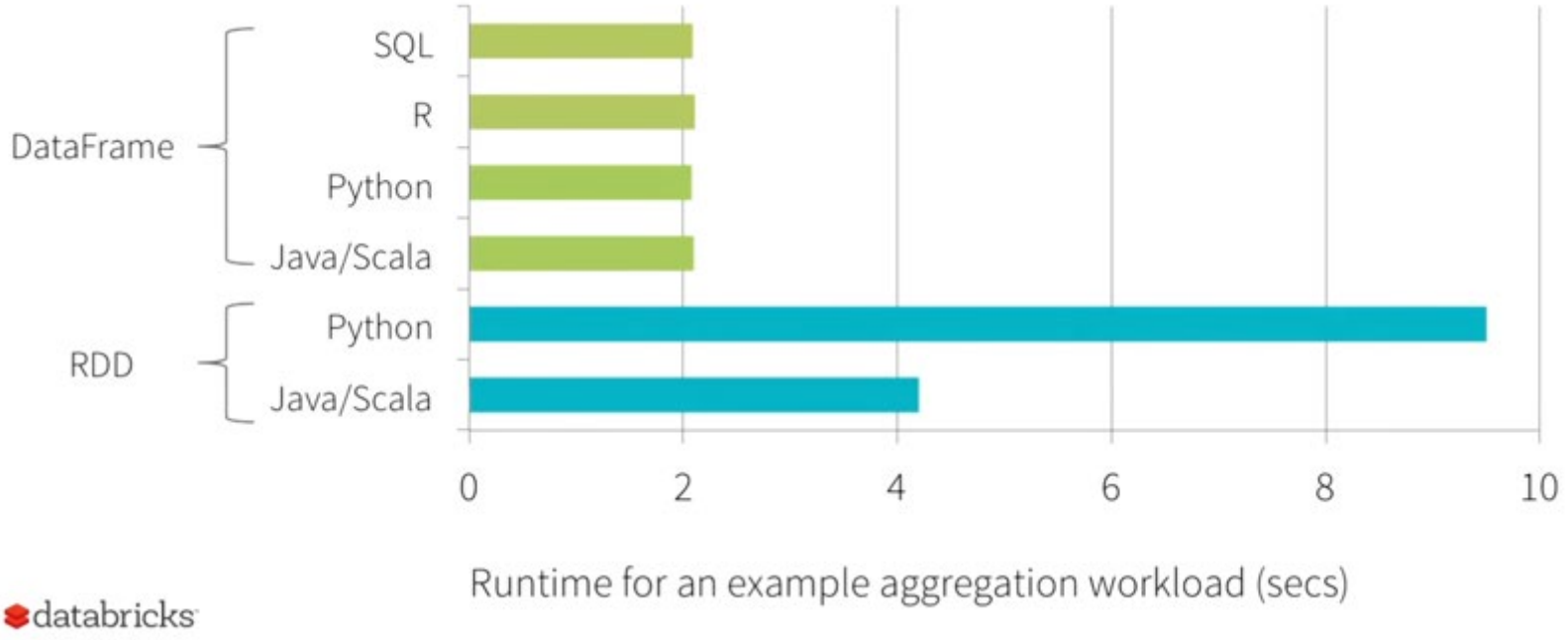


```scala
// In Scala
// Users DataFrame read from a Parquet table
val usersDF  = ...
// Events DataFrame read from a Parquet table
val eventsDF = ...
// Join two DataFrames
val joinedDF = users
  .join(events, users("id") === events("uid"))
  .filter(events("date") > "2015-01-01")
```



Benefit of Logical Plan

- Performance Parity Across Languages



Source: https://youtu.be/VbSar607HM0

Today's Plan

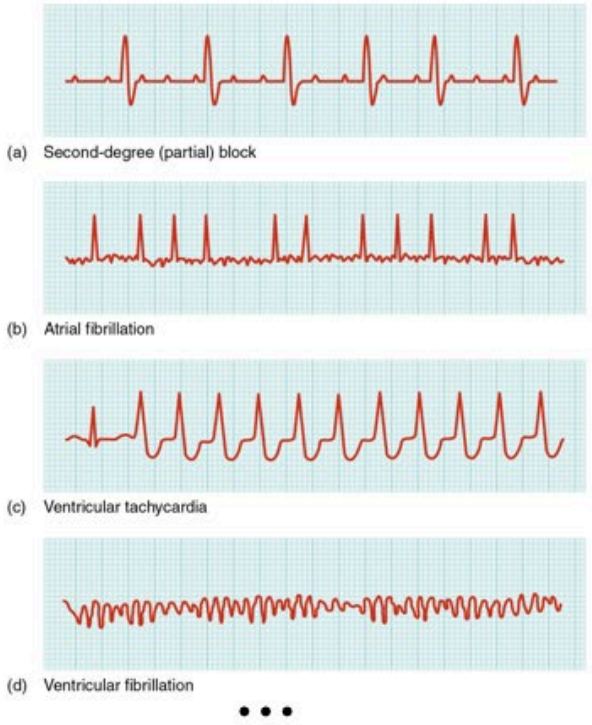
- Spark SQL
- **Machine Learning with Spark ML**

Example: Heart Rhythm Classification

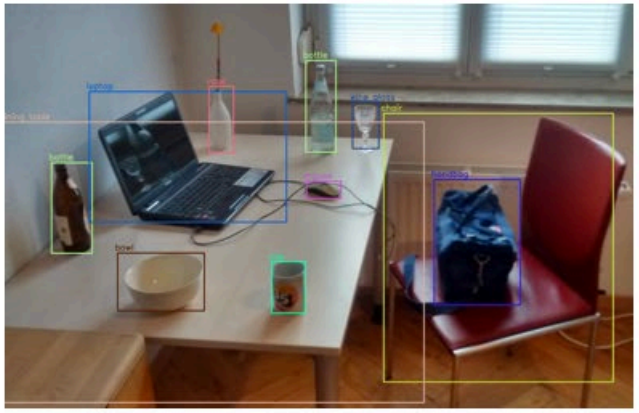
Medical ECG devices

Wearable devices

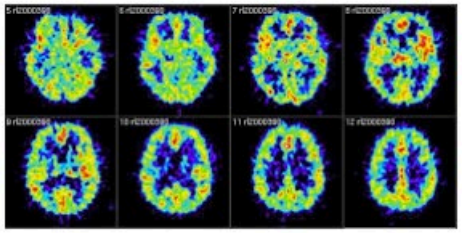
Classification



More Applications



Object Recognition



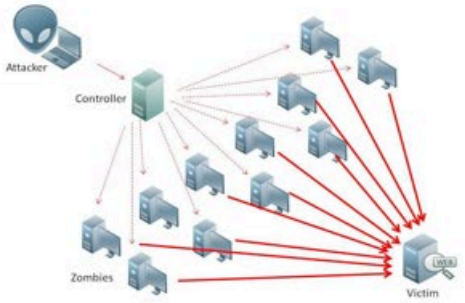
Medical Imaging

PayPal

PayPal Customre Care

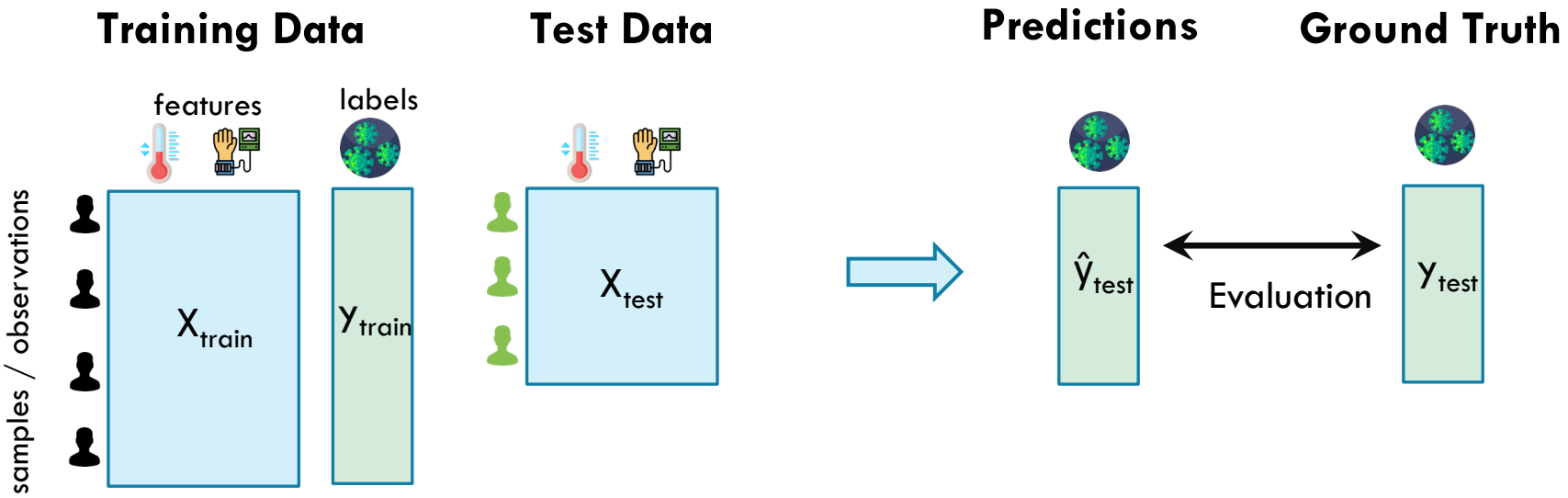
Hi,
Dear customer
At first Thank you for paying attention to PayPal Customer Care.
We contact you for confirming you Paypal account.because of security reason you have to confirm you account in PayPal again.our log on your account show us some illegal usage then we want you to pay some time and Confirm your account again. for confirming just login to PayPal with attached form just from

Spam Detection



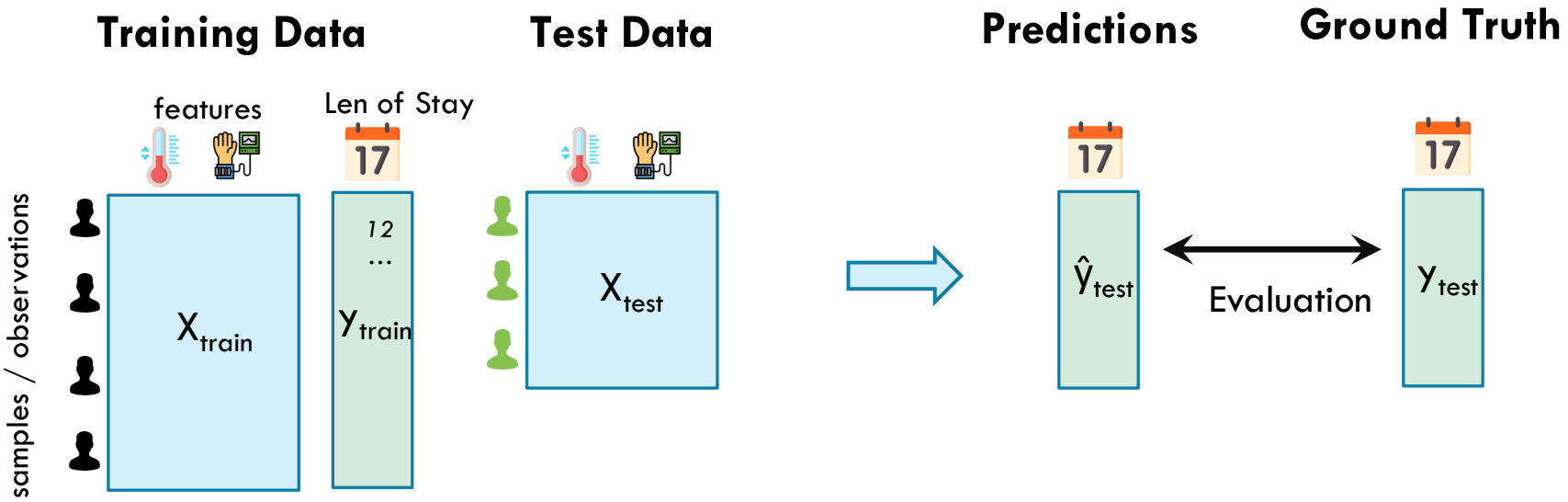
Network Intrusion Detection

Classification: Problem Setup



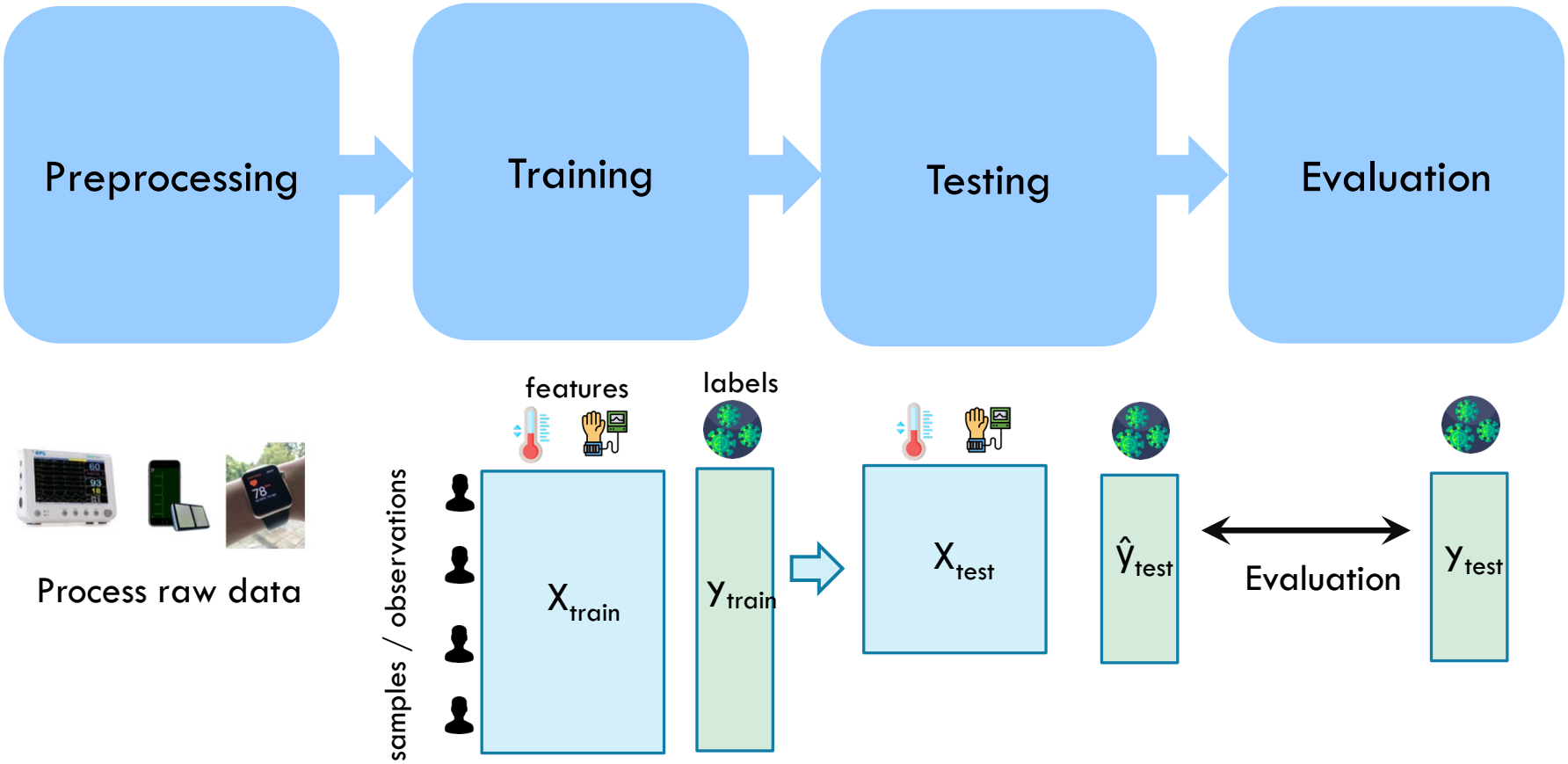
Classification: Categorize samples into *classes*, given training data

Regression: Problem Setup

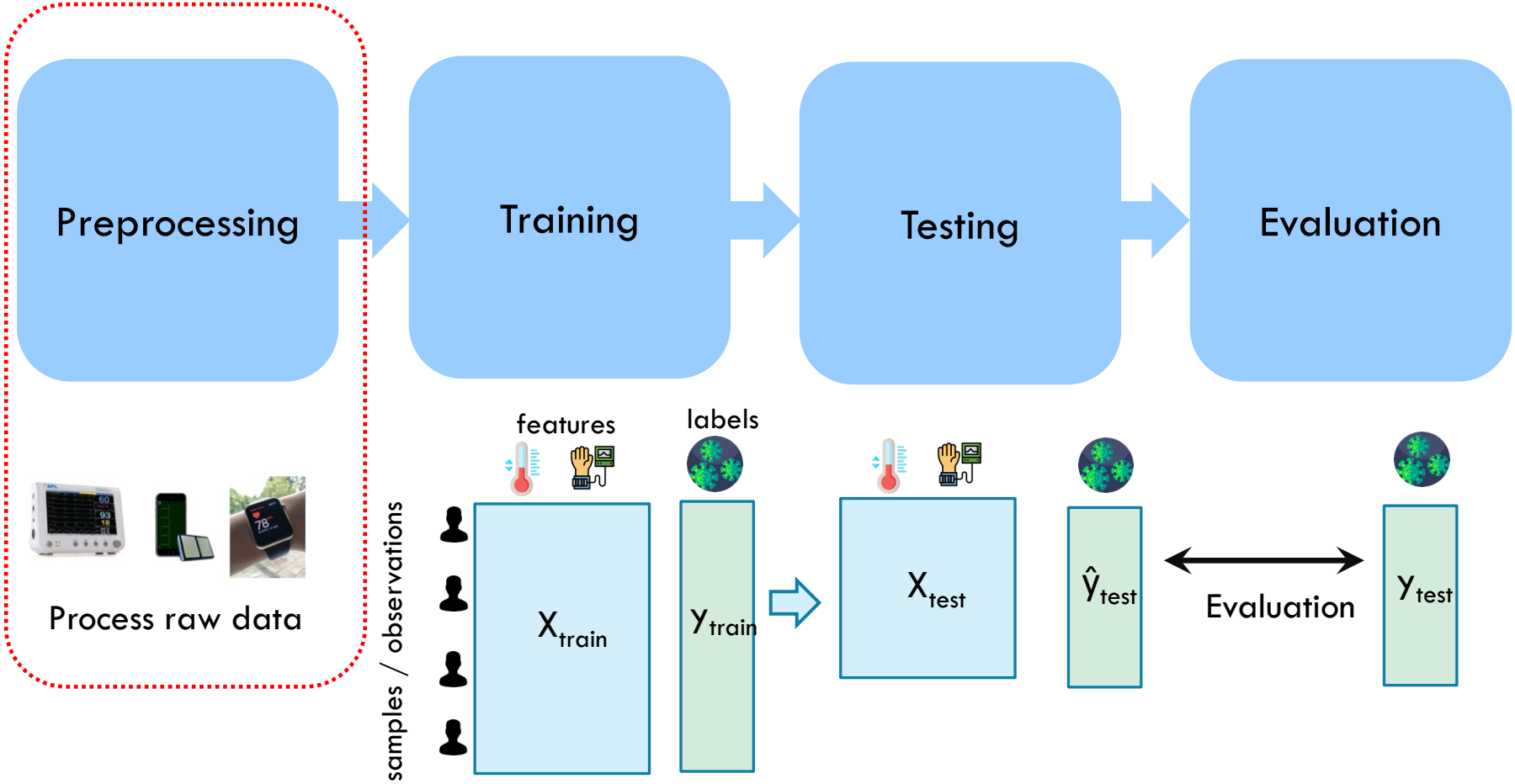


Regression: predict *numeric* labels, given training data

Typical Machine Learning Pipeline



Typical Machine Learning Pipeline



Data Quality

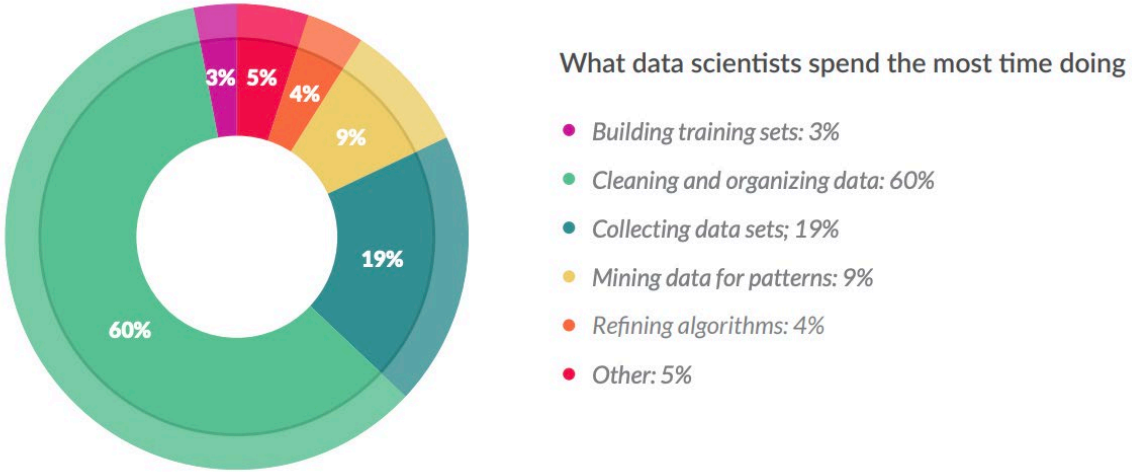
The most important point is that poor data quality is an unfolding disaster.

Poor data quality costs the typical company at least 10% of revenue; 20% is probably a better estimate.

Thomas C. Redman, DM Review
August 2004

Data Preprocessing

- Often an under-valued part of data science, but very important
  - "Garbage in, garbage out"
  - Google AI (*SIGCHI* 2021): 92% of the analysed high-stakes AI projects practitioners report negative downstream effects from data issues
  - Anecdotally on Kaggle (prediction contest site), feature engineering is often one of the key important factors for winning contests
- NY Tir[illegible] data scientists spend 50-80% of their time on data prepar[illegible]



https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html
https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf

Data Quality: Missing Values

Why is data missing?

- Information was not collected: e.g. people decline to give weight
- *Missing at random*: missing values are randomly distributed. If data is instead *missing not at random*: then the missingness itself may be important information.

How to handle missing values?

- Eliminate objects (rows) with missing values
- Or: fill in the missing values ("imputation")
  - E.g. based on the **mean** / **median** of that attribute
  - Or: by fitting a **regression** model to predict that attribute given other attributes

UserID	Height (m)	Country	...
1	1.61	SG	...
2	1.50	US	...
3	***NA***	MY	...
...	...	...	...

Median / Regression Imputation

UserID	Height (m)	Country	...
1	1.61	SG	...
2	1.50	US	...
3	***1.55***	MY	...
...	...	...	...

Data Quality: Missing Values

Why is data missing?

- Information was not collected: e.g. people decline to give weight
- *Missing at random*: missing values are randomly distributed. If data is instead *missing not at random*: then the missingness itself may be important information.

How to handle missing values?

- Eliminate objects (rows) with missing values
- Or: fill in the missing values ("imputation")
  - E.g. based on the **mean** / **median** of that attribute
  - Or: by fitting a **regression** model to predict that attribute given other attributes
  - **Dummy variables**: optionally insert a column which is 1 if the variable was missing, and 0 otherwise

UserID	Height (m)	Country	...
1	1.61	SG	...
2	1.50	US	...
3	***NA***	MY	...
...	...	...	...

Dummy Variable Imputation

UserID	Height (m)	Missing?	Country	...
1	1.61	0	SG	...
2	1.50	0	US	...
3	***1.55***	1	MY	...
...	...		...	...

Data Quality: Missing Values

Why is data missing?

- Information was not collected: e.g. people decline to give weight
- *Missing at random*: missing values are randomly distributed. If data is instead *missing not at random*: then the missingness itself may be important information.

How to handle missing values?

- Eliminate objects (rows) with missing values
- Or: fill in the missing values ("imputation")
  - E.g. based on the **mean** / **median** of that attribute
  - Or: by fitting a **regression** model to predict that attribute given other attributes
  - **Dummy variables**: optionally insert a column which is 1 if the variable was missing, and 0 otherwise

UserID	Height (m)	Country	...
1	1.61	SG	...
2	1.50	US	...
3	***NA***	MY	...
...	...	...	...

Dummy Variable Imputation

UserID	Height (m)	Missing?	Country	...
1	1.61	0	SG	...
2	1.50	0	US	...
3	***1.55***	1	MY	...
...	...		...	...

In PySpark:

```
from pyspark.ml.feature import Imputer

imputer = Imputer(inputCols=["a", "b"],outputCols=["out_a", "out_b"])
model = imputer.fit(df)
model.transform(df).show()
```

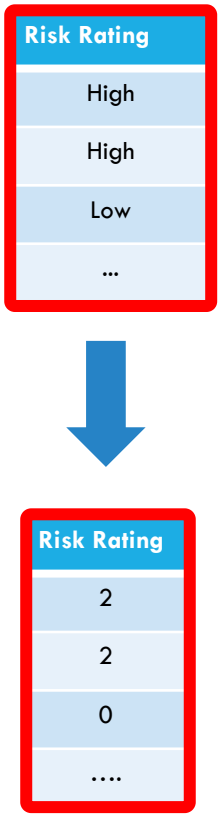
Categorical Encoding

Convert categorical feature to numerical features.

Numerical values are often assigned in a way that represents the ordinal relationship or inherent order among the categories

E.g. the risk rating [Low, Medium, High] will be converted into [0, 1, 2]

This lets us apply algorithms which can handle numerical features (e.g. linear regression)

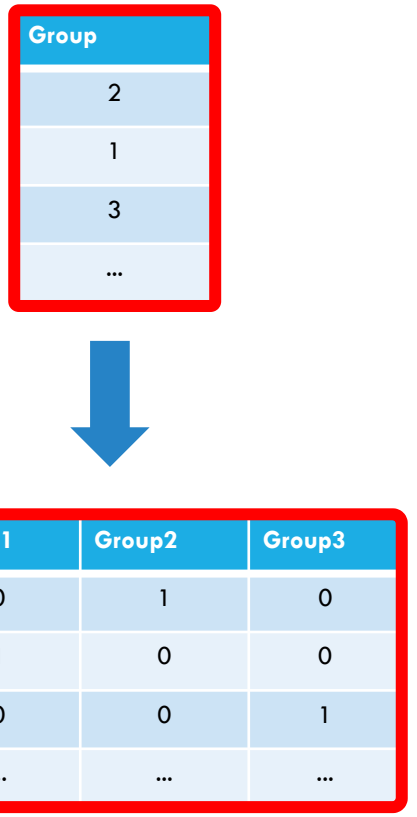


One Hot Encoding

Convert discrete feature to a series of binary features.

E.g. the first record has group 2, so we set its 2nd binary feature to 1, and all the rest to 0.

This method is useful when there's no ordinal relationship among categories, and you want to ensure that the categorical variable doesn't imply any numerical relationship.

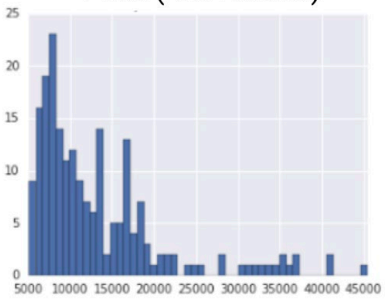


Group
2
1
3
...

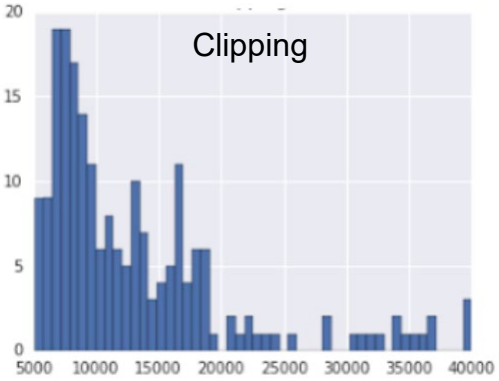
Group1	Group2	Group3
0	1	0
1	0	0
0	0	1
...	...	...

Normalization

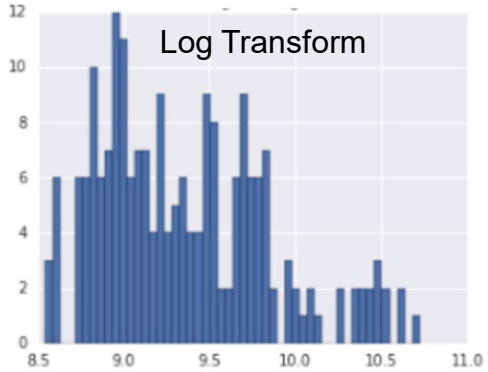
Price (raw feature)

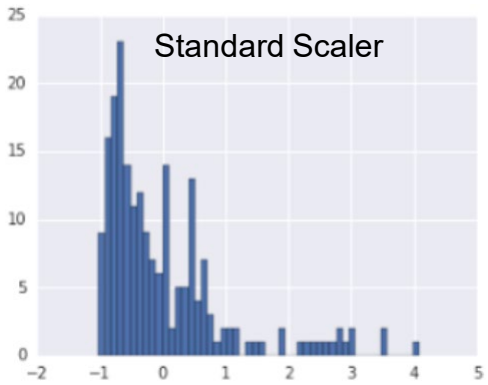


$\text{clip}(x)$

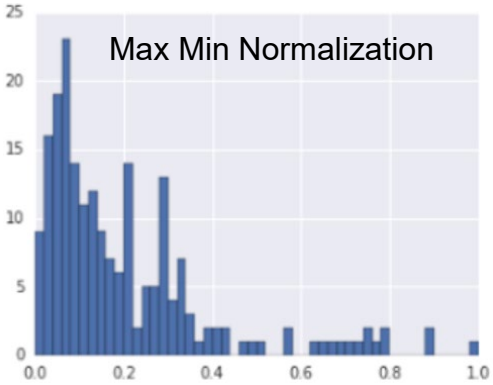


$\log(1+x)$



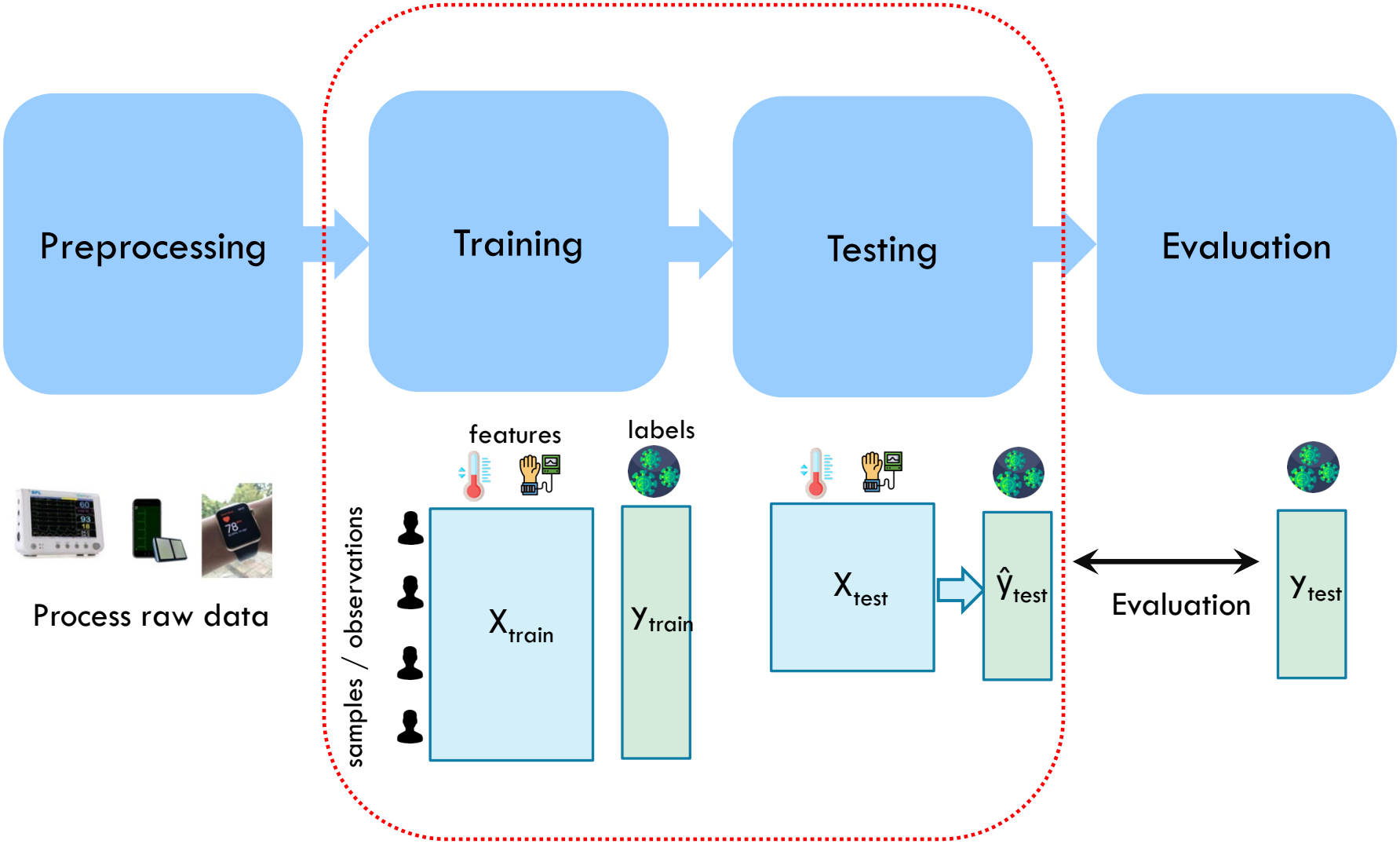


$$\frac{x - mean(x)}{std(x)}$$



$$\frac{x - \min(x)}{\max(x) - \min(x)}$$

Typical Machine Learning Pipeline

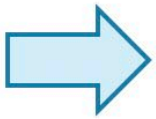


Running Example: Spam Classification

Features

Number of '$'	Number of '!'
5	2

$$x = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$$



(Predicted probability of the email being spam)

?

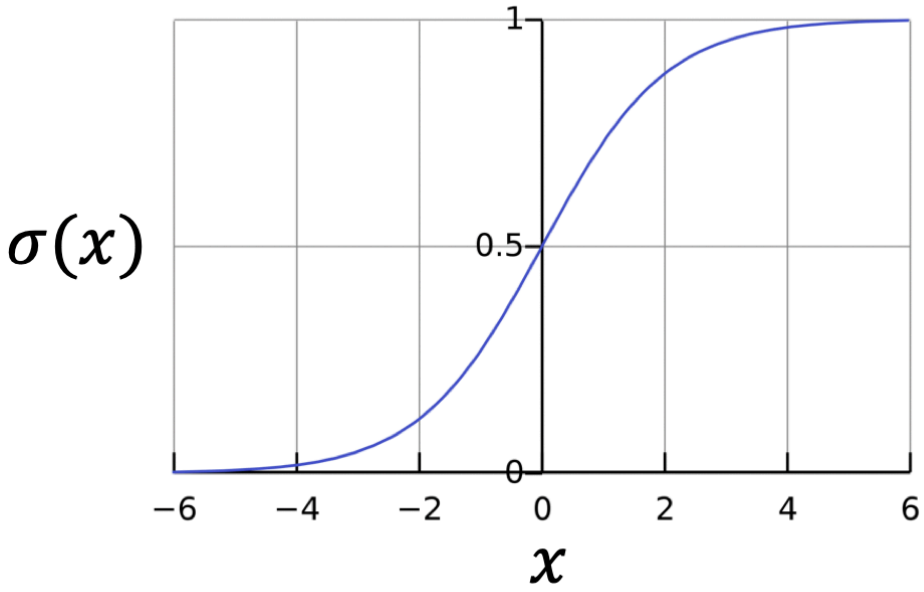
$\hat{y}$

Logistic Regression: Sigmoid Function

The sigmoid function $\sigma(x)$ maps the real numbers to the range $(0, 1)$

It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Logistic Regression: Sigmoid Function

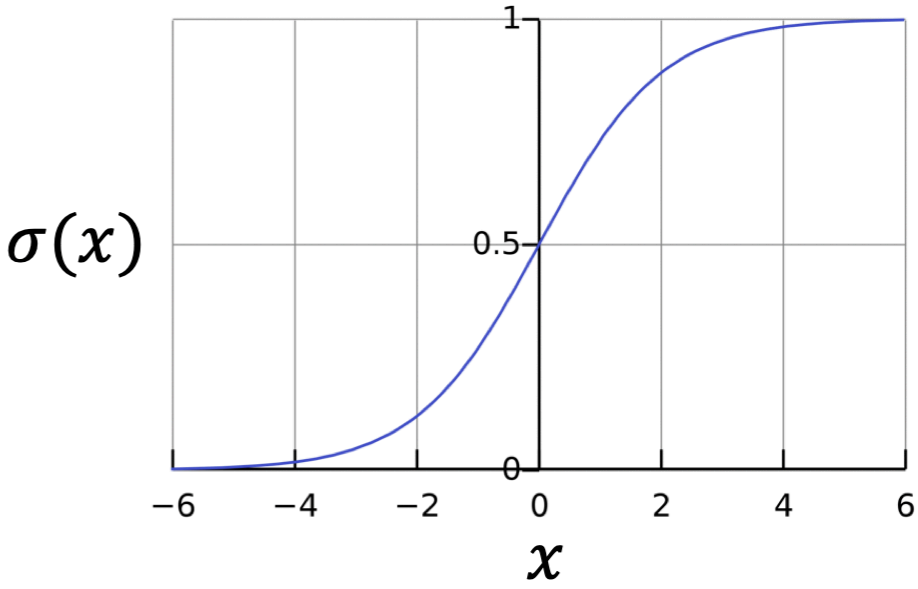
The sigmoid function $\sigma(x)$ maps the real numbers to the range $(0, 1)$

It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Check:

As $x \to -\infty, \sigma(x) \to \frac{1}{1+e^{\infty}} = 0$



Logistic Regression: Sigmoid Function

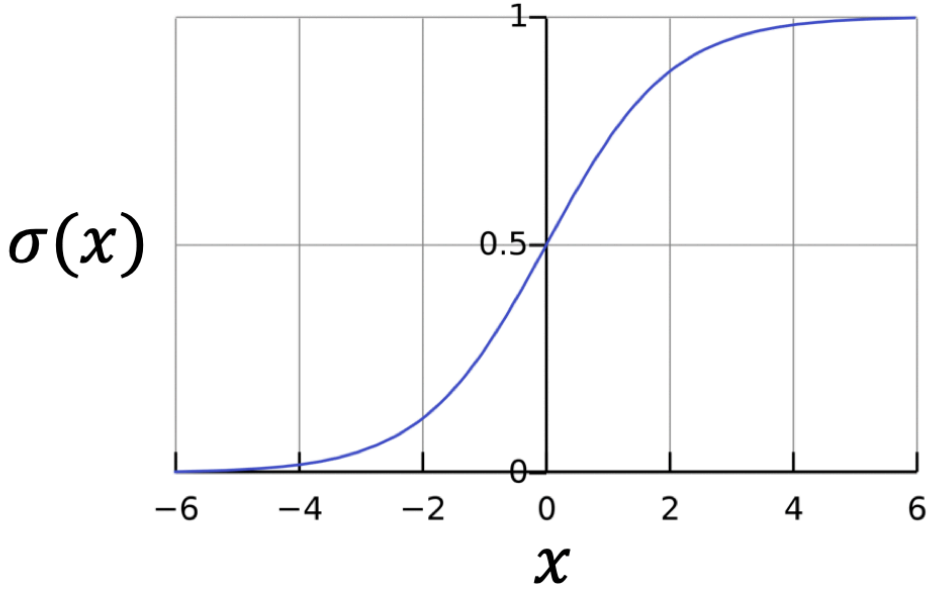
The sigmoid function $\sigma(x)$ maps the real numbers to the range $(0, 1)$

It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Check:

$$\sigma(0) = \frac{1}{1+e^{0}} = 0.5$$



Logistic Regression: Sigmoid Function

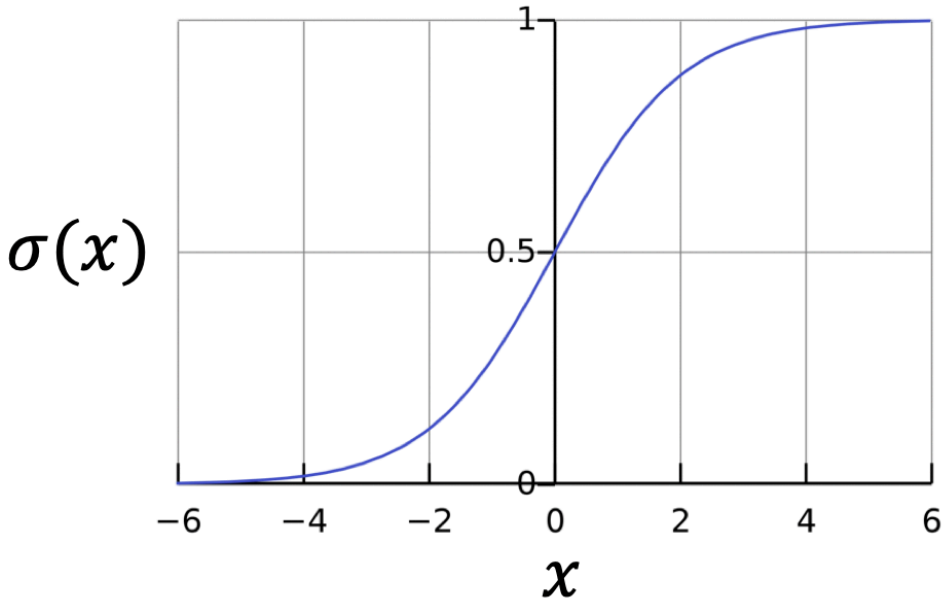
The sigmoid function $\sigma(x)$ maps the real numbers to the range $(0, 1)$

It is defined as:

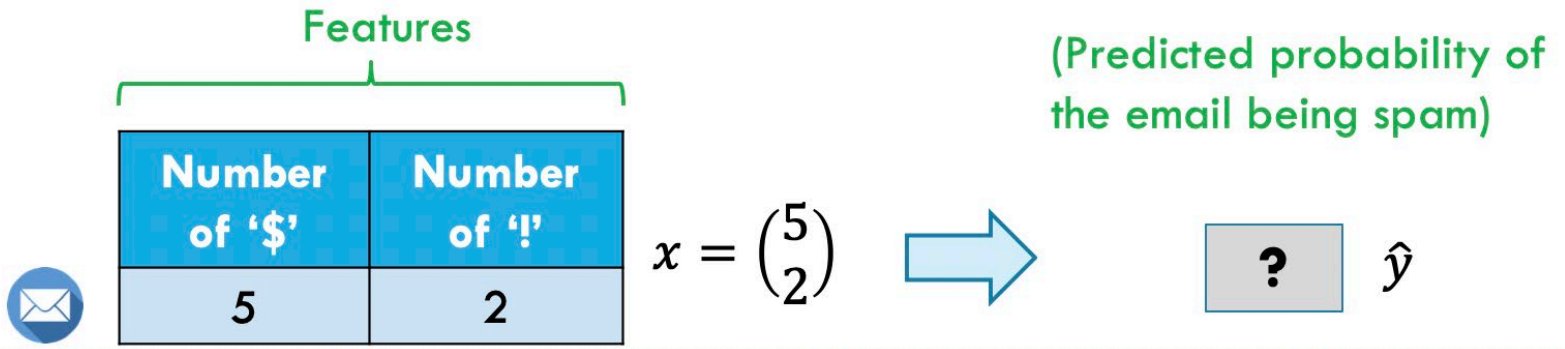
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Check:

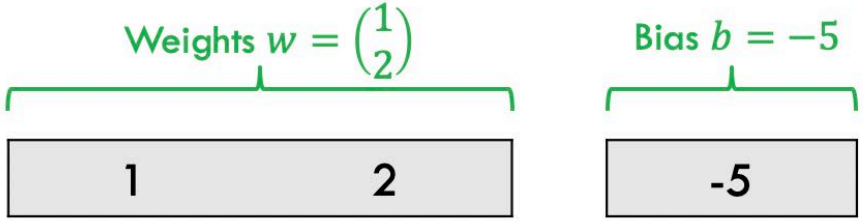
As $x \to \infty, \sigma(x) \to \frac{1}{1+e^{-\infty}} = 1$



Logistic Regression: Parameters



Parameters:



Logistic Regression: Prediction

Features

Number of '$'	Number of '!'
5	2

$x = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$

(Predicted probability of the email being spam)

? $\hat{y}$

Parameters:

Weights $w = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

1	2

Bias $b = -5$

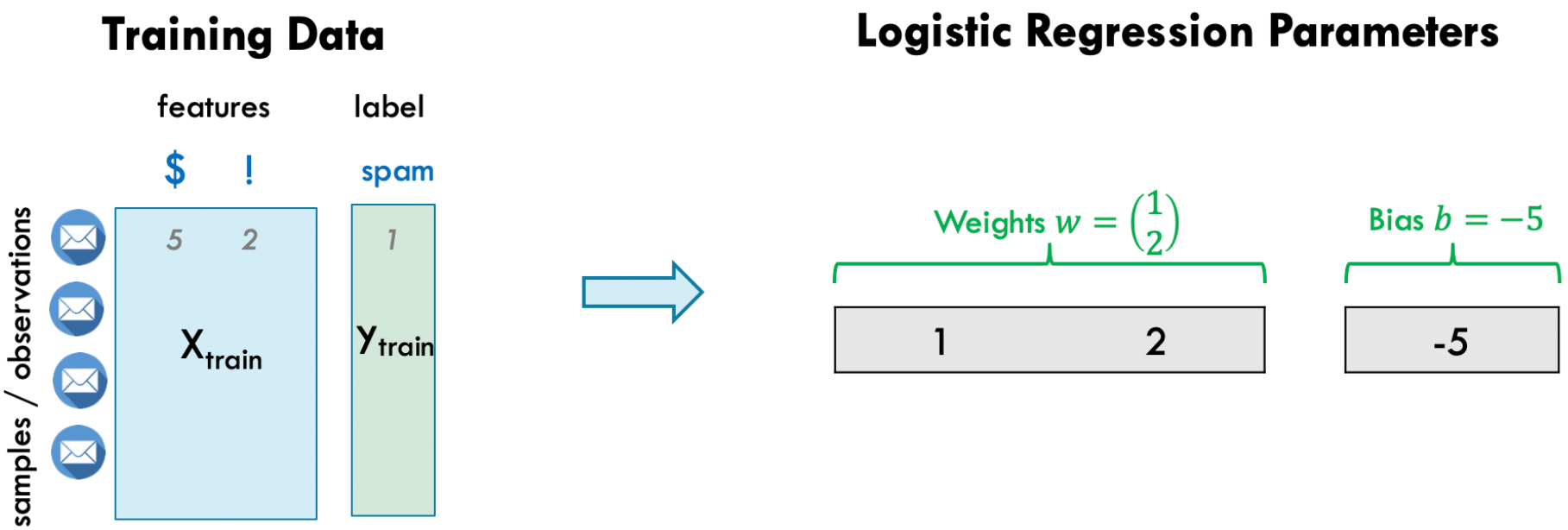
-5

Prediction:

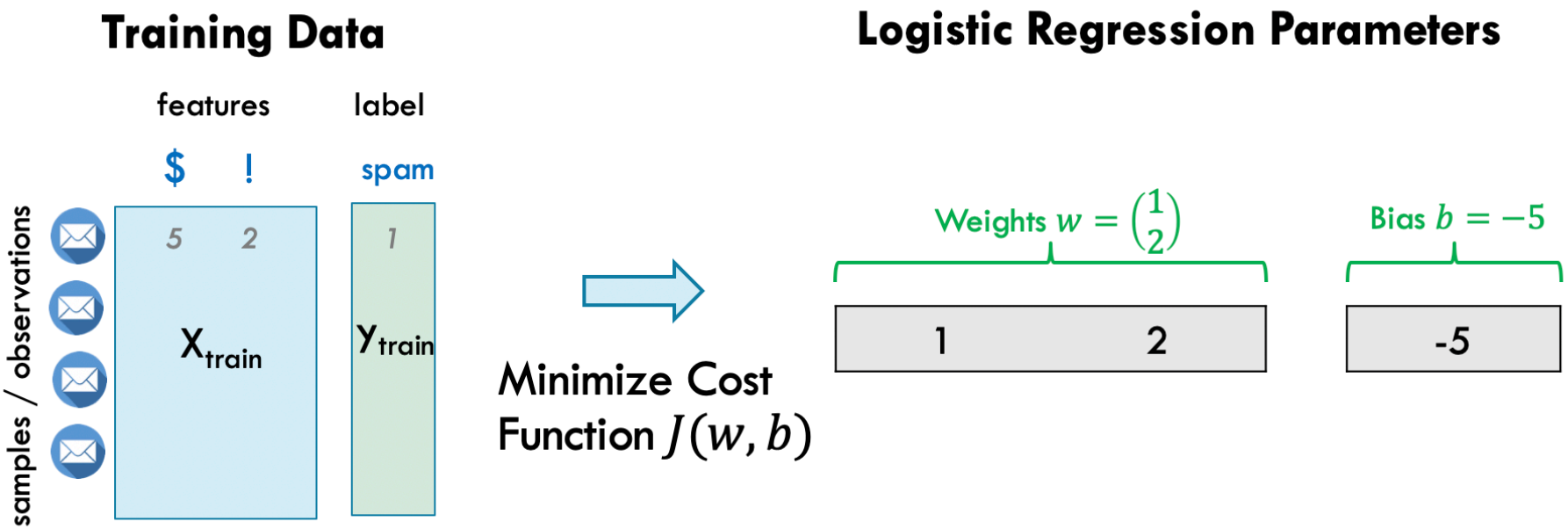
Sigmoid function, Dot product

$$\hat{y} = \sigma(x \cdot w + b) = \sigma\left(\begin{pmatrix} 5 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} - 5\right) = \sigma(9 - 5) = \frac{1}{1 + e^{-4}} = 0.982$$

Training Logistic Regression



Training Logistic Regression

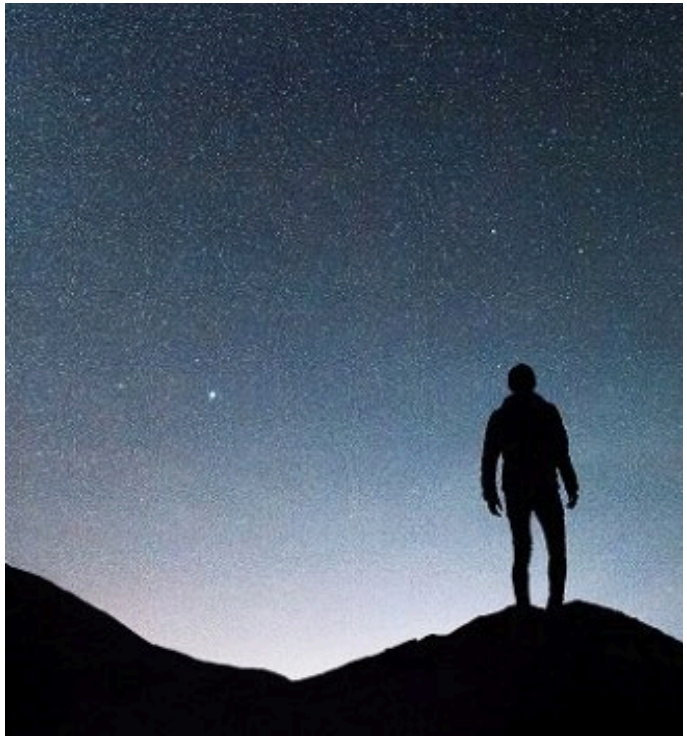


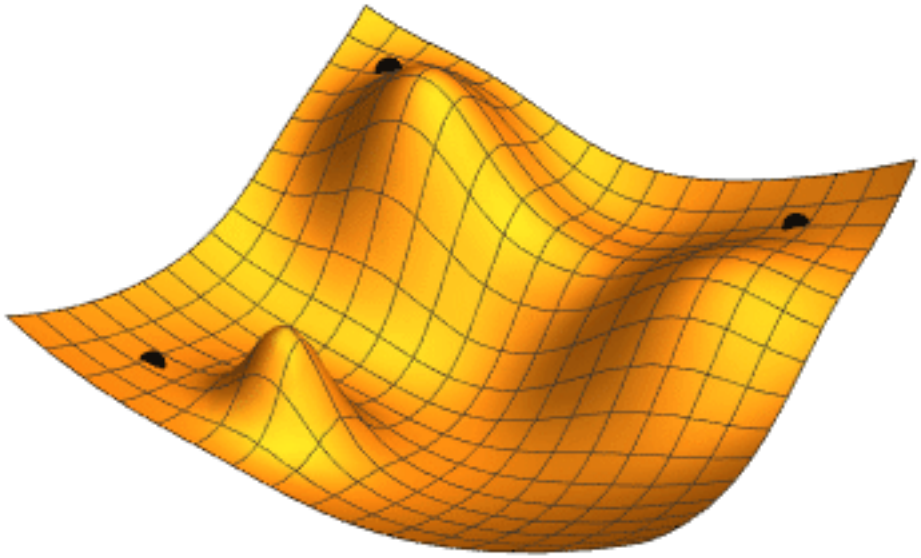
Big Picture: machine learning involves fitting the ***parameters*** of a model (here *w, b*) by minimizing a ***loss / cost function***.

Here, the cost function J is ***Cross Entropy Loss*** (intuitively: think of the model's predictions as a probability. The closer the prediction probability to the label the lower the loss value.

Minimizing cost function J using gradient descent

Minimizing cost function J using gradient descent

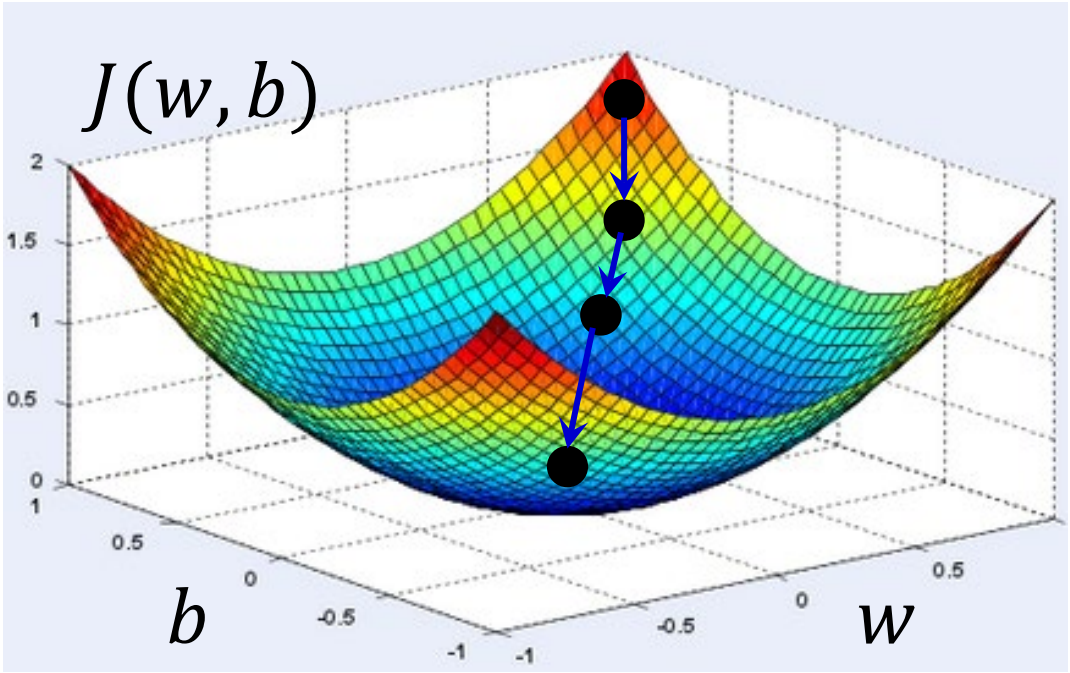




Minimizing cost function J using gradient descent

Optional

- We want to find w, b to minimize $J(w, b)$
- Start at an arbitrary point, then move following the steepest downward slope (‘*gradient*’)
- Continue until convergence
  - Stop when improvement in J is below a fixed threshold



Gradient Descent (assume bias b=0)

Optional

- Given n training samples with d features:
  - Update Rule (from iteration i to iteration i+1):

Step Size

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \nabla J(\mathbf{w}_i)$$

Gradient / Slope

  - Weights Vector Update:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \sum_{j=1}^{n} \left[\sigma(\mathbf{x}^{(j)} \cdot \mathbf{w}_i) - y^{(j)}\right] \mathbf{x}^{(j)}$$

Parallel Gradient Descent

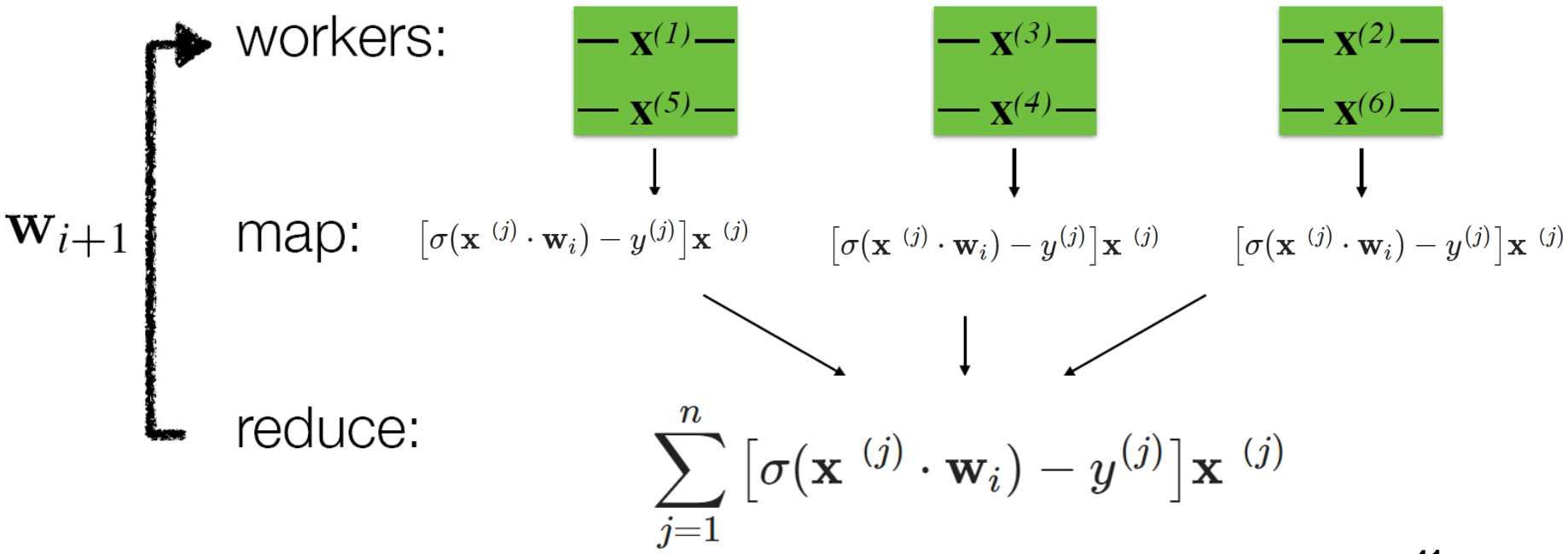
Optional

Vector Update: $\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \sum_{j=1}^{n} \left[\sigma(\mathbf{x}^{(j)} \cdot \mathbf{w}_i) - y^{(j)}\right] \mathbf{x}^{(j)}$

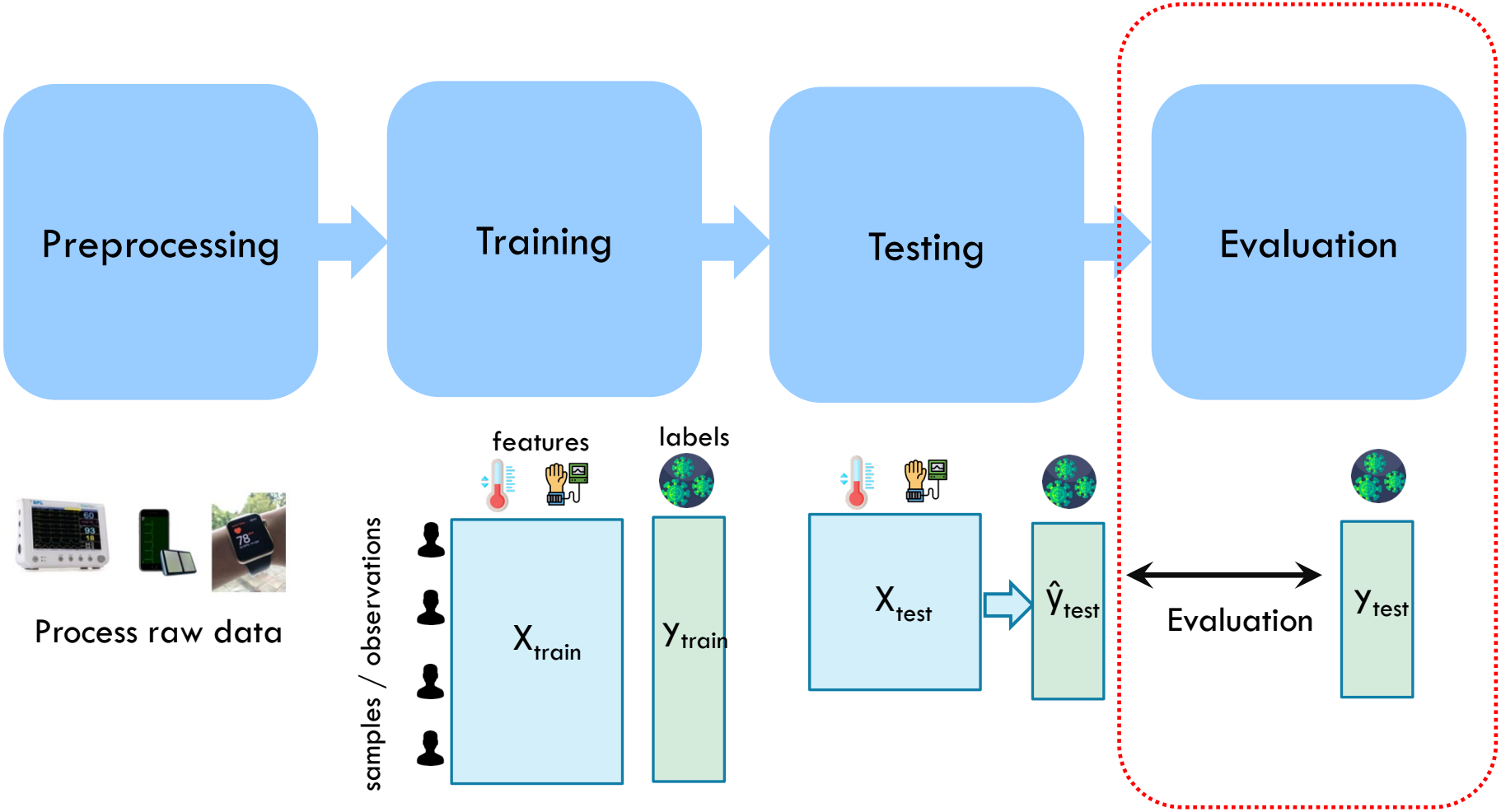
Weights (broadcast variable)

Cache Training Data

Example: $n = 6$; 3 workers



Typical Machine Learning Pipeline



THE STRAITS TIMES

Fast Covid-19 tests to be added to Singapore's testing arsenal: How do antigen rapid tests work?

The Health Ministry will be turning to Covid-19 antigen rapid tests to complement existing polymerase chain reaction tests – the gold standard for testing – as Singapore further opens its economy.

Clara Chong

PUBLISHED OCT 21, 2020, 5:00 AM SGT

Benefits

• Faster (less than 30 minutes) and hence more feasible for pre-event testing.

• Cheaper.

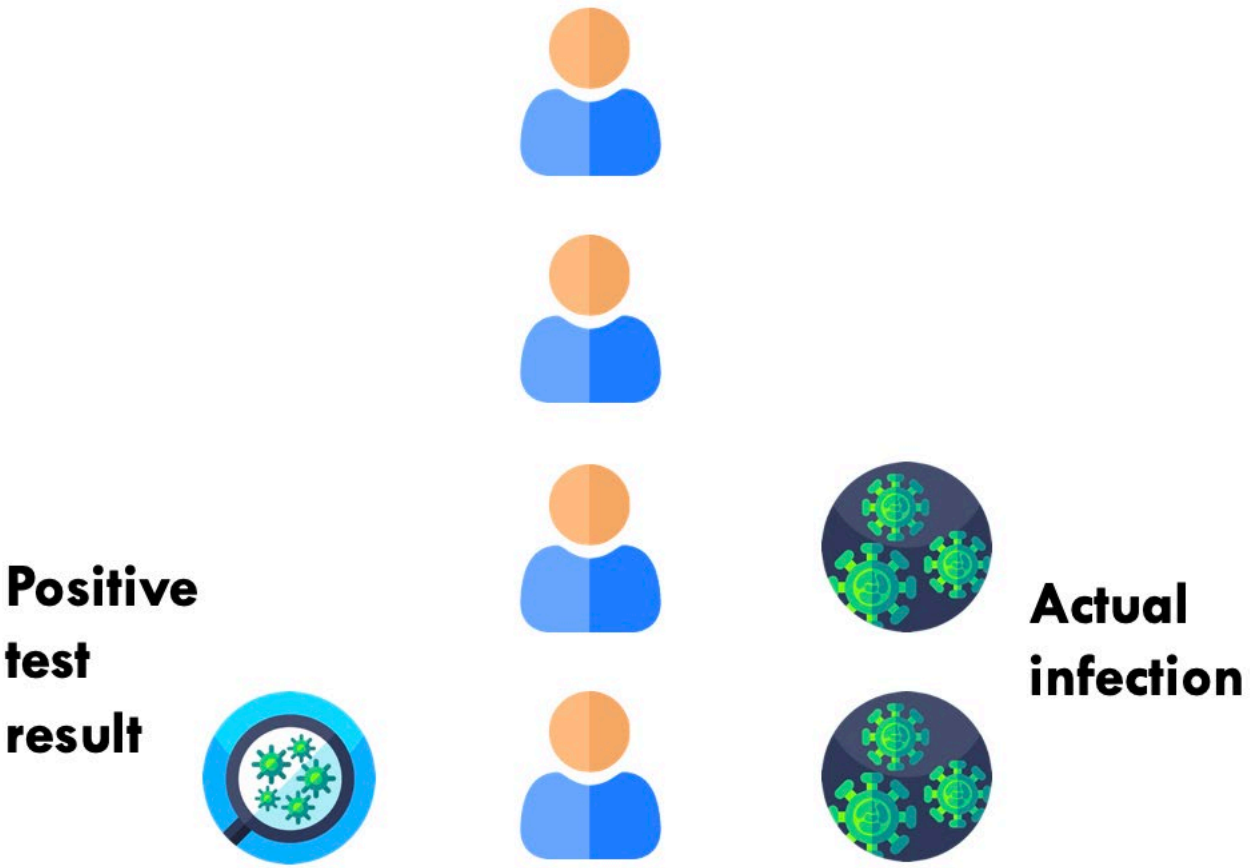
• Easier to administer.

Drawbacks

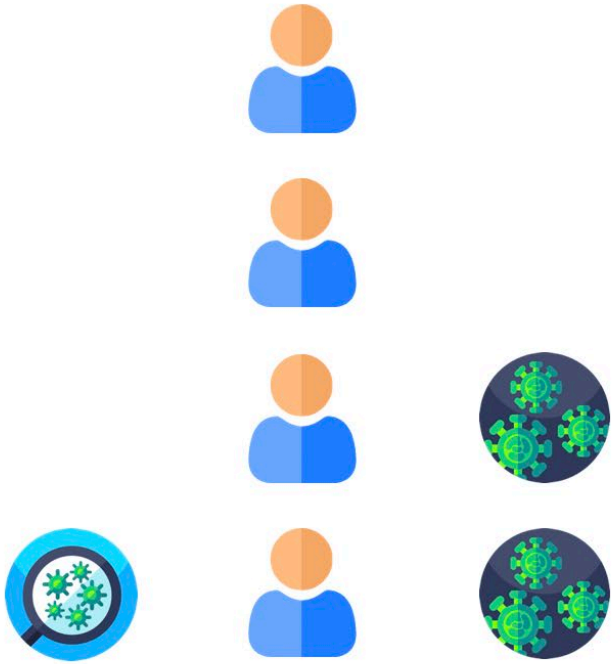
• Tests have lower sensitivity and specificity, and may carry a higher risk of false positives and false negatives.

• The World Health Organisation recommends at least 80 per cent sensitivity and 97 per cent specificity. This means that at least 80 per cent of those infected are identified, and at most 3 per cent of healthy subjects are tested false positives.

Binary Classification Setting



Binary Classification Setting



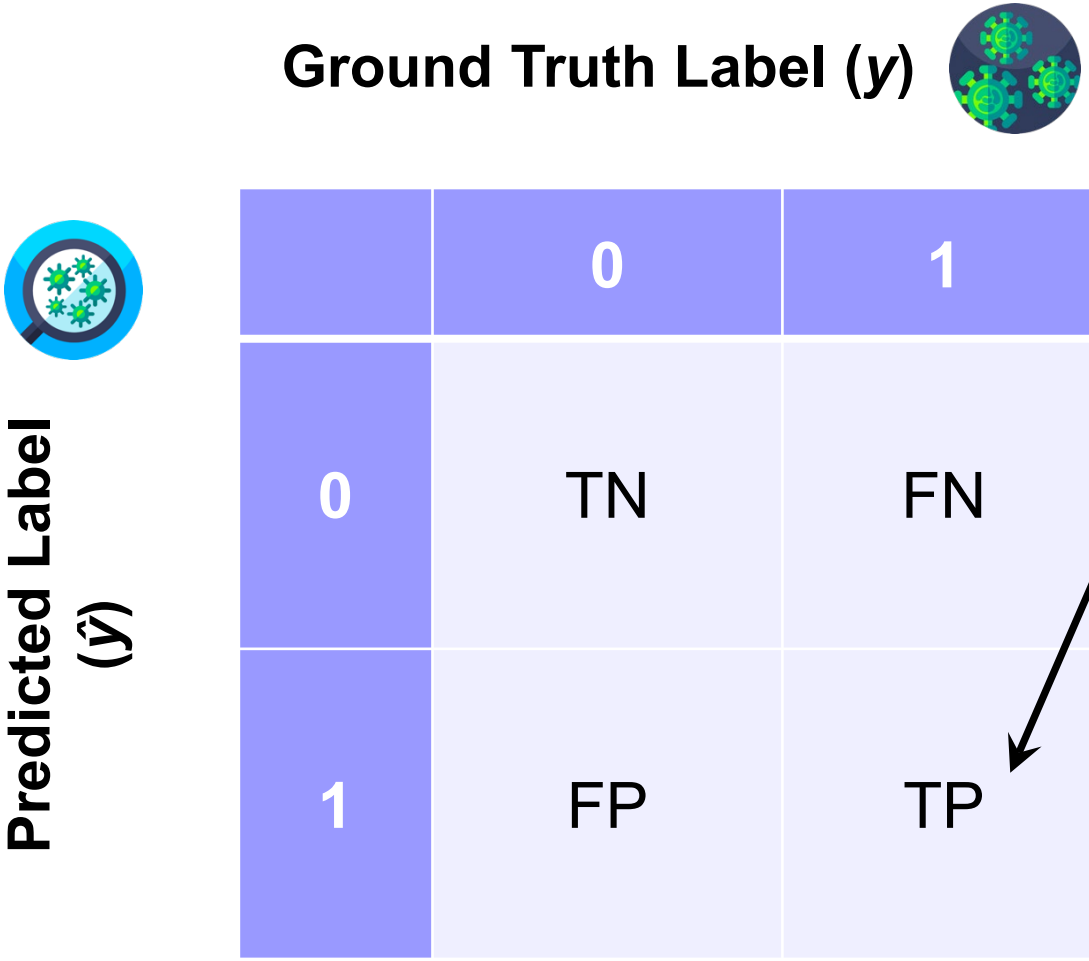
Predicted Label ($\hat{y}$)	Ground Truth Label (y)
0	0
0	0
0	1
1	1

Confusion Matrix

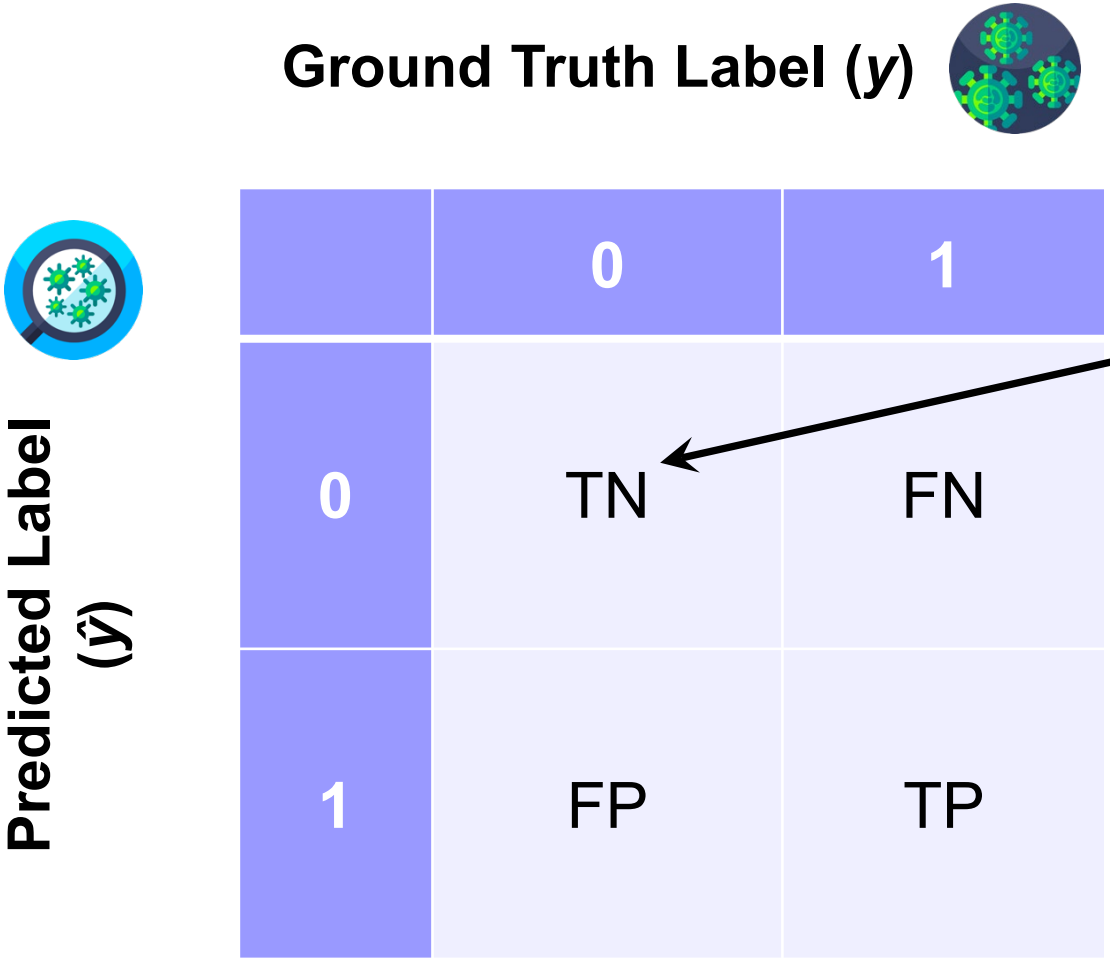
Predicted Label ($\hat{y}$)	Ground Truth Label (y)
0	0
0	0
0	1
1	1

Predicted Label ($\hat{y}$)	Ground Truth Label (y) = 0	Ground Truth Label (y) = 1
0	2	1
1	0	1

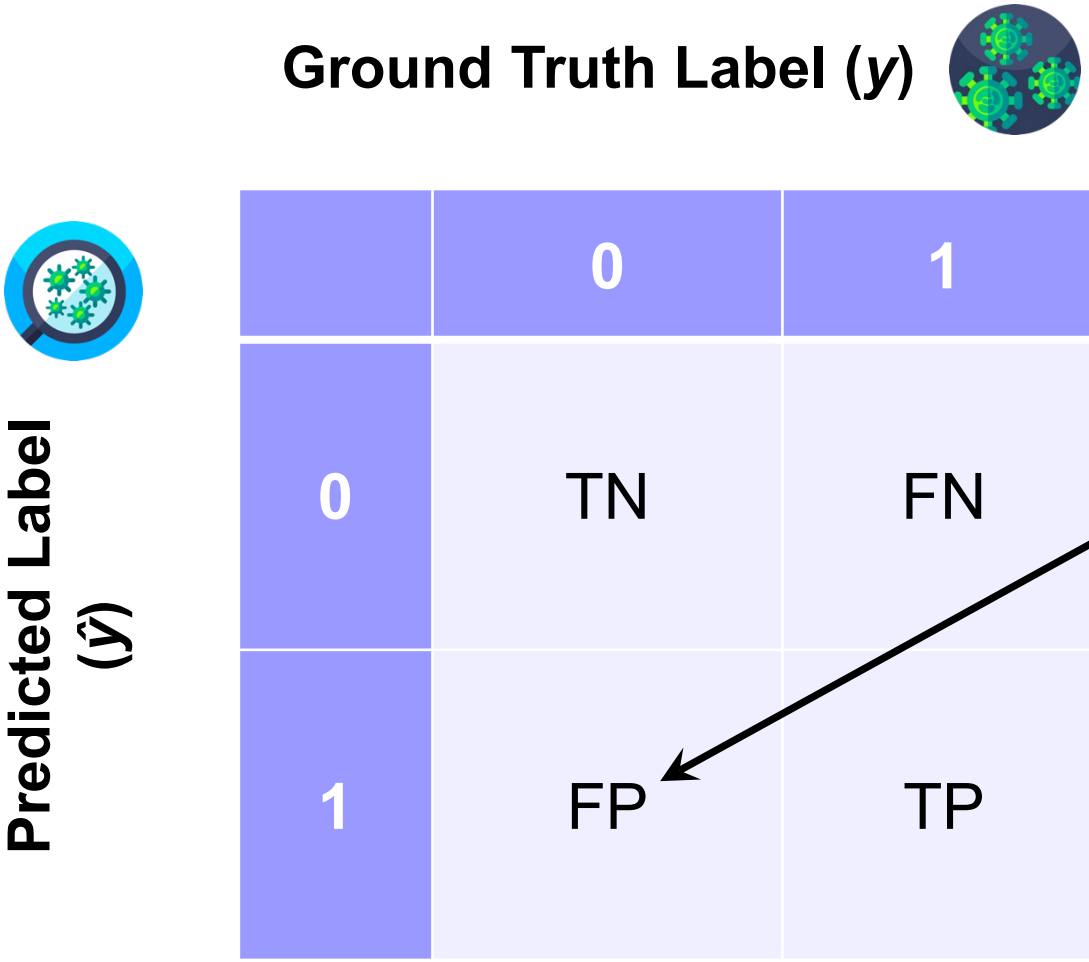
True/False Positives/Negatives



True/False Positives/Negatives



True/False Positives/Negatives



True Positive: test correctly output positive
True Negative: test correctly output negative
False Positive: test wrongly outputs positive

True/False Positives/Negatives

Ground Truth Label (*y*)

Predicted Label ($\hat{y}$)

	0	1
0	TN	FN
1	FP	TP

True Positive: test correctly output positive
True Negative: test correctly output negative
False Positive: test wrongly outputs positive
False Negative: test wrongly outputs negative

Accuracy

Ground Truth Label (*y*)

	0	1
0	TN	FN
1	FP	TP

Accuracy: fraction of correct predictions

Sensitivity / Specificity

Ground Truth Label (*y*)

	0	1
0	TN	FN
1	FP	TP

Sensitivity: fraction of positive cases that are detected

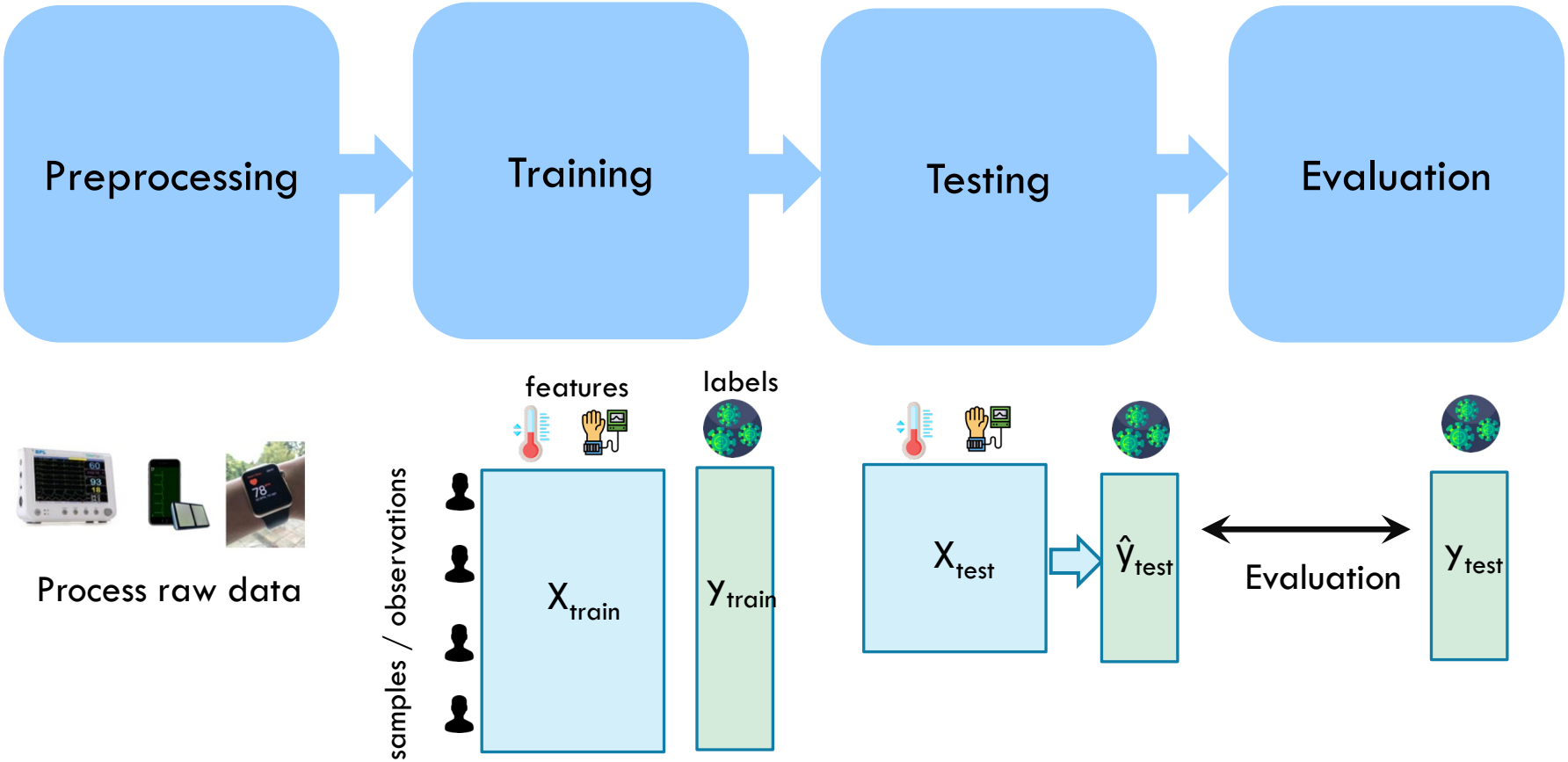
Sensitivity / Specificity

	Ground Truth Label (*y*) = 0	Ground Truth Label (*y*) = 1
Model Prediction = 0	TN	FN
Model Prediction = 1	FP	TP

Sensitivity: fraction of positive cases that are detected

Specificity: fraction of actual negatives that are correctly identified

Typical Machine Learning Pipeline



Pipelines

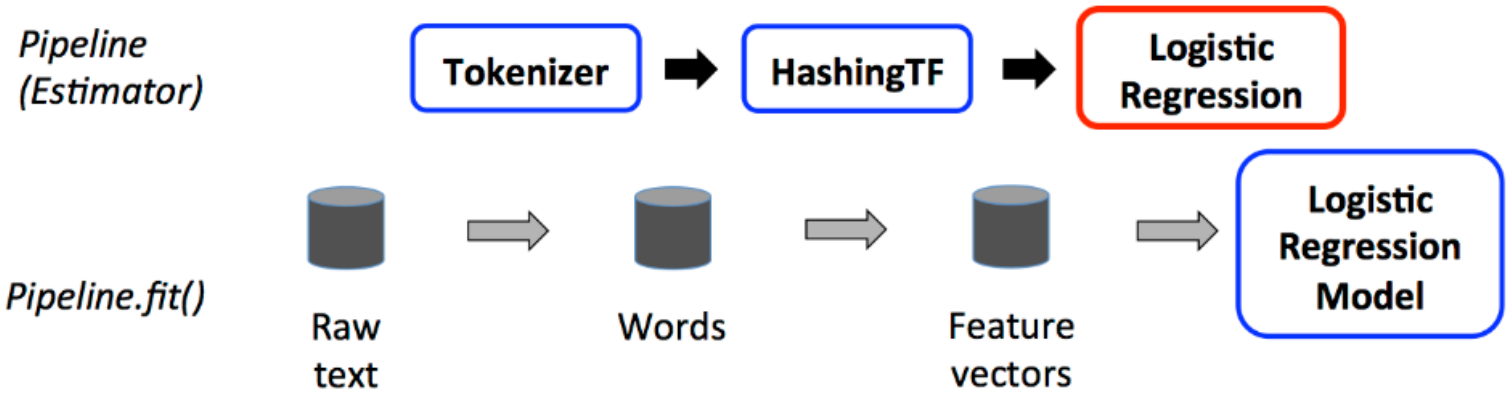
Idea: building complex pipeline out of simple building blocks
(Note: scikit-learn pipelines are basically the same as Spark MLLib ones)

Pipelines

Idea: building complex pipeline out of simple building blocks: e.g. encoding, normalization, feature transformation, model fitting.

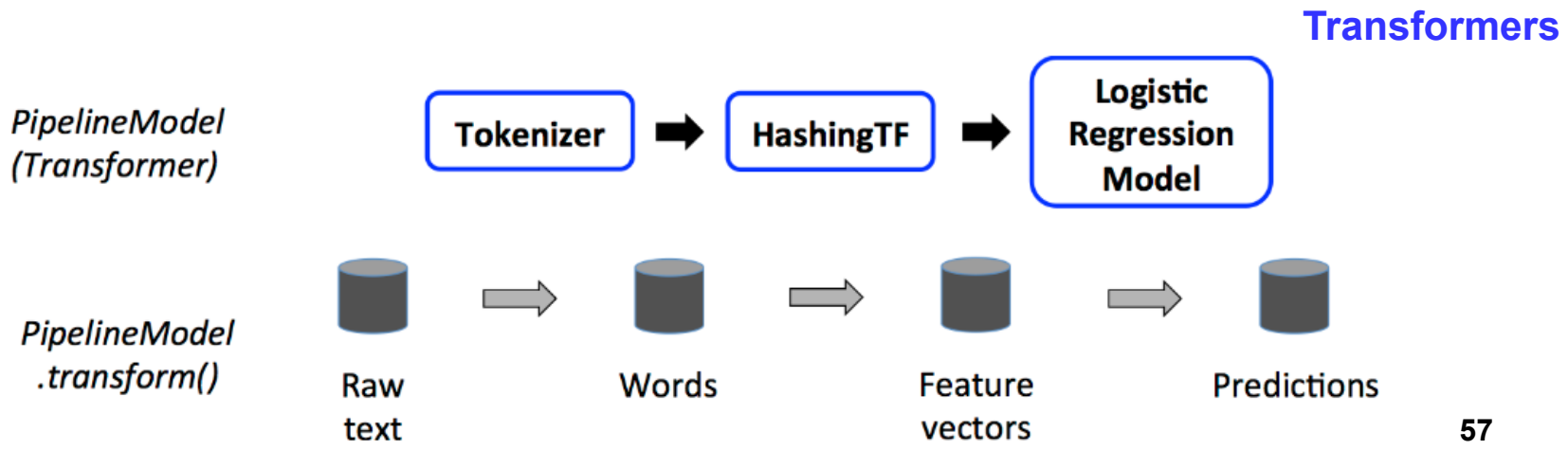
Why?

- Better code reuse: without pipelines, we would repeat a lot of code, e.g. between the training and test pipelines, cross-validation, model variants, etc.
- Easier to perform cross validation, and hyperparameter tuning.



Building Blocks: Transformers

- **Transformers** are for mapping DataFrames to DataFrames
  - Examples: one-hot encoding, tokenization
  - Specifically, a Transformer object has a transform() method, which performs its transformation
- Generally, these transformers output a new DataFrame which ***append*** their result to the original DataFrame.
  - Similarly, a fitted model (e.g. logistic regression) is a Transformer that transforms a DataFrame into one with the predictions appended.



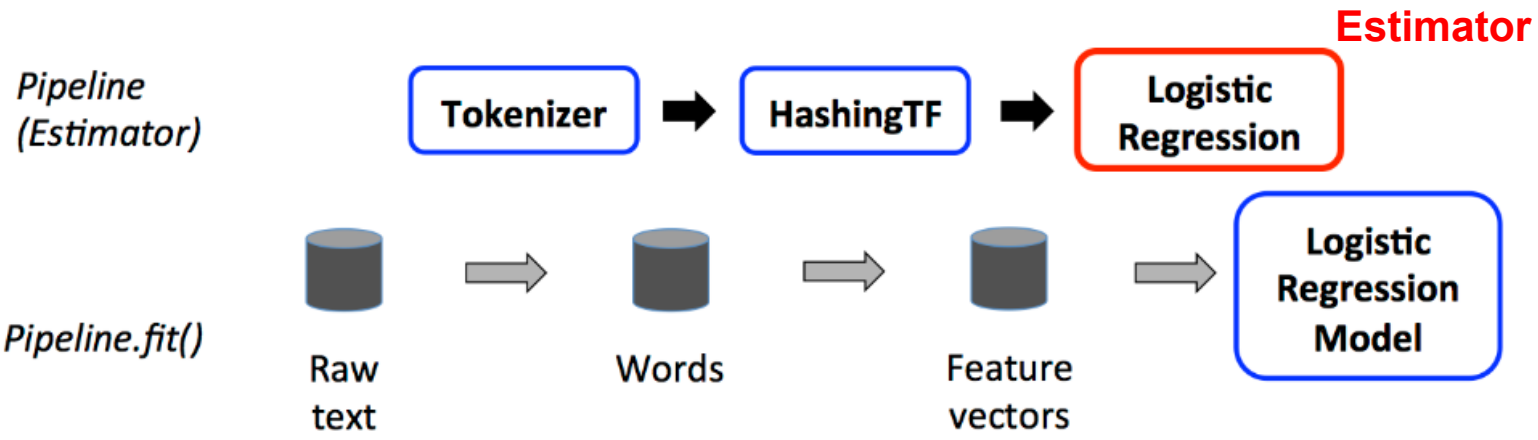
Building Blocks: Estimator

- **Estimator** is an algorithm which takes in data, and outputs a fitted model. For example, a learning algorithm (the LogisticRegression object) can be fit to data, producing the trained logistic regression model.
- They have a fit() method, which returns a Transformer.

```
from pyspark.ml.classification import LogisticRegression

training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lr = LogisticRegression(maxIter=10)

lrModel = lr.fit(training)

print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

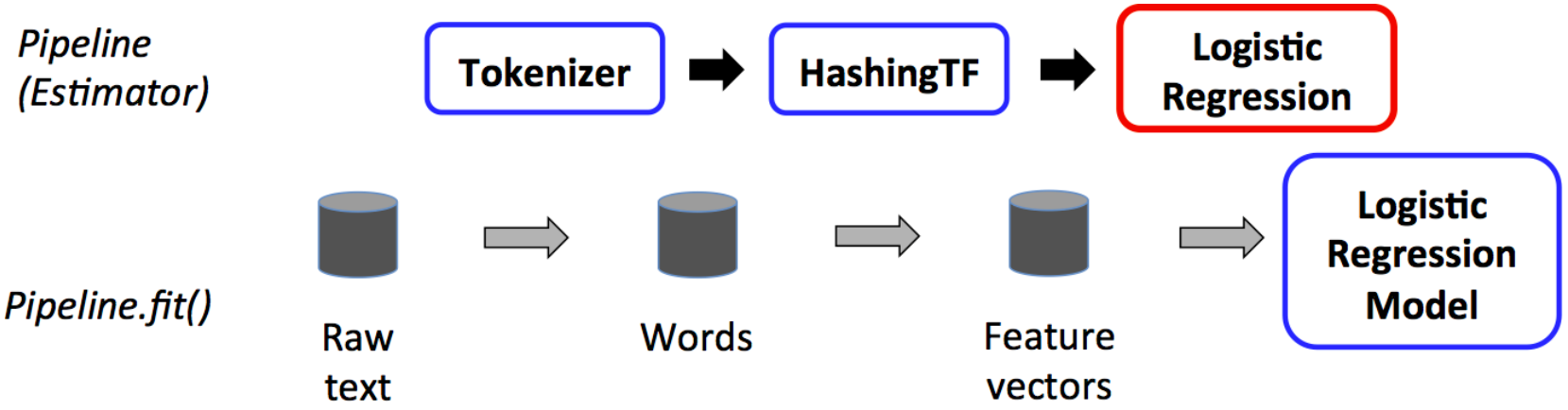
Building Blocks: Estimator

- **Estimator** is an algorithm which takes in data, and outputs a fitted model. For example, a learning algorithm (the LogisticRegression object) can be fit to data, producing the trained logistic regression model.
- They have a fit() method, which returns a Transformer.



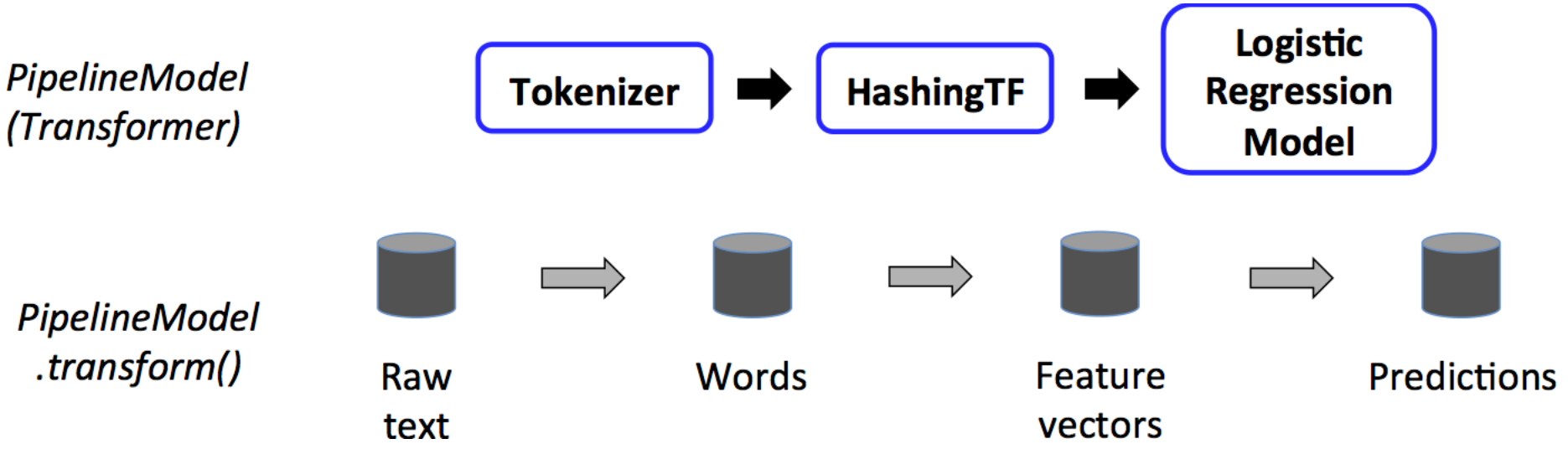
Pipeline: Training Time

- A pipeline chains together multiple Transformers and Estimators to form an ML workflow.
- Pipeline is an Estimator. When Pipeline.fit() is called:
  - Starting from the beginning of the pipeline:
  - For Transformers, it calls transform()
  - For Estimators, it calls fit() to fit the data and returns a fitted model



Pipeline: Test Time

- The output of Pipeline.fit() is the estimated pipeline model (of type PipelineModel).
  - It is a transformer, and consists of a series of Transformers.
  - When its transform() is called, each stage's transform() method is called.



Demo_3: Machine Learning Pipeline

```
# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])
```

```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

```
# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

```
1  # Make predictions on train documents and print columns of interest.
2  pred_train = model.transform(training)
3  pred_train.drop('rawPrediction').show(truncate = False)
```

▸ (3) Spark Jobs

▸ pred_train: pyspark.sql.dataframe.DataFrame = [id: long, text: string ... 6 more fields]

```
+---+---------------+-----+--------------------+--------------------------------------------------------------------------+----------------------------------------+----------+
|id |text           |label|words               |features                                                                  |probability                             |prediction|
+---+---------------+-----+--------------------+--------------------------------------------------------------------------+----------------------------------------+----------+
|0  |a b c d e spark|1.0  |[a, b, c, d, e, spark]|(262144,[74920,89530,107107,148981,167694,173558],[1.0,1.0,1.0,1.0,1.0,1.0])|[0.002628213496942035,0.9973717865030579]|1.0       |
|1  |b d            |0.0  |[b, d]              |(262144,[89530,148981],[1.0,1.0])                                         |[0.9963902711801113,0.0036097288198887467]|0.0       |
|2  |spark f g h    |1.0  |[spark, f, g, h]    |(262144,[36803,173558,209078,228158],[1.0,1.0,1.0,1.0])                   |[0.0022081050570269896,0.997791894942973]|1.0       |
|3  |hadoop mapreduce|0.0 |[hadoop, mapreduce] |(262144,[132966,198017],[1.0,1.0])                                        |[0.9987232337063715,0.0012767662936284951]|0.0       |
+---+---------------+-----+--------------------+--------------------------------------------------------------------------+----------------------------------------+----------+
```

```
# Prepare test documents
test = spark.createDataFrame([
    (4, "spark i j k", 1.0),
    (5, "l m n", 0.0),
    (6, "spark hadoop spark", 1.0),
    (7, "apache hadoop", 0.0)
], ["id", "text", "label"])
```

```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

```
# Make predictions on test documents and print columns of interest.
pred_test = model.transform(test)
pred_test.drop('rawPrediction').show(truncate = False)
```

▸ (3) Spark Jobs

▸ pred_test: pyspark.sql.dataframe.DataFrame = [id: long, text: string ... 6 more fields]

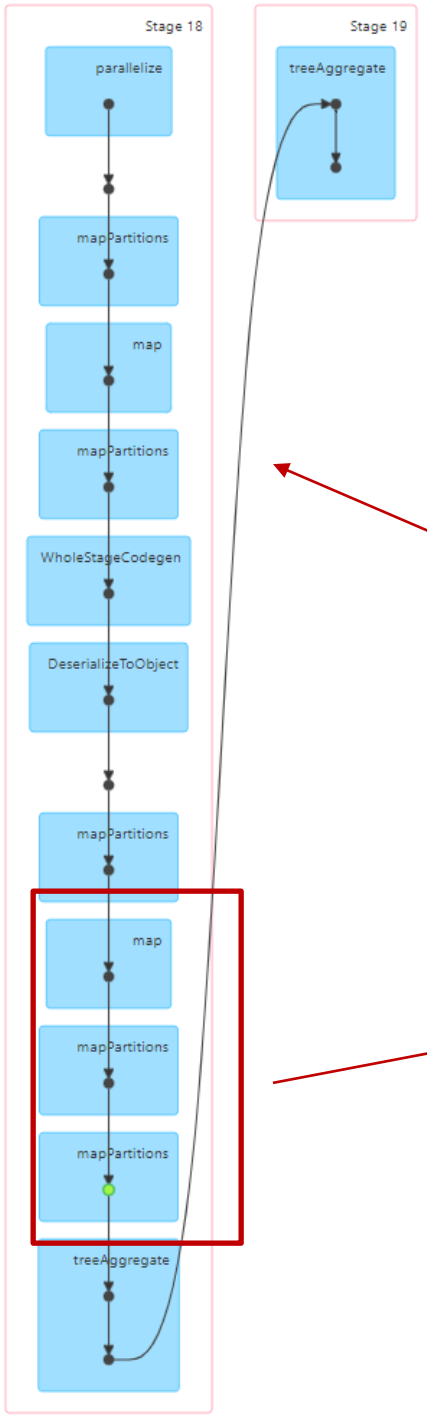
```
+---+------------------+-----+---------------------+-------------------------------------------------------+----------------------------------------+----------+
|id |text              |label|words                |features                                               |probability                             |prediction|
+---+------------------+-----+---------------------+-------------------------------------------------------+----------------------------------------+----------+
|4  |spark i j k       |1.0  |[spark, i, j, k]     |(262144,[19036,68693,173558,213660],[1.0,1.0,1.0,1.0])|[0.6292098489668488,0.37079015103315116]|0.0       |
|5  |l m n             |0.0  |[l, m, n]            |(262144,[1303,52644,248090],[1.0,1.0,1.0])             |[0.984770006762304,0.015229993237696027]|0.0       |
|6  |spark hadoop spark|1.0  |[spark, hadoop, spark]|(262144,[173558,198017],[2.0,1.0])                     |[0.13412348342566147,0.8658765165743385]|1.0       |
|7  |apache hadoop     |0.0  |[apache, hadoop]     |(262144,[68303,198017],[1.0,1.0])                      |[0.9955732114398529,0.00442678856014711]|0.0       |
+---+------------------+-----+---------------------+-------------------------------------------------------+----------------------------------------+----------+
```

```
# compute accuracy on the test set
predictionAndLabels = pred_test.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

▸ (1) Spark Jobs

▸ predictionAndLabels: pyspark.sql.dataframe.DataFrame = [prediction: double, label: double]

```
Test set accuracy = 0.75
```

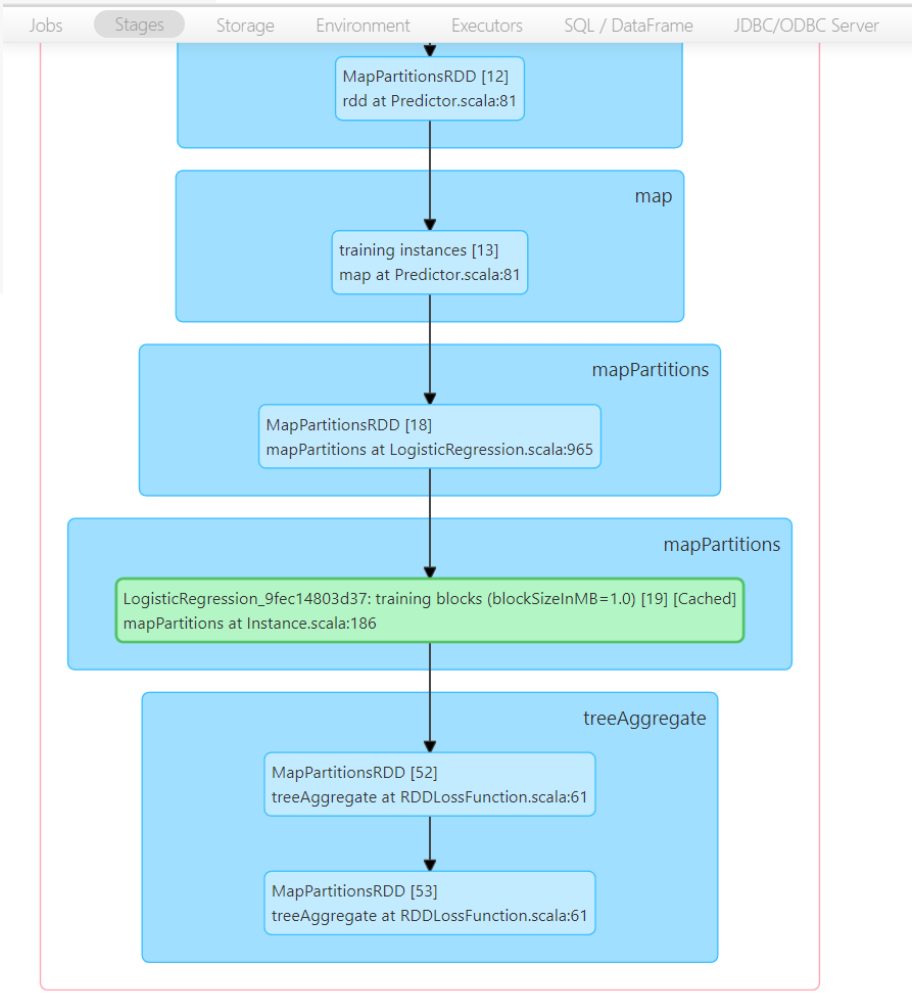



```
# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

(12) Spark Jobs

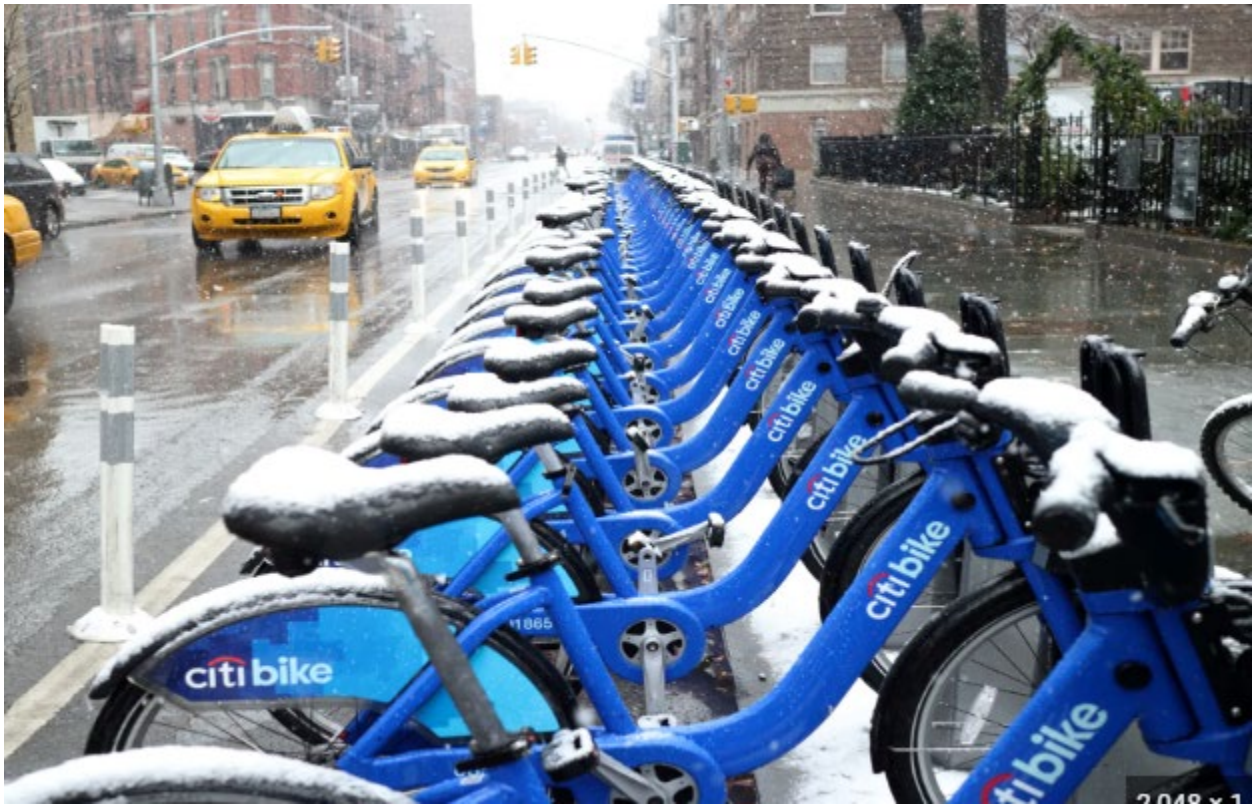
- Job 0 View (Stages: 2/2)
- Job 1 View (Stages: 2/2)
- Job 2 View (Stages: 2/2)
- Job 3 View (Stages: 2/2)
- Job 4 View (Stages: 2/2)
- Job 5 View (Stages: 2/2)
- Job 6 View (Stages: 2/2)
- Job 7 View (Stages: 2/2)
- Job 8 View (Stages: 2/2)
- Job 9 View (Stages: 2/2)
  - Stage 18: 8/8
  - Stage 19: 2/2
- Job 10 View (Stages: 2/2)
- Job 11 View (Stages: 2/2)

```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```



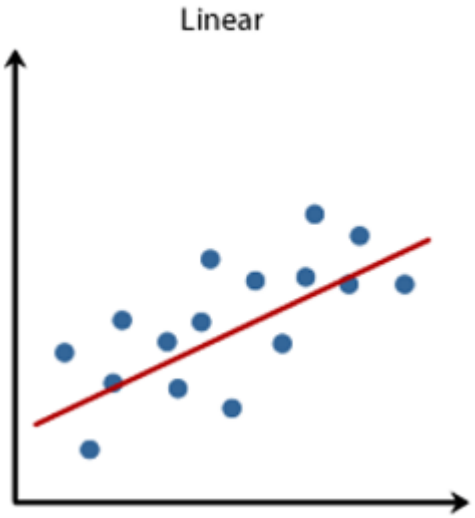
Assignment 2

A Regression Problem: predict the number of daily trips using citi bike data and other supporting data (e.g. weather data and etc.)



Regression Models

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- ……..



$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

trip_count (target / label)	feature_1	feature_2	feature_3	……..	feature_n

Predict a Numeric Label

Evaluate Regression Model

- Mean Absolute Error (MAE)

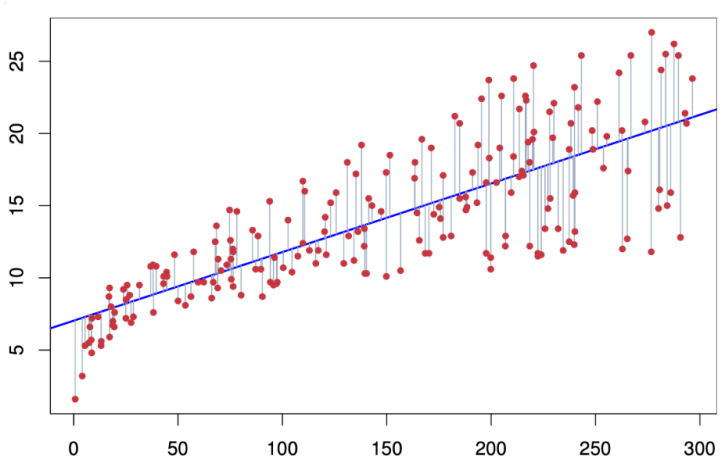
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Mean Squared Error (MSE)

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

- Root Mean Squared Error (RMSE)

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



Evaluate Regression Model

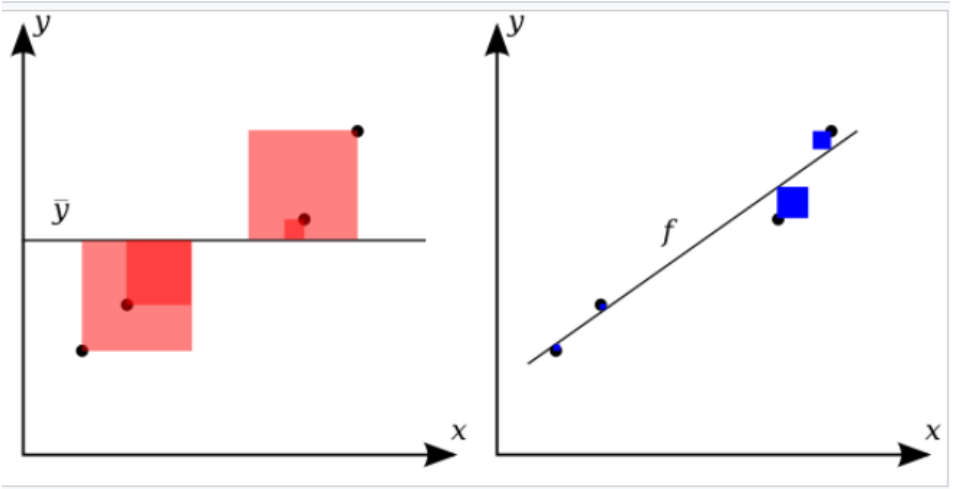
- R Squared Value
  - The closer the value to 1, the better the model fits the data.

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$



$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$

The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of R^2 is to 1. The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value.

https://en.wikipedia.org/wiki/Coefficient_of_determination

Practical 2: a regression example using Airbnb data

Practical 2a: Data Cleansing with Airbnb

Source: https://github.com/databricks/LearningSparkV2

We're going to start by doing some exploratory data analysis & cleansing

Cmd 2

Let's load the SF Airbnb dataset (comment out each of the options if you

Cmd 3

```
1  filePath = "/databricks-datasets/learning-spark-v2/sf-airbr
2
3  rawDF = spark.read.csv(filePath, header="true", inferSchema
4
5  display(rawDF)
```

Practical 2b: Regression - Predicting Rental Price

In this notebook, we will use the dataset we cleansed in the previous lab to predict Airbnb rental prices in San Francisco

Cmd 2

```
1  filePath = "/databricks-datasets/learning-spark-v2/sf-airbnb/sf-airbnb-clean.parquet"
2  airbnbDF = spark.read.parquet(filePath)
3  display(airbnbDF)
```

▸ (2) Spark Jobs

Practical 2c: One-Hot Encoding

In this notebook we will be adding additional features to our model, as well as discus

Cmd 2

```
1  filePath = "/databricks-datasets/learning-spark-v2/sf-airbnb/sf-airbn
2  airbnbDF = spark.read.parquet(filePath)
```

▸ (1) Spark Jobs

▸ airbnbDF: pyspark.sql.dataframe.DataFrame = [host_is_superhost: string, cancellation_policy: str

Command took 1.60 seconds -- by aixin@comp.nus.edu.sg at 9/29/2023, 9:32:59 AM on MyCluste

Cmd 3

Practical 2d: Hyperparameter Tuning

Let's perform hyperparameter tuning on a random forest to find the best hyperparameters!

Cmd 2

```
1  from pyspark.ml.feature import StringIndexer, VectorAssembler
2
3  filePath = "/databricks-datasets/learning-spark-v2/sf-airbnb/sf-airbnb-clean.parquet"
4  airbnbDF = spark.read.parquet(filePath)
5  (trainDF, testDF) = airbnbDF.randomSplit([.8, .2], seed=42)
6
7  categoricalCols = [field for (field, dataType) in trainDF.dtypes if dataType == "string"]
8  indexOutputCols = [x + "Index" for x in categoricalCols]
9
10 stringIndexer = StringIndexer(inputCols=categoricalCols, outputCols=indexOutputCols, handle
11
12 numericCols = [field for (field, dataType) in trainDF.dtypes if ((dataType == "double") & (
13 assemblerInputs = indexOutputCols + numericCols
14 vecAssembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
```

Source: https://github.com/databricks/LearningSparkV2

Acknowledgements

- CS4225 slides by He Bingsheng and Bryan Hooi
- Jules S. Damji, Brooke Wenig, Tathagata Das & Denny Lee, “Learning Spark: Lightning-Fast Data Analytics”
- Bill Chambers, Matei Zaharia, “Spark: The Definitive Guide”
- Spark SQL: Relational Data Processing in Spark, SIGMOD’15
- https://spark.apache.org/docs/latest/ml-pipeline.html