



Scalable Machine Learning with Apache Spark™



Introductions

- Introductions
 - Name
 - Spark/ML/Databricks Experience
 - Professional Responsibilities
 - Fun Personal Interest/Fact
 - Expectations for the Course



Course Objectives

- 1 Create data processing pipelines with Spark
- 2 Build and tune machine learning models with Spark ML
- 3 Track, version, and deploy machine learning models with MLflow
- 4 Perform distributed hyperparameter tuning with Hyperopt
- 5 Scale the inference of single-node models with Spark



Agenda (half-days)

Day 1

1. Spark Review*
2. Delta Lake Review*
3. ML Overview*
4. Break
5. Data Cleansing
6. Data Exploration Lab
7. Break
8. Linear Regression, pt. 1

Day 2

1. Linear Regression, pt. 1 Lab
2. Linear Regression, pt. 2
3. Break
4. Linear Regression, pt. 2 Lab
5. MLflow Tracking
6. Break
7. MLflow Model Registry
8. MLflow Lab

Day 3

1. Decision Trees
2. Break
3. Random Forest and Hyperparameter Tuning
4. Hyperparameter Tuning Lab
5. Break
6. Hyperopt
7. Hyperopt Lab

Day 4

1. AutoML
2. AutoML Lab
3. Feature Store
4. Break
5. XGBoost
6. Inference with Pandas UDFs
7. Pandas UDFs Lab
8. Break
9. Training with Pandas Function API
10. Pandas API on Spark

Agenda (full days)

Day 1

1. **Spark Review***
2. **Delta Lake Review***
3. **ML Overview***
4. Break
5. **Data Cleansing**
6. **Data Exploration Lab**
7. Break
8. **Linear Regression, pt. 1**
9. **Linear Regression, pt. 1 Lab**
10. **Linear Regression, pt. 2**
11. Break
12. **Linear Regression, pt. 2 Lab**
13. **MLflow Tracking**
14. Break
15. **MLflow Model Registry**
16. **MLflow Lab**

Day 2

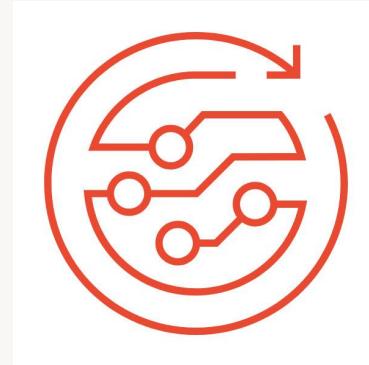
1. **Decision Trees**
2. Break
3. **Random Forest and Hyperparameter Tuning**
4. **Hyperparameter Tuning Lab**
5. Break
6. **Hyperopt**
7. **Hyperopt Lab**
8. **AutoML**
9. **AutoML Lab**
10. **Feature Store**
11. Break
12. **XGBoost**
13. **Inference with Pandas UDFs**
14. **Pandas UDFs Lab**
15. Break
16. **Training with Pandas Function API**
17. **Koalas**

Survey

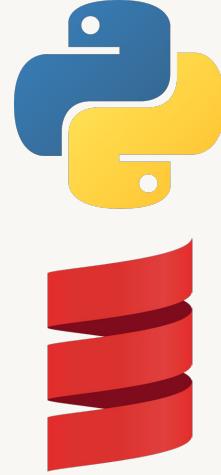
Apache Spark



Machine Learning



Programming
Language



LET'S GET STARTED

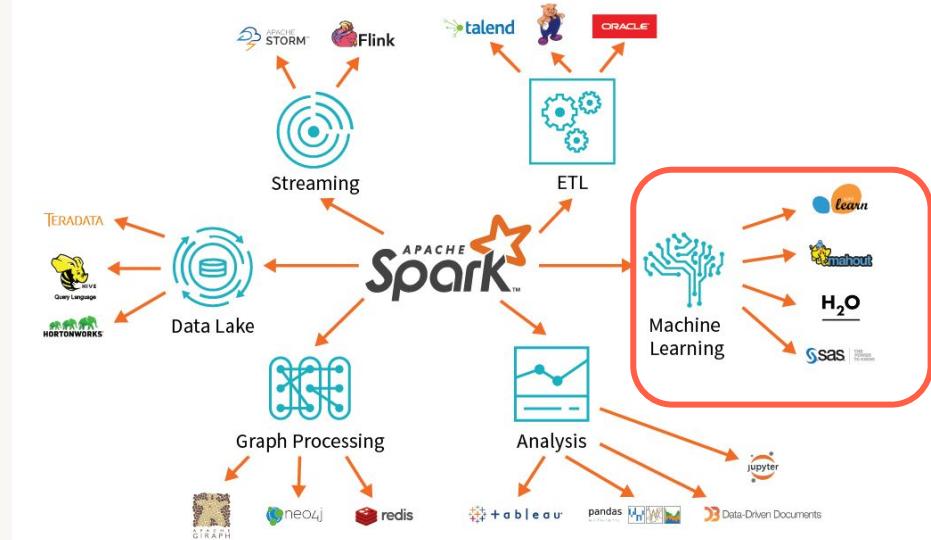


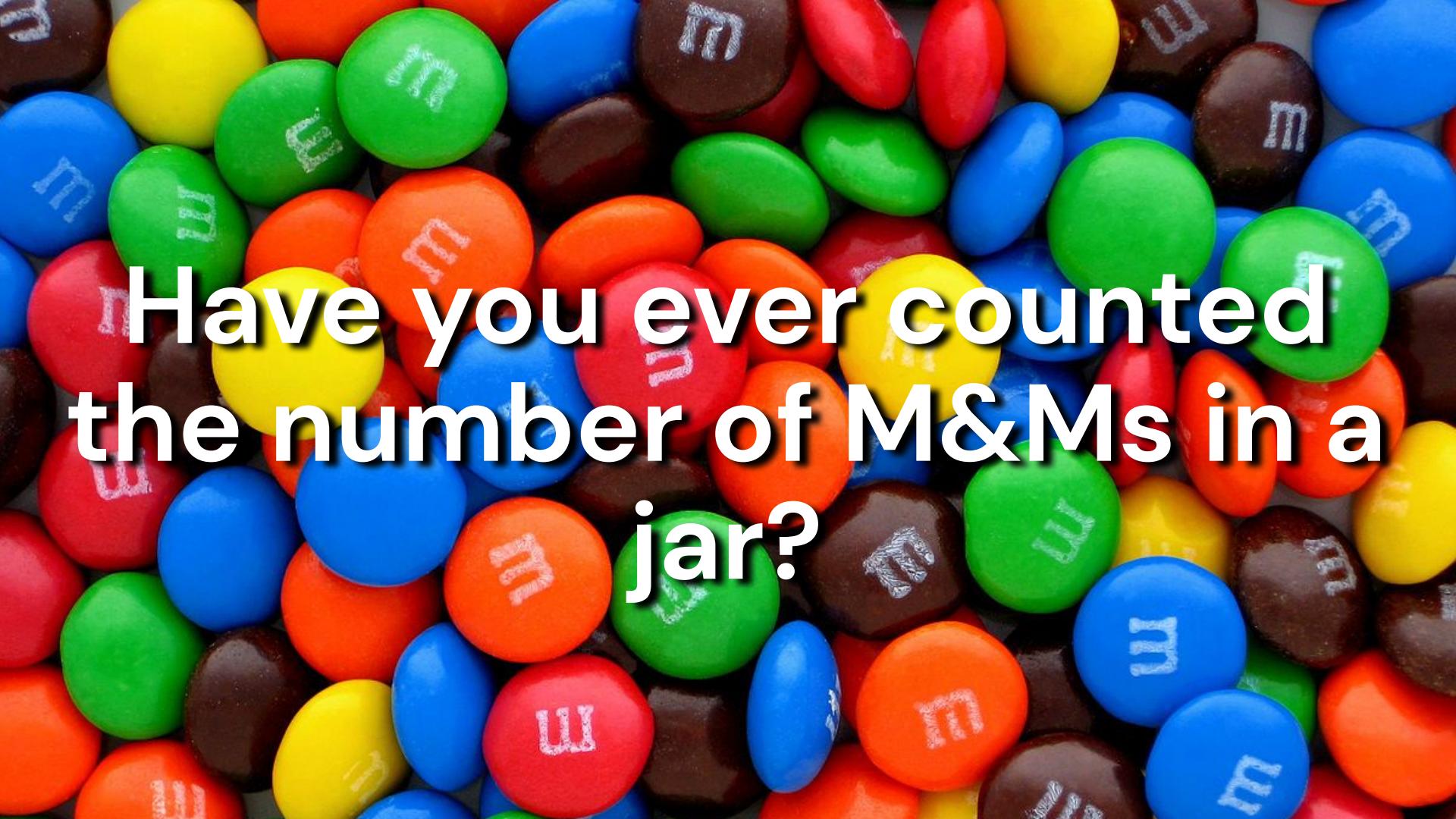
Apache Spark™ Overview



Apache Spark Background

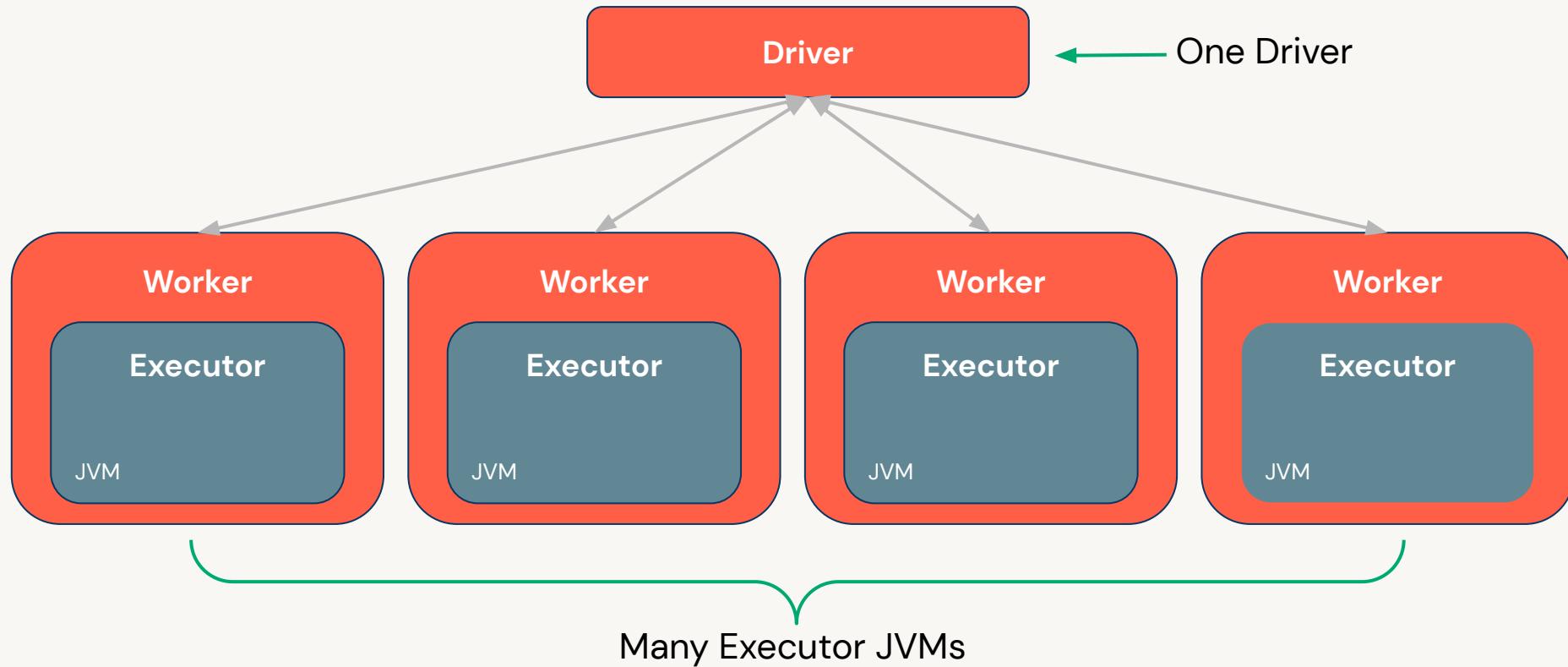
- Founded as a research project at UC Berkeley in 2009
- Open-source unified data analytics engine for big data
- Built-in APIs in SQL, Python, Scala, R, and Java



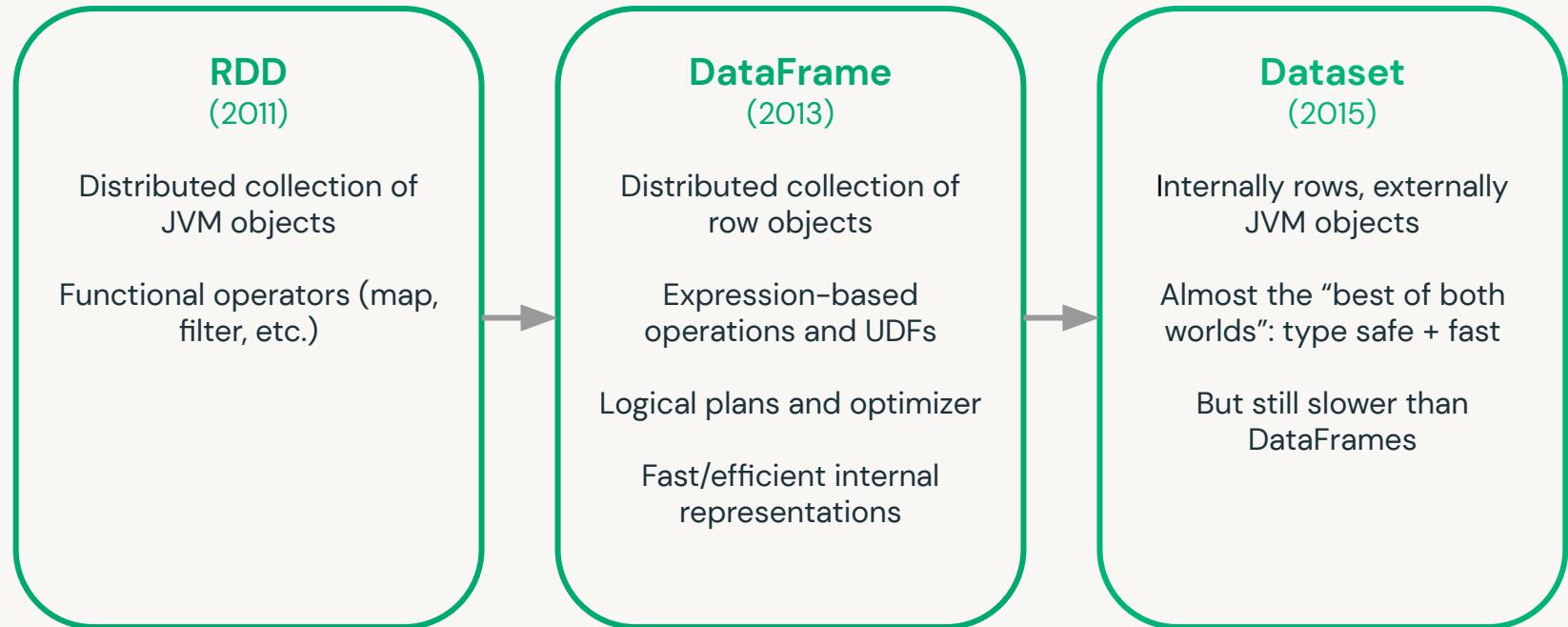
A close-up photograph of a large pile of M&Ms candies. The candies are of various colors including red, blue, green, yellow, orange, and brown. Each candy has a white 'm' printed on it. The candies are scattered across the frame, filling the background.

Have you ever counted
the number of M&Ms in a
jar?

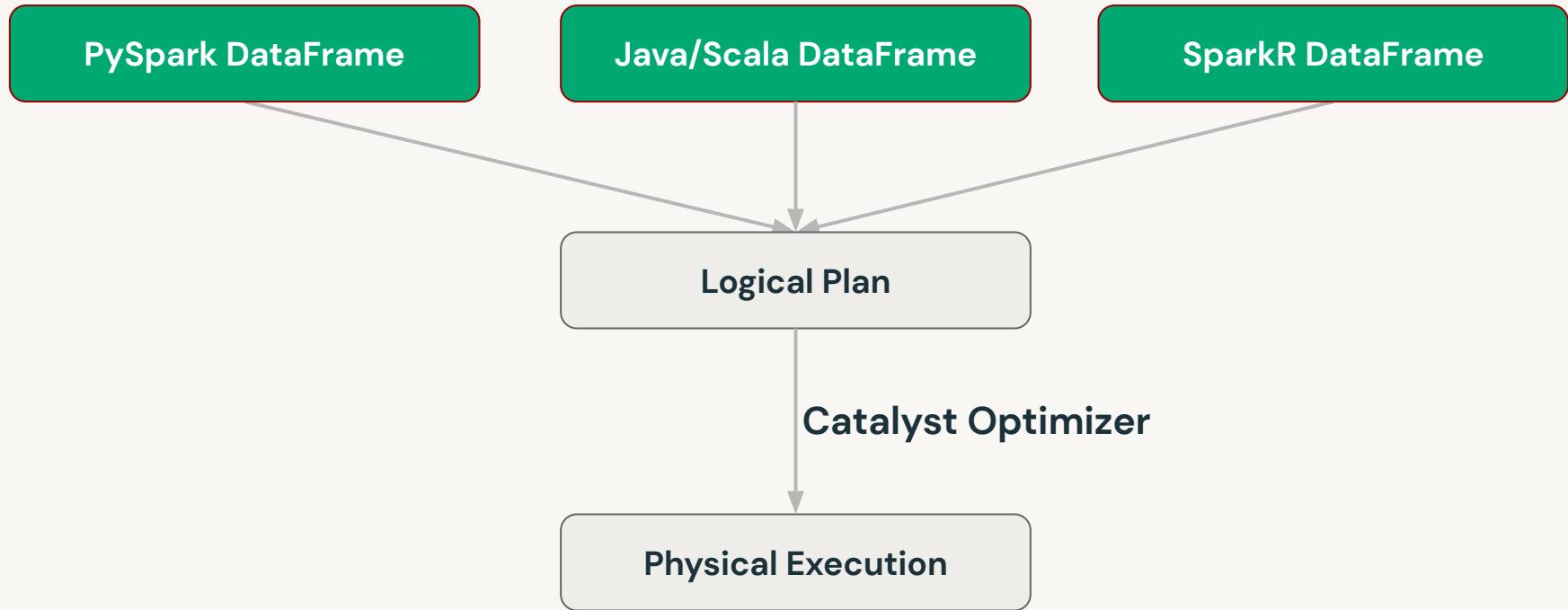
Spark Cluster



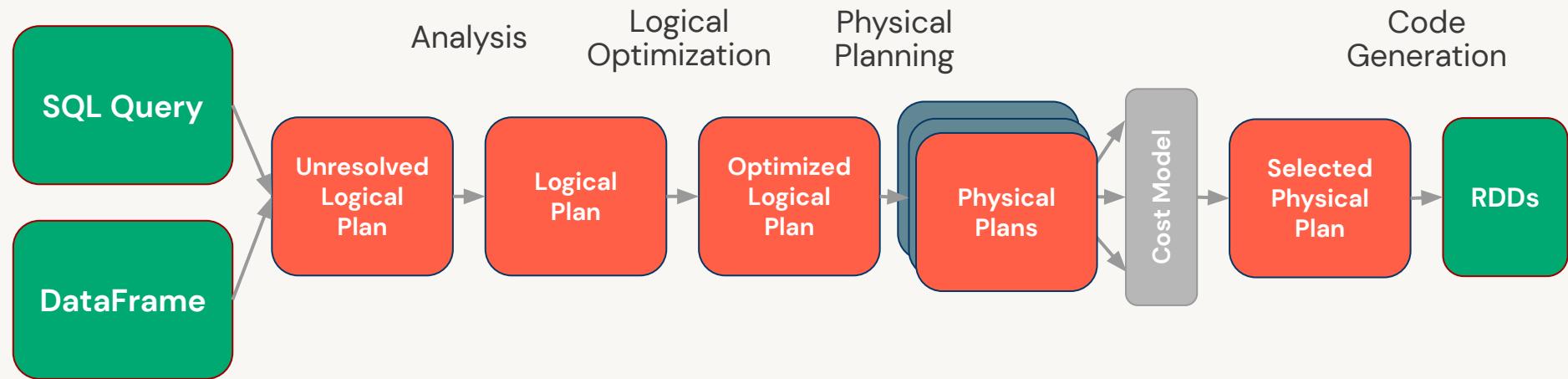
Spark's Structured Data APIs



Spark DataFrame Execution



Under the Catalyst Optimizer's Hood



When to Use Spark

Scaling Out

Data or model is too large to process on a single machine, commonly resulting in out-of-memory errors

Speeding Up

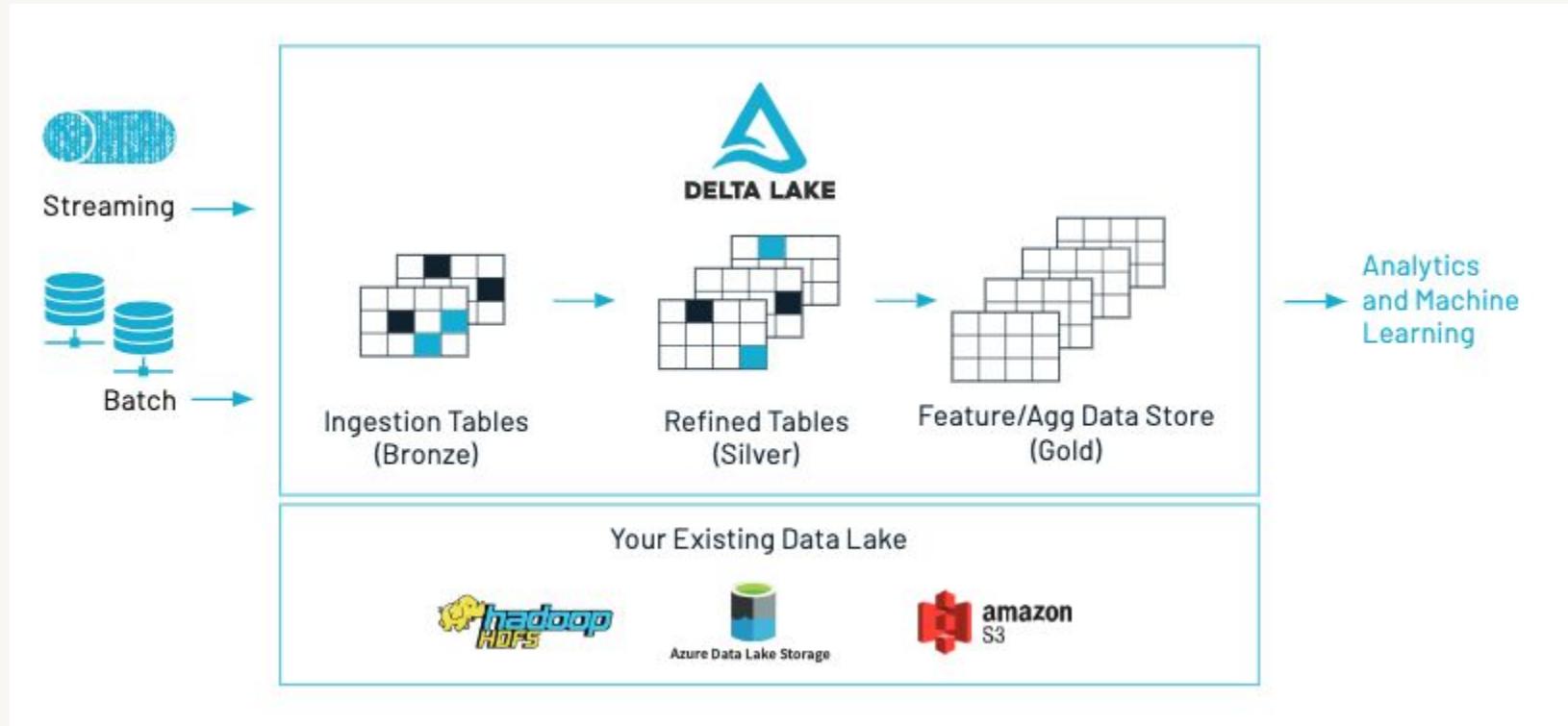
Data or model is processing slowly and could benefit from shorter processing times and faster results



Delta Lake Overview



Open-source Storage Layer



Delta Lake's Key Features

- ACID transactions
- Time travel (data versioning)
- Schema enforcement and evolution
- Audit history
- Parquet format
- Compatible with Apache Spark API



Machine Learning Overview (Optional)



What is Machine Learning

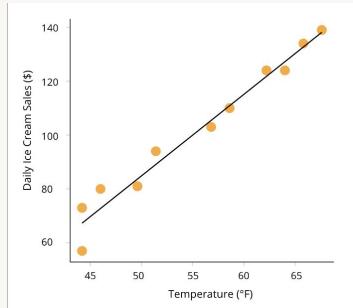
- Learn **patterns** and **relationships** in your data without explicitly programming them
- Derive an approximation function to map **features** to an **output** or relate them to each other



Types of Machine Learning

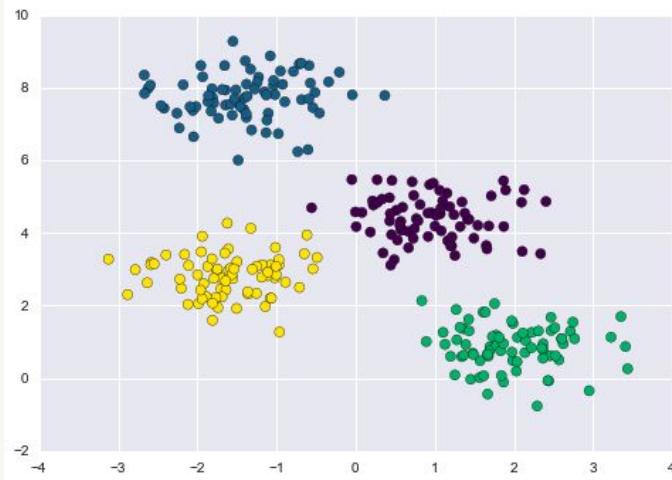
Supervised Learning

- Labeled data (known function output)
- Regression (a continuous/ordinal-discrete output)
- Classification (a categorical output)



Unsupervised Learning

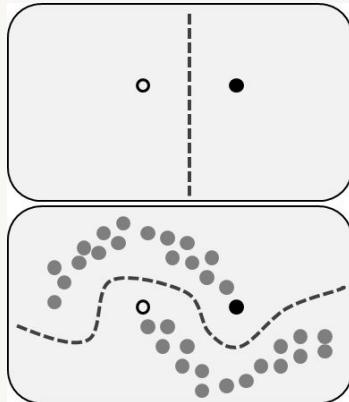
- Unlabeled data (no known function output)
- Clustering (categorize records based on features)
- Dimensionality reduction (reduce feature space)



Types of Machine Learning

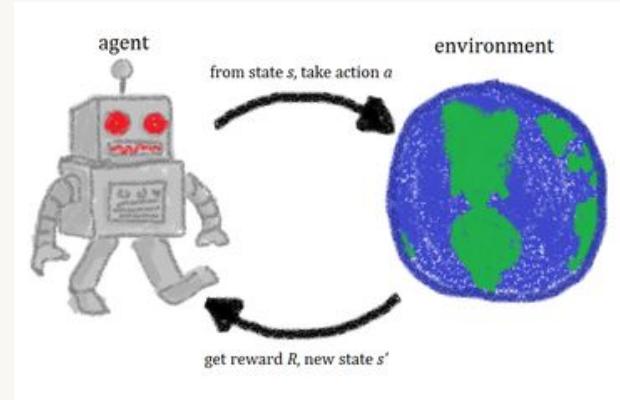
Semi-supervised Learning

- Labeled and unlabeled data, mostly unlabeled
- Combines supervised learning and unsupervised learning
- Commonly trying to label the unlabeled data to be used in another round of training

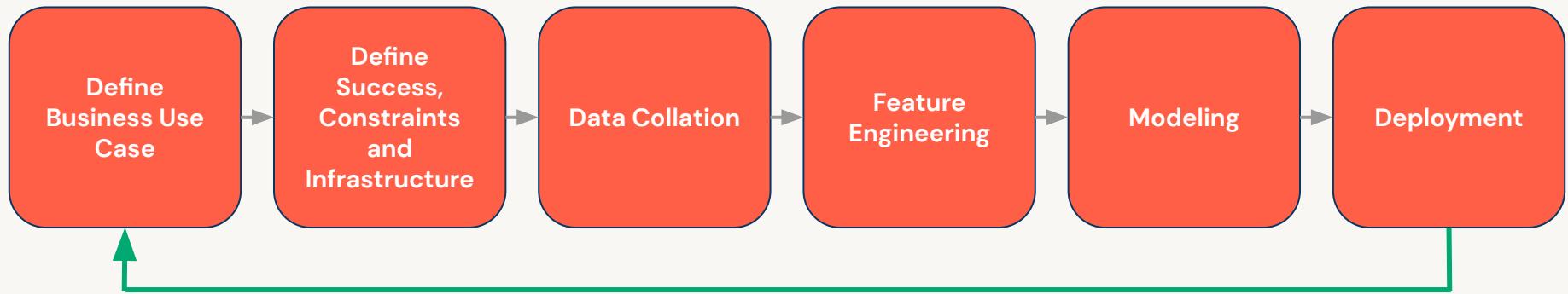


Reinforcement Learning

- States, actions, and rewards
- Useful for exploring spaces and exploiting information to maximize expected cumulative rewards
- Frequently utilizes neural networks and deep learning



Machine Learning Workflow



Defining and Measuring Success: Establish baseline!

		Prediction	
		Positive	Negative
Actual	Positive	True Positive	False Negative
	Negative	False Positive	True Negative



DATA CLEANSING DEMO



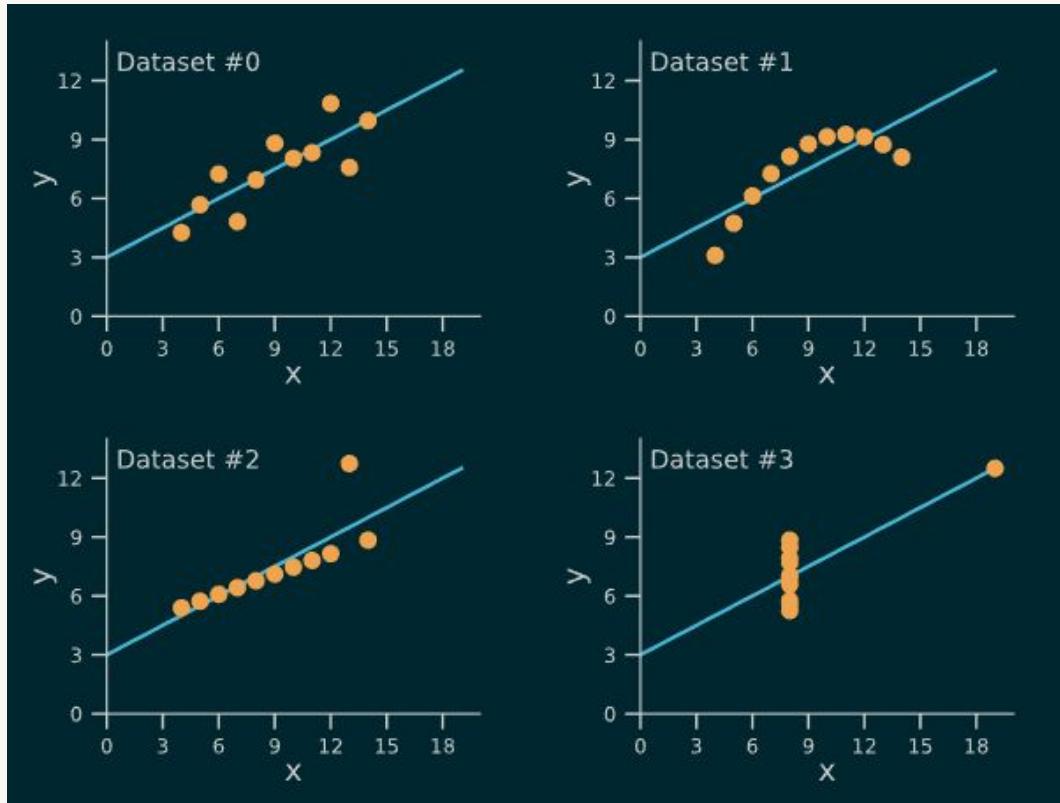
Importance of Data Visualization

Dataset #0		Dataset #1		Dataset #2		Dataset #3	
x	y	x	y	x	y	x	y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

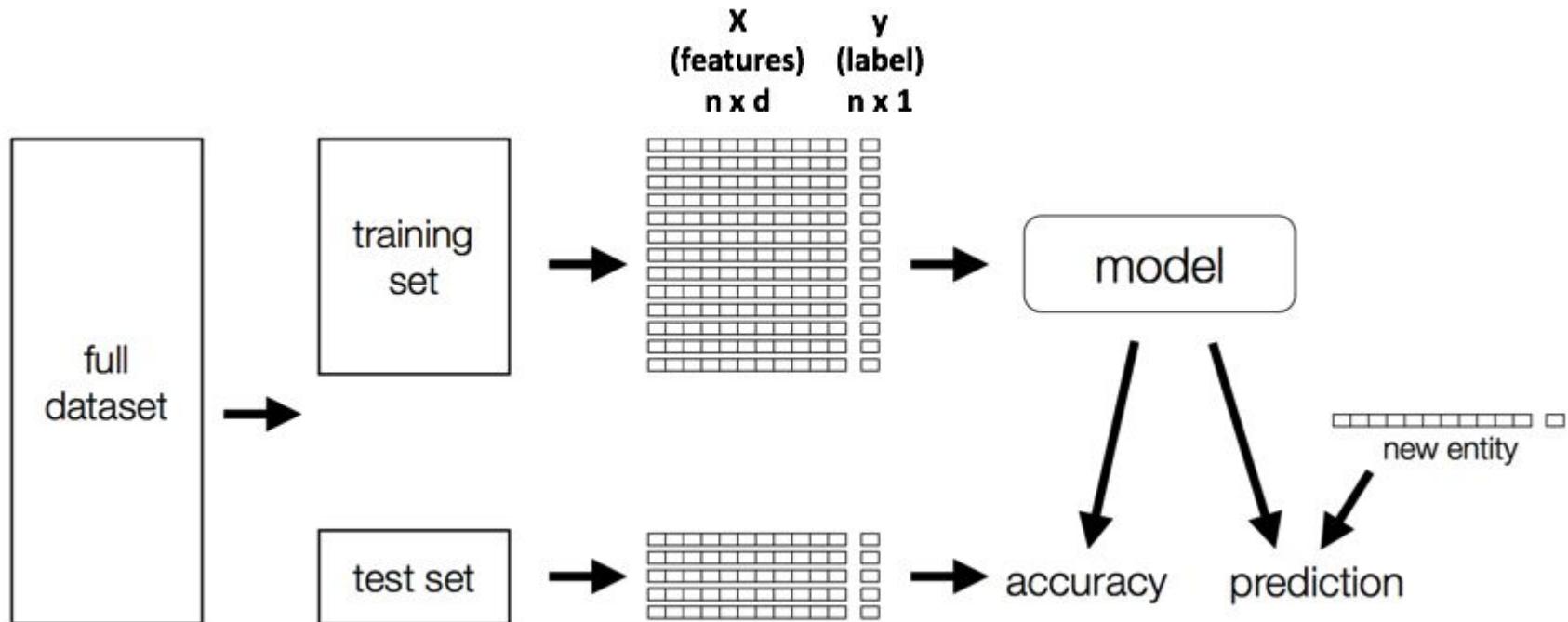
Mean	9	7.5	9	7.5	9	7.5	9	7.5
Variance	11	4.1	11	4.1	11	4.1	11	4.1
Correlation	0.86		0.86		0.86		0.86	
Regression line	$y = 3 + 0.5x$							



Importance of Data Visualization



How do we build and evaluate models?



DATA EXPLORATION LAB



Linear Regression



Linear Regression

Goal: Find the *line of best fit*.

$$\hat{y} = w_0 + w_1 x$$

$$y \approx \hat{y} + \epsilon$$

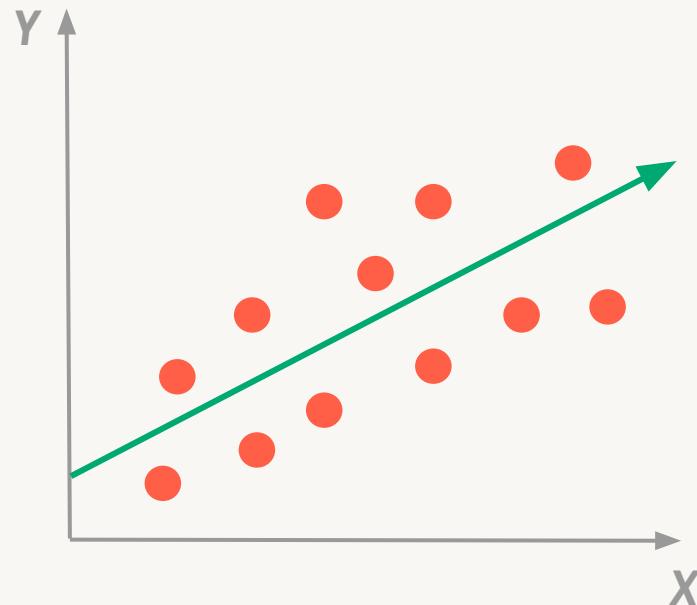
where...

x : feature

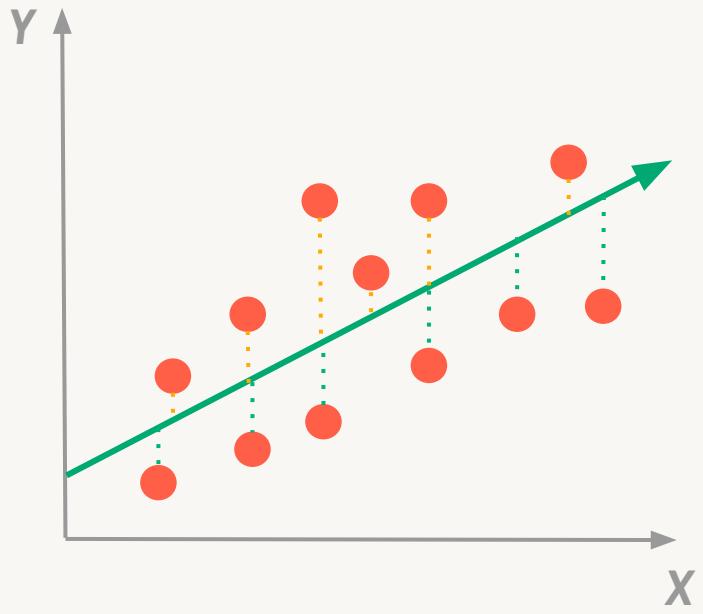
y : label

w_0 : y -intercept

w_1 : slope of the line of best fit



Minimizing the Residuals



- **Red point:** True value
- **Green-dotted line:** Positive residual
- **Orange-dotted line:** Negative residual
- **Green line:** Line of best fit

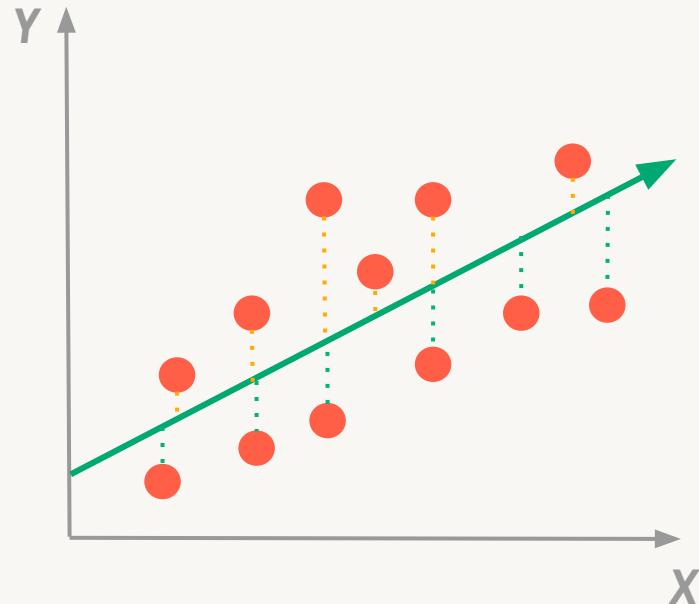
The goal is to draw a line that minimizes the sum of the squared residuals.

Regression Evaluators

Measure the “closeness” between the **actual value** and the **predicted value**.

Evaluation Metrics

- **Loss:** $(y - \hat{y})$
- **Absolute loss:** $|y - \hat{y}|$
- **Squared loss:** $(y - \hat{y})^2$



Evaluation Metric: Root mean-squared-error (RMSE)

$$Error = (y_i - \hat{y}_i)$$

$$SE = (y_i - \hat{y}_i)^2$$

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

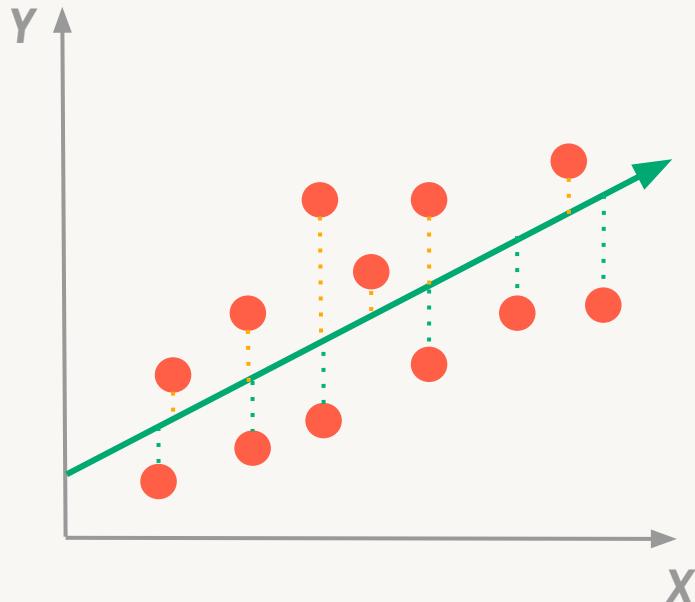
$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

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



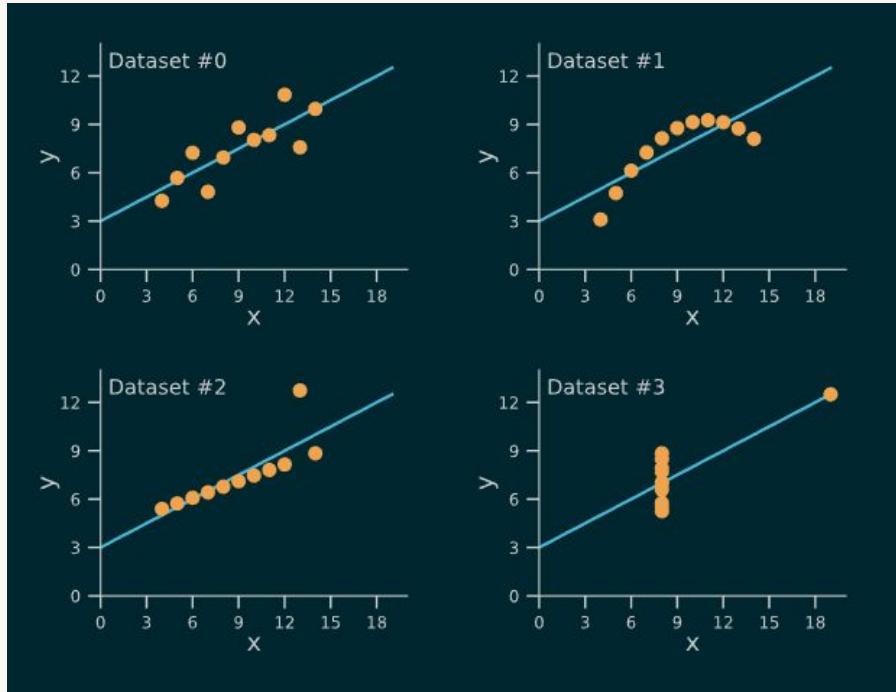
Linear Regression Assumptions

- Linear relationship between X and the mean of Y (*linearity*)
- Observations are independent from one another (*independence*)
- Y is normally distributed for any fixed observation (*normality*)
- The variance of residual is the same for any feature (*homoscedasticity*)

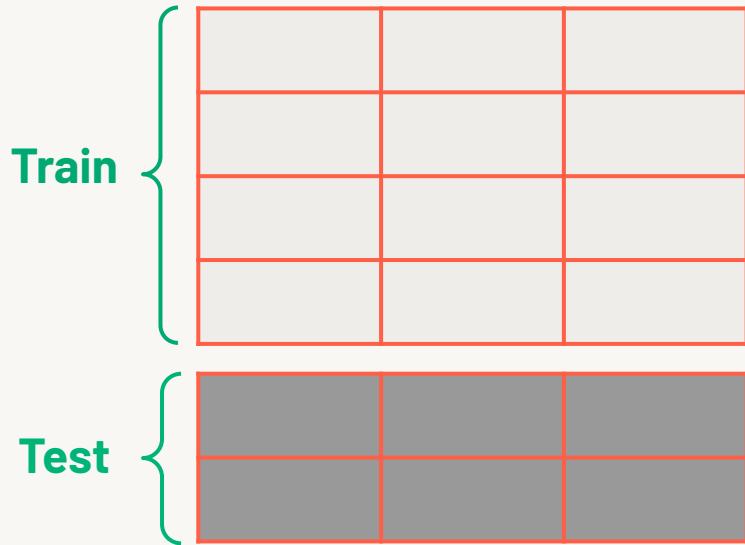


Linear Regression Assumptions

So, which datasets are suited for linear regression?



Train vs. Test RMSE



Which is more important? Why?

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

Evaluation Metric: R²

$$SS_{tot} = \sum_{i=1}^n (y_i - \bar{y})^2$$

$$SS_{res} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

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

What is the range of R²?

Do we want it to be higher or lower?



Machine Learning Libraries



Scikit-learn is a popular single-node machine learning library.

But what if our data or model get too big?

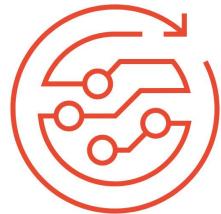


Machine Learning in Spark

Scale Out and **Speed Up**

Machine learning in Spark allows us to work with **bigger data** and train models **faster** by **distributing the data and computations** across **multiple workers**.

Spark Machine Learning Libraries



MLlib

Original ML API
for Spark

Based on RDDs

Maintenance
Mode

Spark ML

Newer ML API for
Spark

Based on
DataFrames



LINEAR REGRESSION DEMO I



LINEAR REGRESSION LAB I



Non-numeric Features

Two primary types of non-numeric features

Categorical Features

A series of categories of a single feature

No intrinsic ordering

e.g. Dog, Cat, Fish

Ordinal Features

A series of categories of a single feature

Relative ordering, but not necessarily consistent spacing

e.g. Infant, Toddler, Adolescent, Teen, Young Adult, etc.



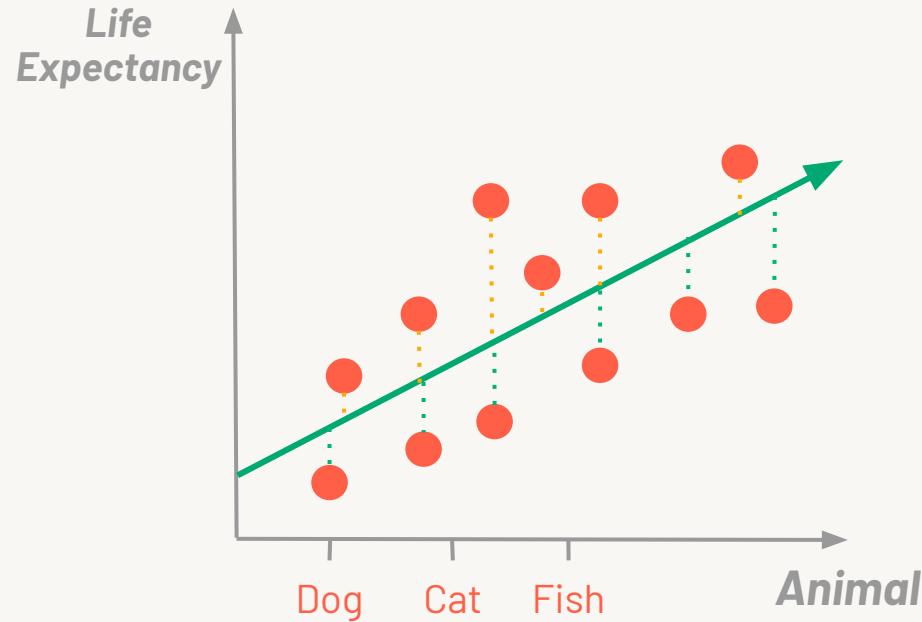
Non-numeric Features in Linear Regression

**How do we handle
non-numeric features for
linear regression?**

- X-axis is numeric, so features need to be numeric
- Convert our non-numeric features to numeric features?

**Could we assign numeric
values to each of the
categories**

- “Dog” = 1, “Cat” = 2, “Fish” = 3, etc.
- Does this make sense?



This implies 1 Cat is equal to 2 Dogs!



Non-numeric Features in Linear Regression

Instead, we commonly use a practice known as **one-hot encoding (OHE)**.

- Creates a binary “dummy” feature for each category

The diagram illustrates the process of one-hot encoding (OHE). On the left, there is a table with a single column labeled "Animal" containing three categories: "Dog", "Cat", and "Fish". An arrow labeled "OHE" points from this table to a larger matrix on the right. The matrix has three columns labeled "Dog", "Cat", and "Fish". The rows correspond to the categories in the original table. The values in the matrix are binary (0 or 1), indicating the presence or absence of each category. For "Dog", the values are 1, 0, 0 respectively. For "Cat", the values are 0, 1, 0 respectively. For "Fish", the values are 0, 0, 1 respectively.

Animal	Dog	Cat	Fish
Dog	1	0	0
Cat	0	1	0
Fish	0	0	1

- Doesn't force a uniformly-spaced, ordered numeric representation



One-hot Encoding at Scale

You might be thinking...

- Okay, I see what's happening here ... this works for a handful of animals.
- But what if we have an entire zoo of animals? That would result in really wide data!

Spark uses sparse vectors for this...

```
DenseVector(0, 0, 0, 7, 0, 2, 0, 0, 0, 0)  
SparseVector(10, [3, 5], [7, 2])
```

- Sparse vectors take the form:

(Number of elements, [indices of non-zero elements], [values of non-zero elements])



LINEAR REGRESSION DEMO II



LINEAR REGRESSION LAB II



MLflow Tracking



MLflow



- Open-source platform for machine learning lifecycle
- Operationalizing machine learning
- Developed by Databricks
- Pre-installed on the Databricks Runtime for ML

Core Machine Learning Issues

- Keeping track of experiments or model development
- Reproducing code
- Comparing models
- Standardization of packaging and deploying models

MLflow addresses these issues.



MLflow Components



Record and query experiments: code, data, config, results



Packaging format for reproducible runs on any platform



General model format that supports diverse deployment tools



Centralized and collaborative model lifecycle management

- APIs: CLI, Python, R, Java, REST

MLflow tracking and autologging

Track ML development with one line of code: parameters, metrics, data lineage, model, and environment.

`mlflow.autolog()`

Ensure reproducibility



mlflow Model Deployment Options



In-Line Code



Containers



Batch & Stream
Scoring



OSS Inference
Solutions



Cloud Inference
Services

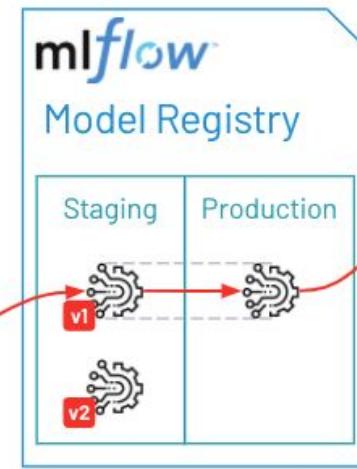
The Full ML Lifecycle

Data Scientists build features.
Data Engineers provide infra for automating featurization.

Data Scientists build models and log them to MLflow, which records environment info.

Data Scientists move models to Staging.

Deployment Engineers manage CI/CD tools which promote models to Production.



MLFLOW TRACKING DEMO



MLflow Model Registry



MLflow Model Registry

- Collaborative, centralized model hub
- Facilitate experimentation, testing, and production
- Integrate with approval and governance workflows
- Monitor ML deployments and their performance

Databricks MLflow Blog Post



One Collaborative Hub for Model Management

Centralized Model Management and Discovery

Registered Models

Create Model

clemens

Name	Latest Version	Staging	Production	Last Modified	Serving
clemens-airlinedelay-latest	Version 1	—	Version 1	2020-10-26 09:12:56	—
clemens-delay-test	Version 3	Version 2	Version 1	2020-10-26 20:52:26	Ready
clemens-model2	Version 1	—	—	2020-02-25 14:58:33	Ready
clemens-power-forecasting-model	Version 11	Version 5	Version 2	2020-11-04 14:25:10	Failed
clemens-windfarm-signature	Version 12	Version 3	Version 2	2020-10-21 11:28:03	Ready

< Page 1 > 10 / page ▾

- Overview of all registered models, their versions at Staging and Production
- Search by name, tags, etc.
- Model-based ACLs

Full lineage from deployed models to training code / data

Registered Models > clemens-power-forecasting-model > Version 11 ▾

Registered At: 2020-11-04 14:25:10 Creator: clemens.mewald@databricks.com Stage: Production

Last Modified: 2020-12-06 16:51:41 Source Run: keras

/Users/clemens.mewald@databricks.com/_MLflow Dem... > keras ▾ Reproduce Run

Date: 2020-09-03 20:54:54 Source: MLflow Model Registry example (multi framework) User: clemens.mewald@databricks.com

Duration: 16.3s Status: FINISHED

You are viewing a notebook revision from Sep 3 2020, 20:55 PM PDT that created the highlighted run. Exit

Notes : first trial Test for overfit : yes Test on golden dataset : no

Cmd 10

```
def train_scikit_model(X, y):
    from sklearn.ensemble import RandomForestRegressor
    ...
```

- Full lineage from Model Version to
 - Run that produced the model
 - Notebook that produced the run
 - Exact revision history of the notebook that produced the run



Version Control and Visibility into Deployment Process

Versioning of ML artifacts

Versions		All	Active(2)	Compare		
	Version	Registered at	Created by	Stage	Pending Requests	Description
<input type="checkbox"/>	Version 3	2020-08-25 12:24:11	clemens.mewald@databricks.com	Staging	1	
<input type="checkbox"/>	Version 2	2020-08-25 12:15:48	clemens.mewald@databricks.com	Production	-	

Registered Models > clemens-windfarm-signature > Comparing 2 Versions

Run ID:	2494e060ef8547a589db131815177cbf	812fab4cba18458ea3c9a67c5ad3aec6
Model Version:	2	3
Run Name:		
Start Time:	2020-08-25 12:15:20	2020-08-25 12:23:31

- Overview of active model versions and their deployment stage
- Comparison of versions and their logged metrics, parameters, etc.

Visibility and auditability of the deployment process

Activities

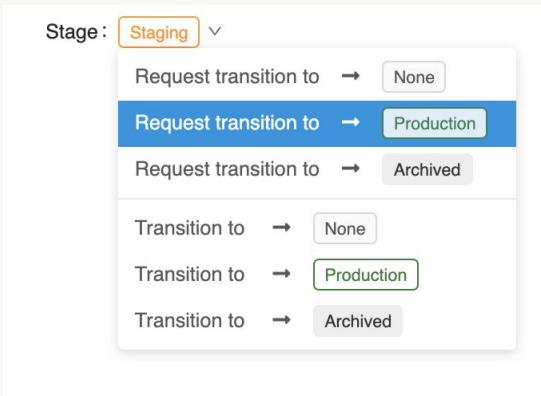
 clemens.mewald@databricks.com applied a stage transition	None	→ Staging	2 months ago
 clemens.mewald@databricks.com requested a stage transition	Staging	→ Production	2 months ago
asdf			
 clemens.mewald@databricks.com rejected a stage transition	Staging	→ Production	2 months ago
 clemens.mewald@databricks.com requested a stage transition	Staging	→ Production	2 months ago

- Audit log of stage transitions and requests per model



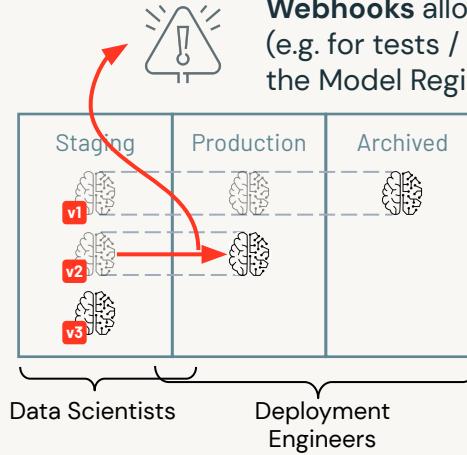
Review Processes and CI/CD Integration

Manual review process



- Stage-based Access Controls
- Request and approval workflow for stage transitions

Automation through CI/CD integration



Webhooks allow registering of callbacks (e.g. for tests / deployment) on events in the Model Registry

- Webhooks for events like model creation, version creation, transition request, etc.
- Mechanisms to store results / metadata through Tags and Comments

MLFLOW MODEL REGISTRY DEMO



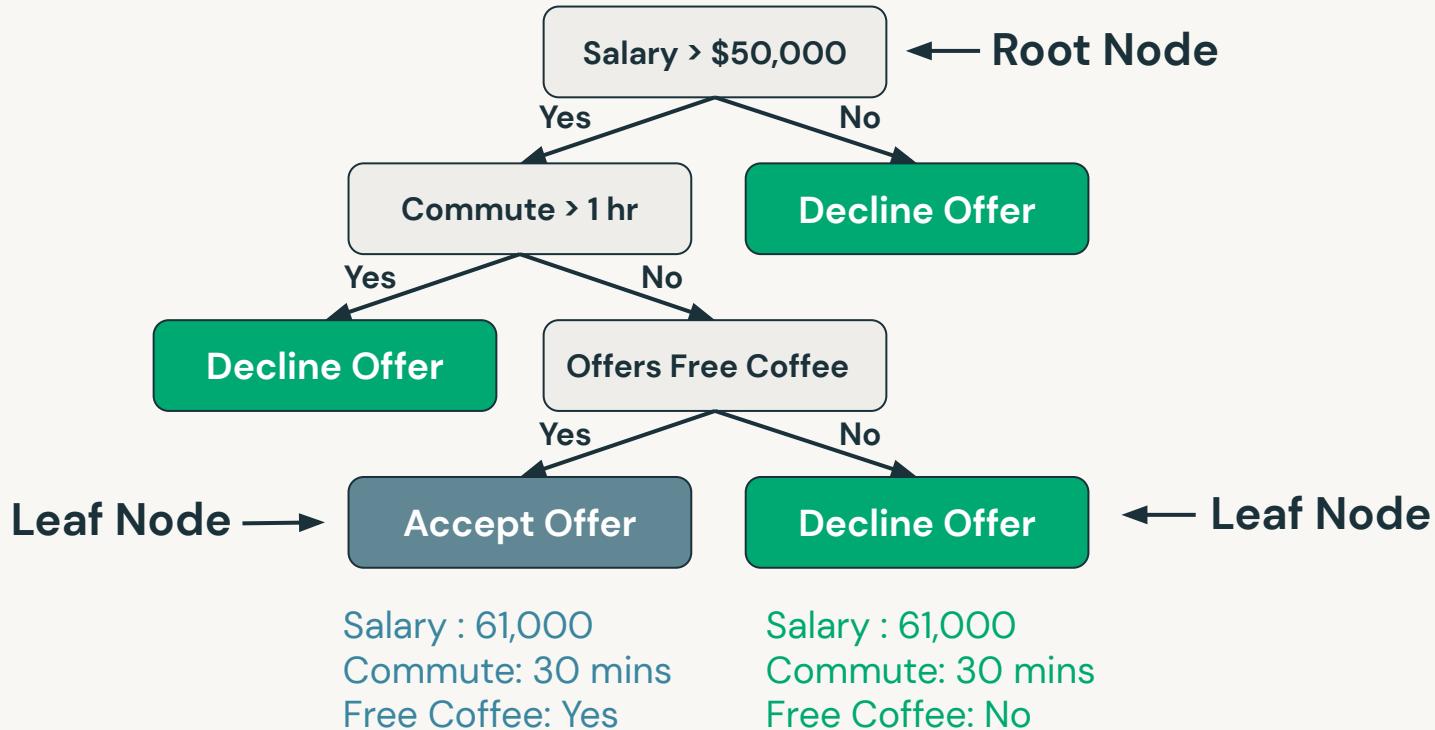
MLFLOW LAB



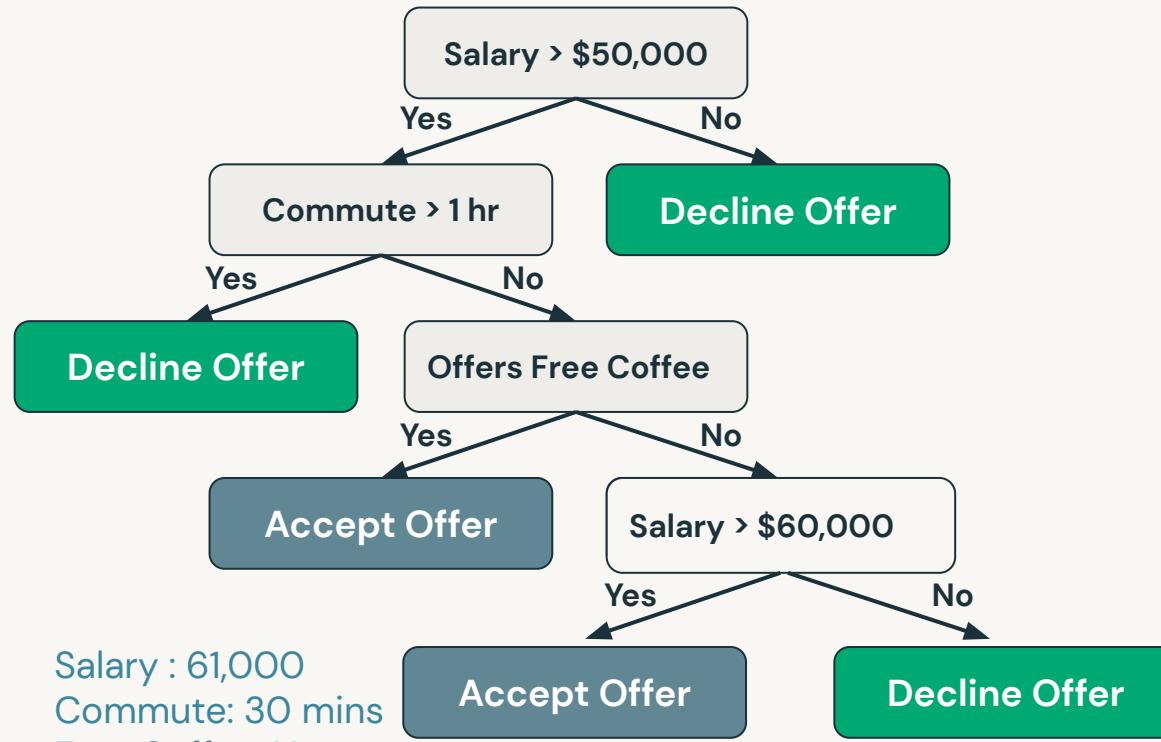
Decision Trees



Decision Making



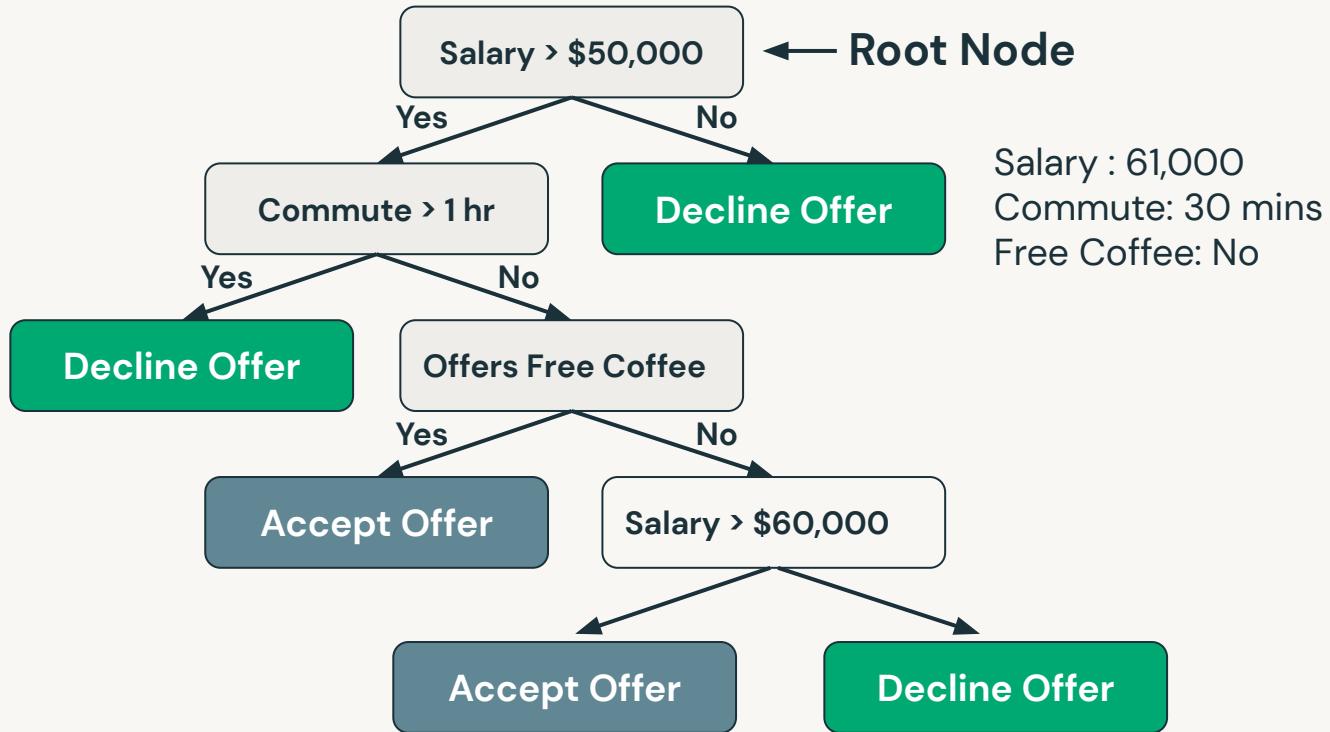
Decision Making



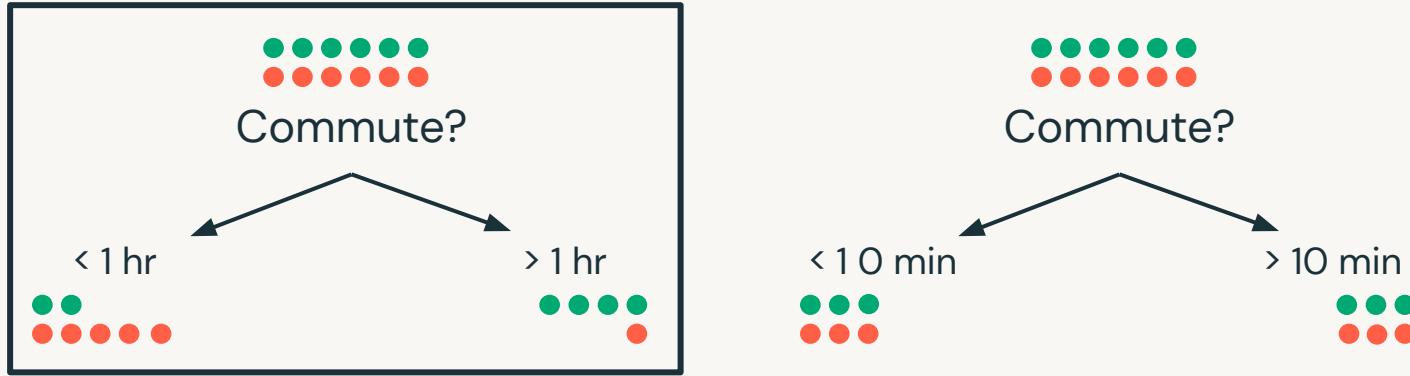
Salary : 61,000
Commute: 30 mins
Free Coffee: No



Decision Making

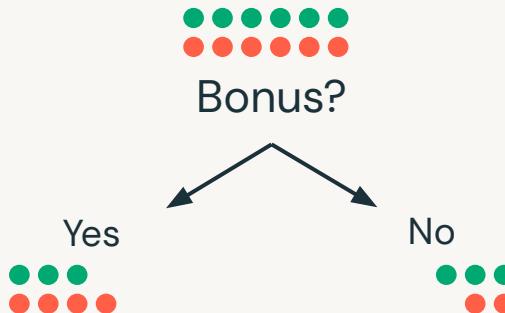
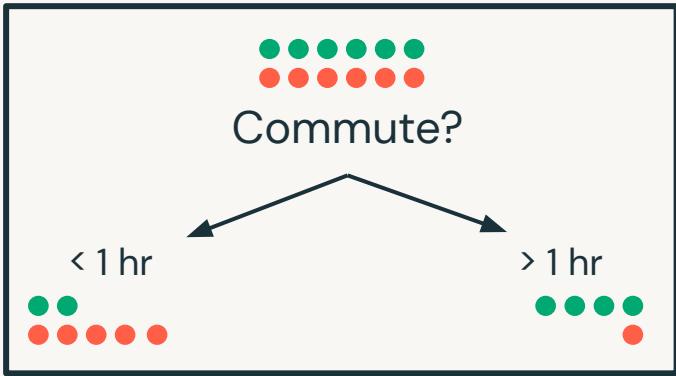


Determining Splits



1 hr is a better splitting point for **Commute** because it provides information about the classification.

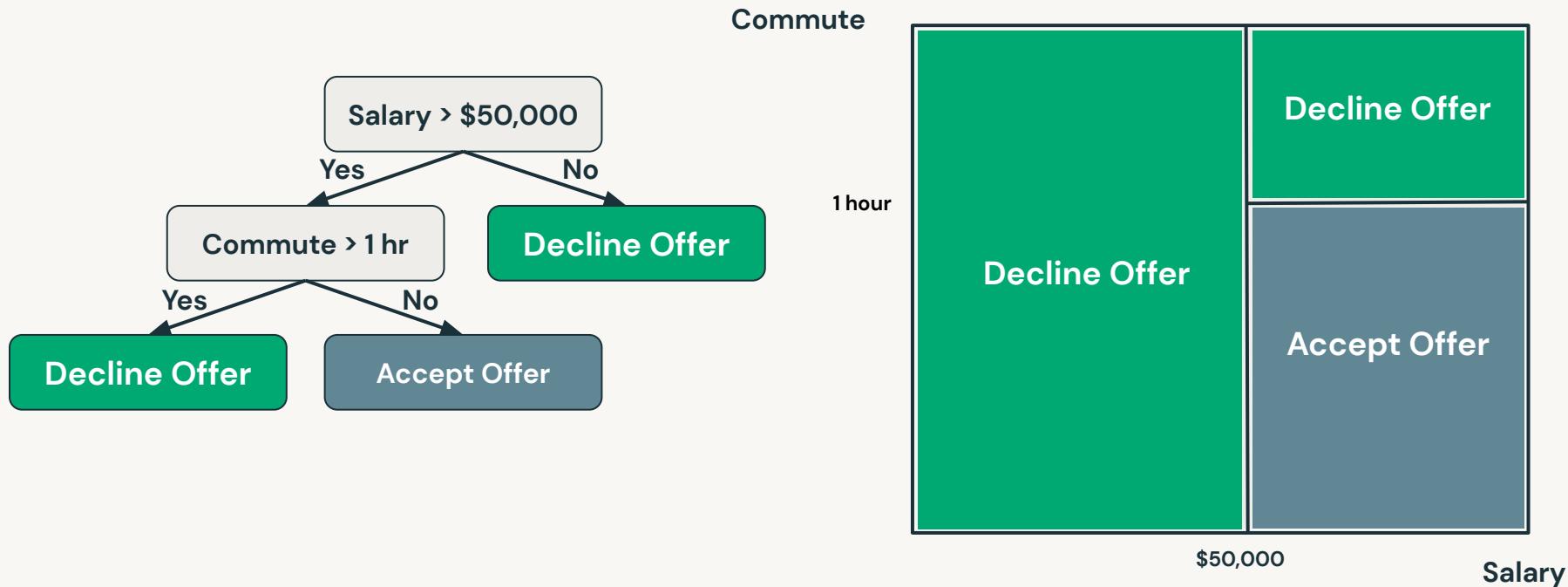
Determining Splits



Commute is a better choice because it provides information about the classification.

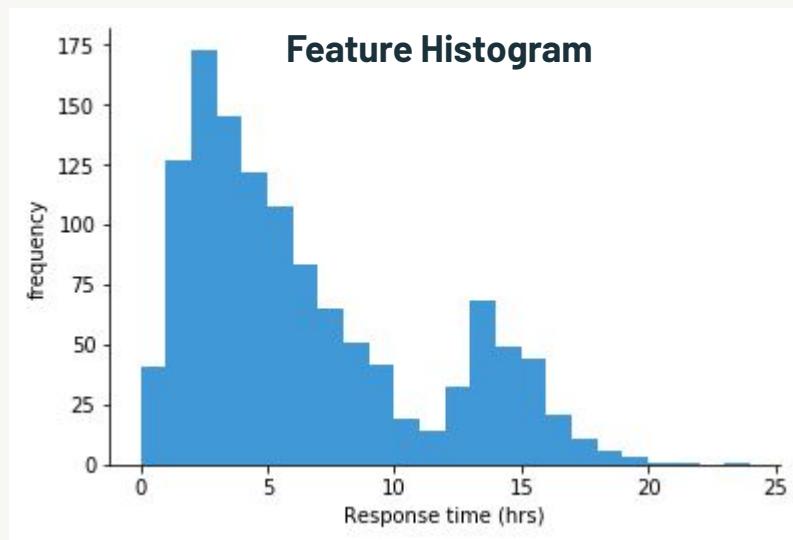


Creating Decision Boundaries



Create Split Candidates

Numerical Features



Feature values

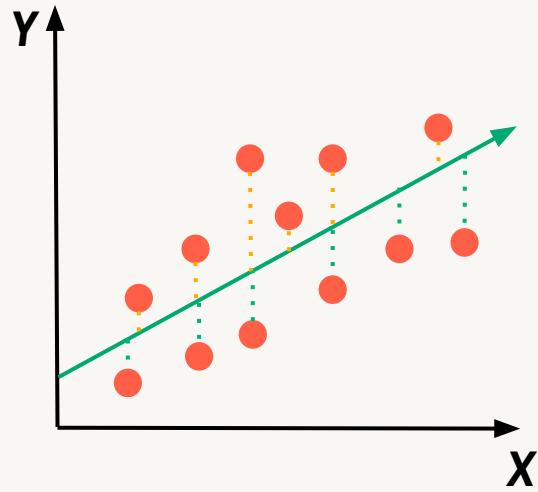
Categorical Features



Lines vs. Boundaries

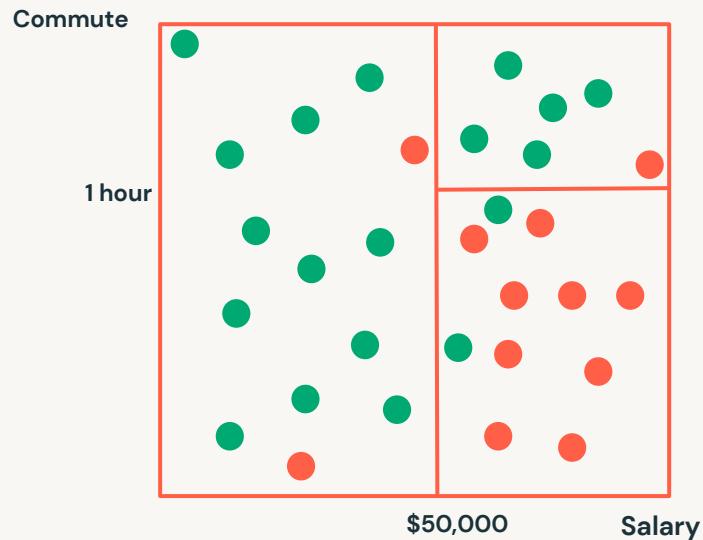
Linear Regression

- Lines through data
- Assumed linear relationship

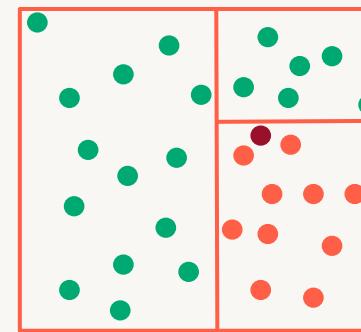
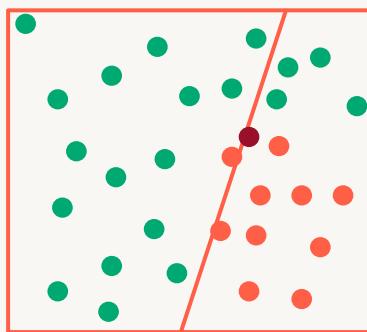
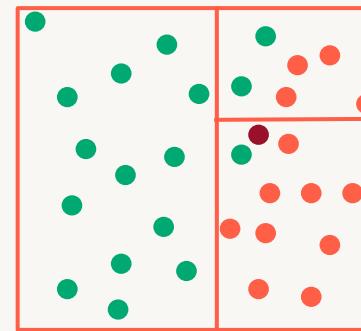
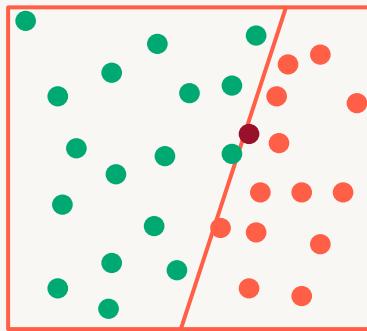


Decision Trees

- Boundaries instead of lines
- Learn complex relationships



Linear Regression or Decision Tree?

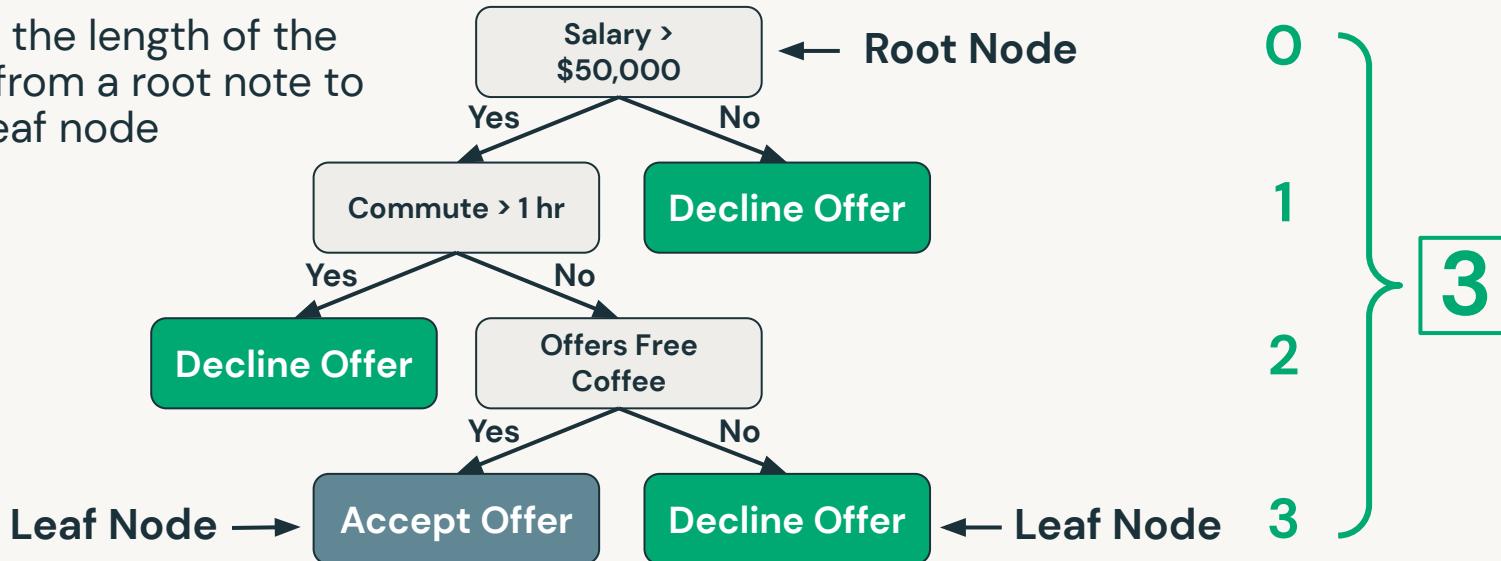


It depends on the data...



Tree Depth

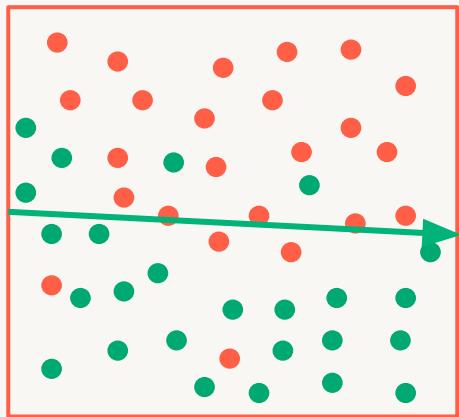
Tree Depth: the length of the longest path from a root note to a leaf node



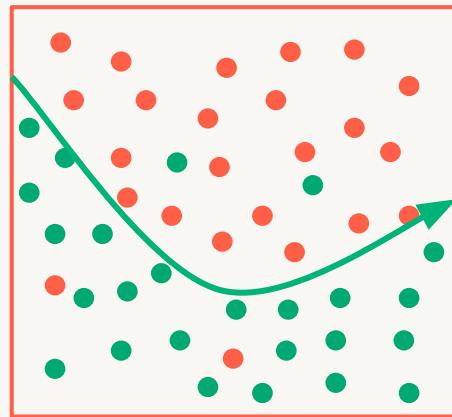
Note: shallow trees tend to **underfit**, and deep trees tend to **overfit**

Underfitting vs. Overfitting

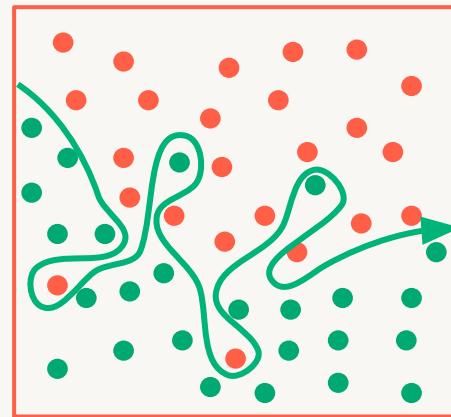
Underfitting



Just Right



Overfitting



Additional Resource

R2D3 has an excellent visualization of
how decision trees work.



DECISION TREE DEMO



Random Forests



Decision Trees

Pros

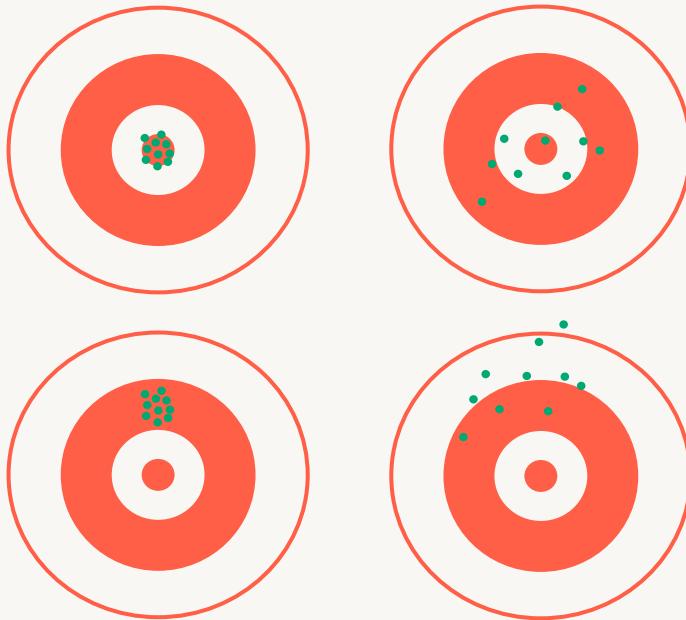
- Interpretable
- Simple
- Classification/Regression
- Nonlinear relationships

Cons

- Poor accuracy
- High variance

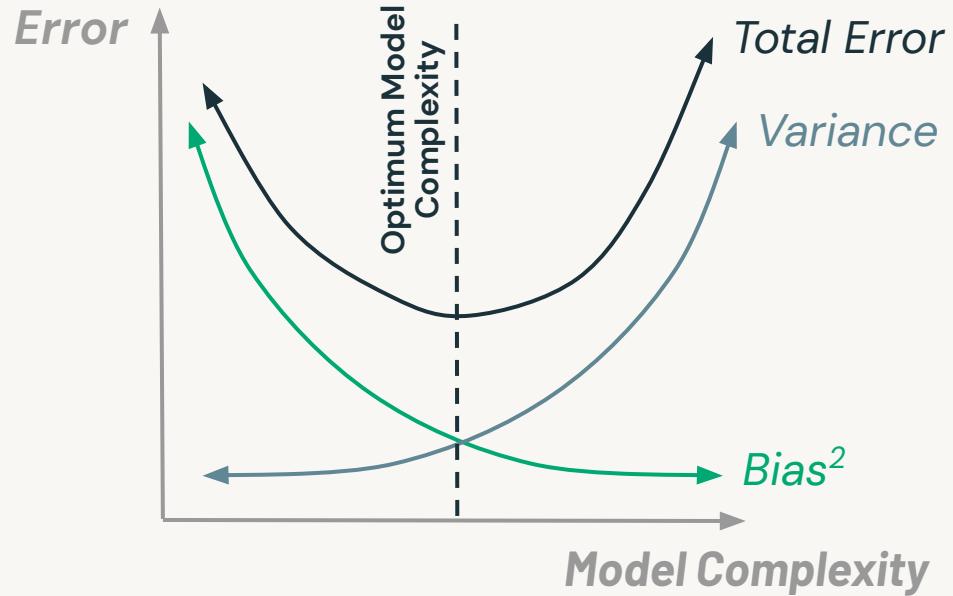


Bias vs. Variance



Bias-Variance Tradeoff

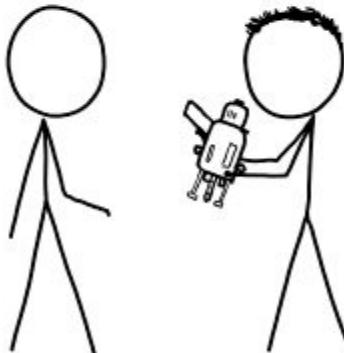
$$\text{Error} = \text{Variance} + \text{Bias}^2 + \text{noise}$$



- Reduce Bias
 - Build more complex models
- Reduce Variance
 - Use a lot of data
 - Build simple models
- What about the noise?

WE NEED TO MAKE 500 HOLES IN THAT WALL,
SO I'VE BUILT THIS AUTOMATIC DRILL. IT USES
ELEGANT PRECISION GEARS TO CONTINUALLY
ADJUST ITS TORQUE AND SPEED AS NEEDED.

GREAT, IT'S THE PERFECT WEIGHT!
WE'LL LOAD 500 OF THEM INTO
THE CANNON WE MADE AND
SHOOT THEM AT THE WALL.



HOW SOFTWARE DEVELOPMENT WORKS

Building Five Hundred Decision Trees

- Using more data reduces variance for one model
- Averaging more predictions reduces prediction variance
- But that would require more decision trees
- And we only have one training set ... or do we?



Bootstrap Sampling

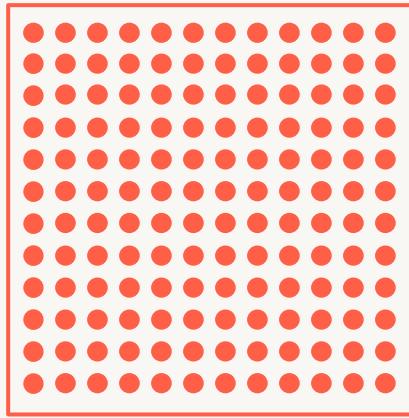
A method for simulating N new datasets:

1. Take sample *with replacement* from original training set
2. Repeat N times

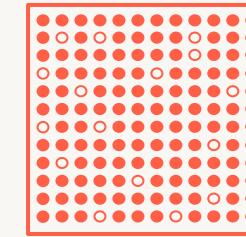
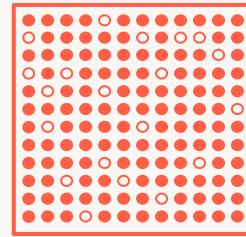


Bootstrap Visualization

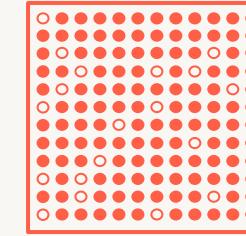
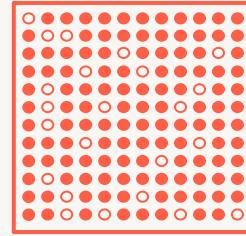
Training Set ($N = 100$)



Bootstrap 1 ($N = 100$) Bootstrap 2 ($N = 100$)



Bootstrap 3 ($N = 100$) Bootstrap 4 ($N = 100$)



Why are some points in the bootstrapped samples not selected?



Training Set Coverage

Assume we are bootstrapping N draws from a training set with N observations ...

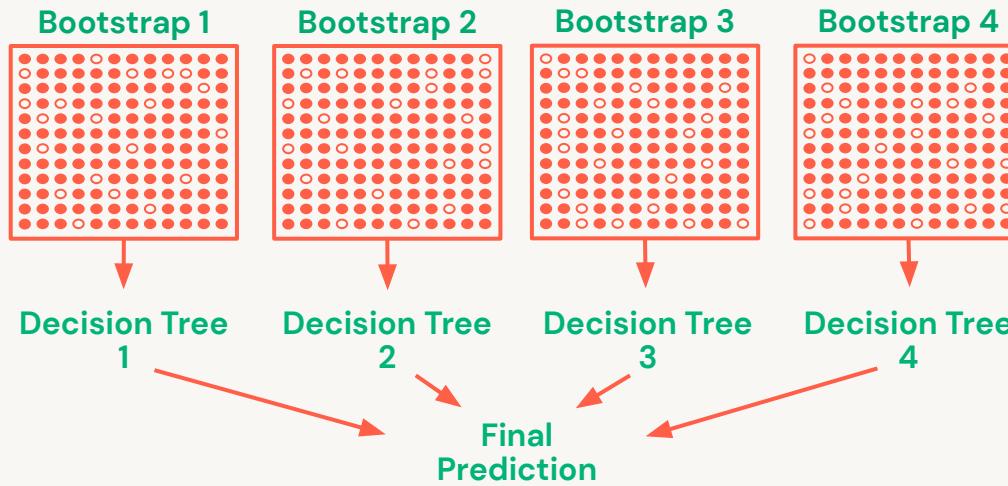
- Probability of an element getting picked in *each draw*: $\frac{1}{N}$
- Probability of an element *not* getting picked in *each draw*: $1 - \frac{1}{N}$
- Probability of an element *not* getting drawn in the entire *sample*: $(1 - \frac{1}{N})^N$

As $N \rightarrow \infty$, the probability for each element of not getting picked in a sample approaches **0.368**.

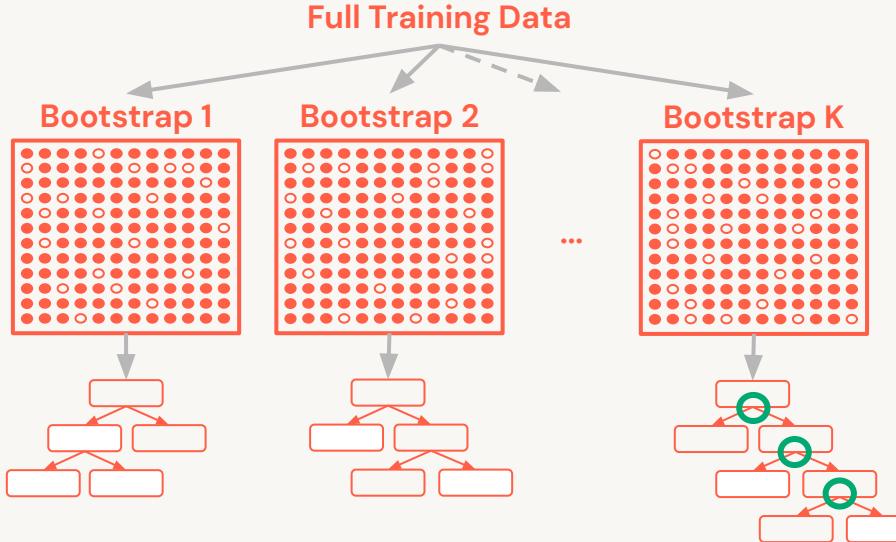


Bootstrap Aggregating

- Train a tree on each of sample, and average the predictions
- This is bootstrap aggregating, commonly referred to as bagging



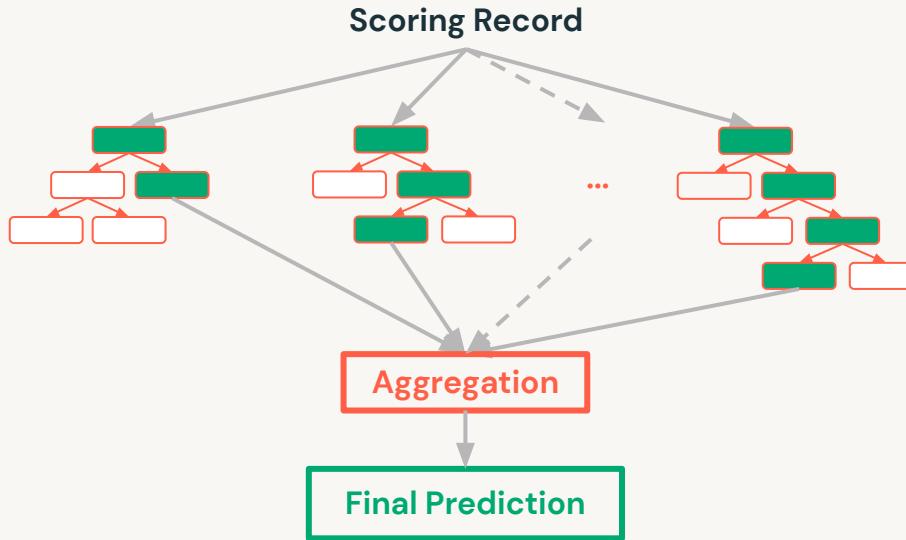
Random Forest Algorithm



At each split, a **subset of features** is considered to ensure each tree is different.



Random Forest Aggregation



- Majority-voting for classification
- Mean for regression



RANDOM FOREST DEMO



Hyperparameter Tuning



What is a Hyperparameter?

- Examples for Random Forest:
 - Tree depth
 - Number of trees
 - Number of features to consider

A parameter whose value is used to control the training process.



Selecting Hyperparameter Values

- Build a model for each hyperparameter value
- Evaluate each model to identify the optimal hyperparameter value
- What dataset should we use to train and evaluate?

Training

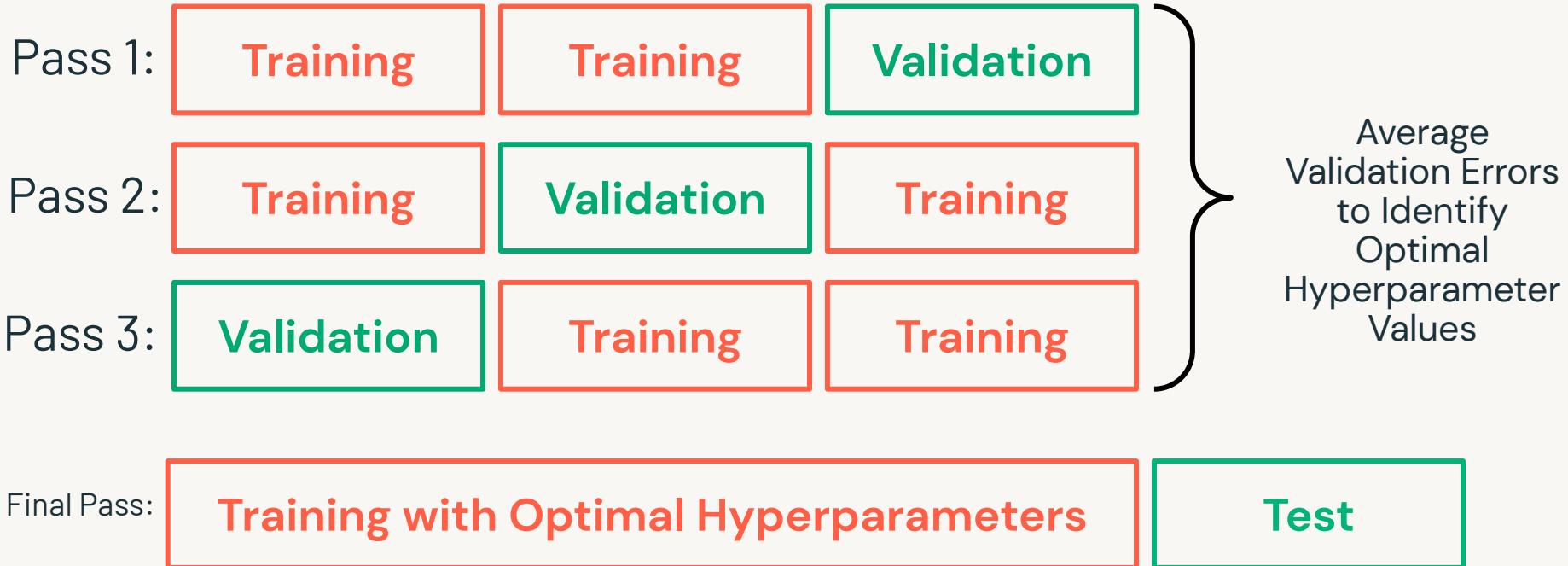
Validation

Test

What if there isn't enough data to split into three separate sets?



K-Fold Cross Validation



HYPERPARAMETER TUNING DEMO



Optimizing Hyperparameter Values

Grid Search

- Train and validate every unique combination of hyperparameters

Tree Depth	Number of Trees
5	2
8	4



Tree Depth	Number of Trees
5	2
5	4
8	2
8	4

Question: With 3-fold cross validation, how many models will this build?

HYPERPARAMETER TUNING LAB



Hyperparameter Tuning with Hyperopt



Problems with Grid Search

- Exhaustive enumeration is expensive
- Manually determined search space
- Past information on good hyperparameters isn't used
- So what do you do if...
 - You have a training budget
 - You have many hyperparameters to tune
 - You want to pick your hyperparameters based on past results

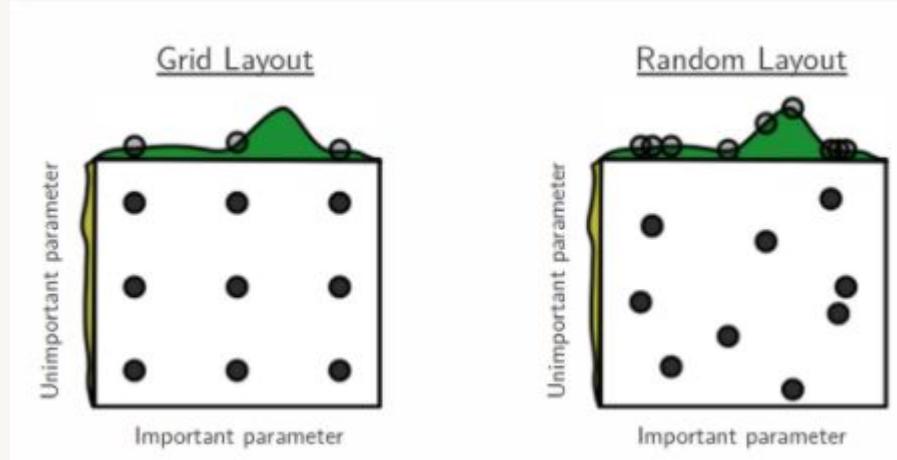


Hyperopt

- Open-source Python library
- Optimization over *awkward search spaces* (real-valued, discrete, and conditional dimensions)
- Supports serial or parallel optimization
- Spark integration
- Core algorithms for optimization:
 - Random Search
 - Adaptive Tree of Parzen Estimators (TPE)

Optimizing Hyperparameter Values

Random Search



Generally outperforms grid search



Optimizing Hyperparameter Values

Tree of Parzen Estimators

- Bayesian process
- Creates meta model that maps hyperparameters to probability of a score on the objective function
- Provide a range and distribution for continuous and discrete values
- Adaptive TPE better tunes the search space by
 - Freezing hyperparameters
 - Tuning number of random trials before TPE



HYPEROPT DEMO



HYPEROPT LAB



AutoML



Databricks AutoML

A glass-box solution that empowers data teams without taking away control

UI and API to start AutoML training

The screenshot shows the 'Configure AutoML experiment' interface. It includes a sidebar with 'AutoML Experiment Configuration' and 'Compute' sections, and a main area with tabs for 'Configure', 'Augment', and 'Preview'. Below these are sections for 'MLflow experiment' and 'Generated Trial Notebook'.

A screenshot of the MLflow experiment tracking interface. It displays a table with columns for Start Time, Run Name, User, and Source. Multiple rows are listed, each corresponding to a run initiated on May 5, 2021, using various models like logistic regression and decision trees.

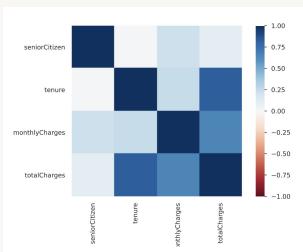
Start Time	Run Name	User	Source
2021-05-05 1	logistic_r...	kase...	Notebo...
2021-05-05 1	logistic_r...	alkis...	21-05...
2021-05-05 1	logistic_r...	alkis...	21-05...
2021-05-05 1	logistic_r...	kase...	Notebo...
2021-05-05 1	logistic_r...	kase...	Notebo...
2021-05-05 1	decision...	kase...	Notebo...
2021-05-05 1	random_f...	kase...	Notebo...

MLflow experiment

Auto-created MLflow Experiment to track models and metrics



Easily deploy to Model Registry



Data exploration notebook

Generated notebook with feature summary statistics and distributions



Understand and debug data quality and preprocessing

A screenshot of a Generated Trial Notebook in Python. The code is for Random Forest training, including loading data, defining preprocessors for numerical columns (One-hot encoding and Feature standardization), and training a classification model. The notebook also includes sections for 'Preprocessors' and 'Numerical columns'.

```
Generated Trial Notebook [Python]
dals_mir_8_new
Random Forest training
Load Data
Preprocessors
Numerical columns
One-hot encoding
Feature standardization
Train classification mo...
```

```
6
7 # Cho...
8 results.
9 examp...
10 # Use ...
11 predi...
12 expla...
13 shap...
14 summa...
```

Reproducible trial notebooks

Generated notebooks with source code for every model



Iterate further on models from AutoML, adding your expertise



AutoML solves two key pain points for data scientists

Quickly Verify the Predictive Power of a Dataset



"Can this dataset be used to predict customer churn?"

Get a Baseline Model to Guide Project Direction



"What direction should I go in for this ML project and what benchmark should I aim to beat?"



Problems with Existing AutoML Solutions

Opaque-Box and Production Cliff Problems in AutoML



Problem	Result / Pain Points
<ol style="list-style-type: none">1. A “production cliff” exists where data scientists need to modify the returned “best” model using their domain expertise before deployment2. Data scientists need to be able to explain how they trained a model for regulatory purposes (e.g., FDA, GDPR, etc.) and most AutoML solutions have “opaque box” models	<ul style="list-style-type: none">• The “best” model returned is often not good enough to deploy• Data scientists must spend time and energy reverse engineering these “opaque-box” returned models so that they can modify them and/or explain them



Configure

Configure AutoML experiment [Preview](#) [Provide Feedback](#)

Experiments > Configure AutoML experiment

1 Configure 2 Augment 3 Train 4 Evaluate

AutoML Experiment Configuration

* Compute
dais_mir_8_new
Select an existing cluster with a Databricks Runtime for ML 8.0

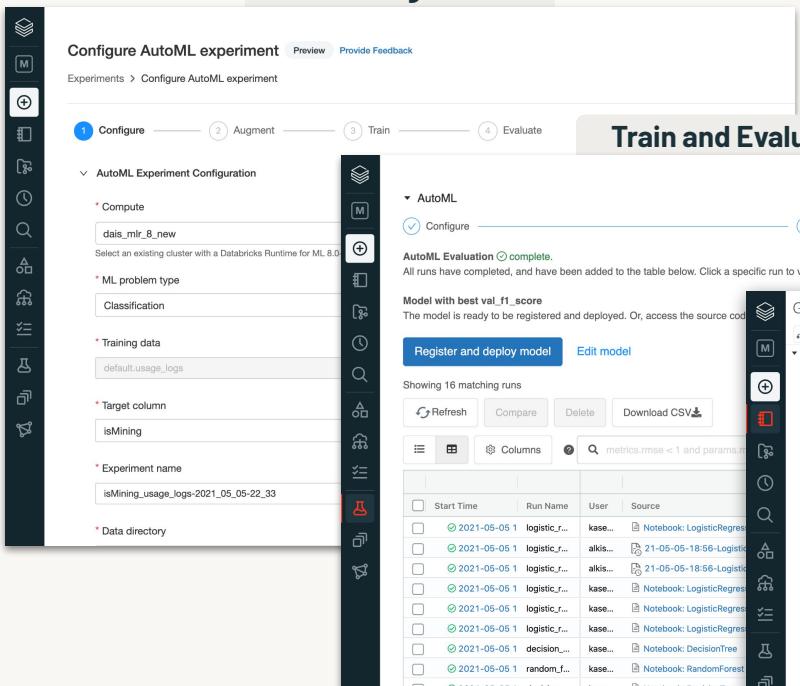
* ML problem type
Classification

* Training data
default_usage_logs

* Target column
isMining

* Experiment name
isMining_usage_logs-2021_05_05-22_33

* Data directory



Train and Evaluate

AutoML Evaluation [complete](#).

All runs have completed, and have been added to the table below. Click a specific run to view details or review the [data exploration notebook](#).

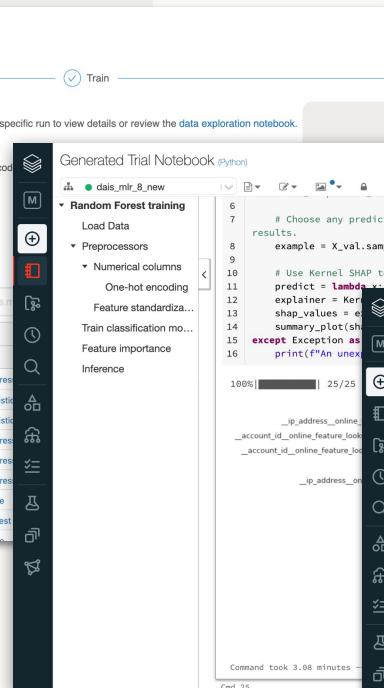
Model with best `val.f1_score`

The model is ready to be registered and deployed. Or, access the source code.

[Register and deploy model](#) [Edit model](#)

Showing 16 matching runs

Start Time	Run Name	User	Source
2021-05-05 1	logistic_r...	kase...	Notebook: LogisticRegr...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...
2021-05-05 1	logistic_r...	kase...	Notebook: LogisticRegr...
2021-05-05 1	logistic_r...	kase...	Notebook: LogisticRegr...
2021-05-05 1	logistic_r...	kase...	Notebook: LogisticRegr...
2021-05-05 1	decision_r...	kase...	Notebook: DecisionTree...
2021-05-05 1	random_f...	kase...	Notebook: RandomForest...
2021-05-05 1	decision_r...	kase...	Notebook: DecisionTree...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...
2021-05-05 1	logistic_r...	alkis...	21-05-05-18:56:Logisti...

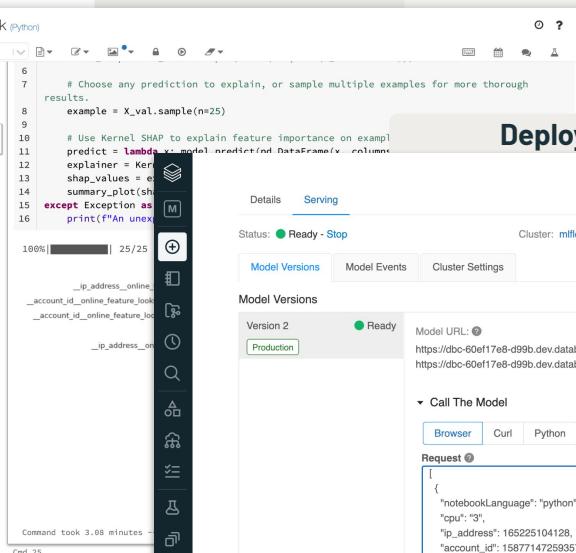


"Glass-Box" AutoML with a UI

Customize

Generated Trial Notebook (Python)

```
# Choose any prediction to explain, or sample multiple examples for more thorough results.  
example = X_val.sample(n=25)  
  
# Use Kernel SHAP to explain feature importance on example  
predict = lambda x: model.predict(x).DataFrame().columns  
explainer = KernelExplainer(predict, X_train)  
shap_values = explainer.shap_values[example]  
summary_plot(shap_values, X_train, example)  
except Exception as e:  
    print(f"An unexpected error occurred: {e}")
```



Deploy

Status: Ready - Stop Cluster: milflow-model-dais_demo

Model Versions

Version	Status
Version 2	Ready

Model URL: https://dbc-60ef17e8-d99b.dev.databricks.com/model/dais_demo/2/invocations
https://dbc-60ef17e8-d99b.dev.databricks.com/model/dais_demo/2/Production/invocations

Call The Model

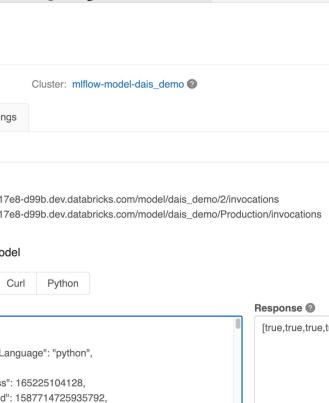
Request

```
[{"notebookLanguage": "python", "cpu": "3", "ip_address": "165.225.104.128", "account_id": "1587714725935792", "memberStartTime": "1614074788455,}]
```

Response

```
[true,true,true,true]
```

Send Request Show Example



Logs

Logs Version Events

```
$(GUNICORN_CMD_ARGS) --milflow-pyfunc.scoring._server.wsgi:app  
[2021-05-06 03:18:46 +0000] [8143] [INFO] Starting gunicorn 20.1.0
```

AutoML Lab

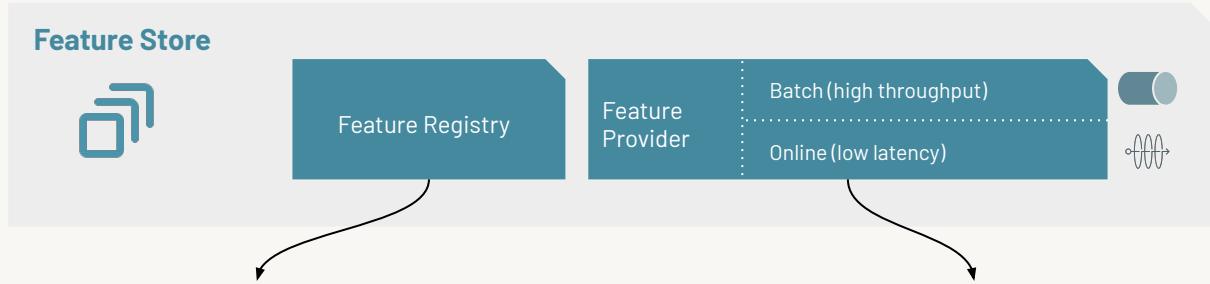


Feature Store



Feature Store

The first Feature Store codesigned with a Data and MLOps Platform



Feature Registry

- Discoverability and Reusability
- Versioning
- Upstream and downstream Lineage

Feature Provider

- Batch and online access to Features
- Feature lookup packaged with Models
- Simplified deployment process

Co-designed with  DELTA LAKE

- Open format
- Built-in data versioning and governance
- Native access through PySpark, SQL, etc.

Co-designed with  mlflow™

- Open model format that supports all ML frameworks
- Feature version and lookup logic hermetically logged with Model

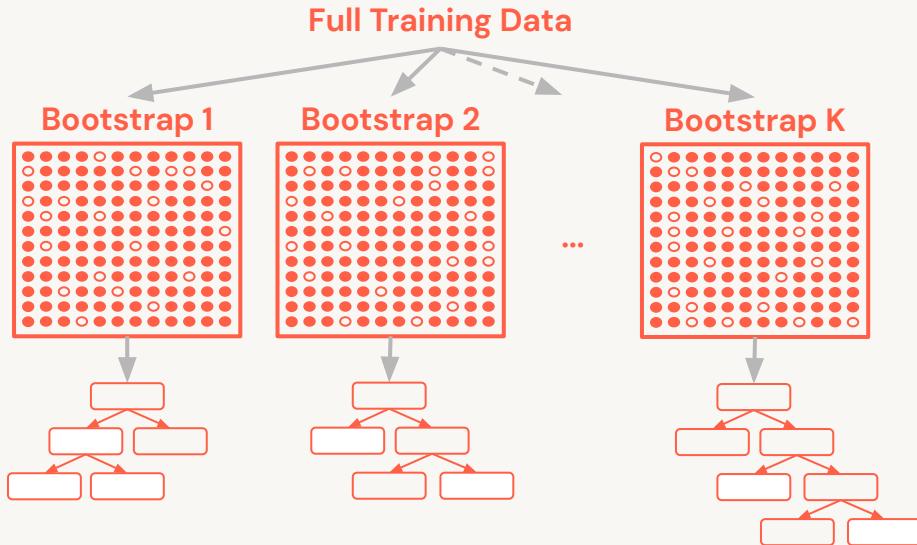


Gradient Boosted Decision Trees



Decision Tree Ensembles

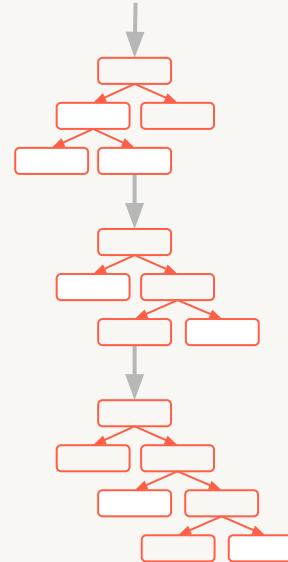
- Combine many decision trees
- Random Forest
 - Bagging
 - Independent trees
 - Results aggregated to a final prediction
- There are other methods of *ensembling* decision trees



Boosting

- Sequential (one tree at a time)
- Each tree learns from the last
- Sequence of trees is the final model

Full Training Data



Gradient Boosted Decision Trees

- Common boosted trees algorithm
- Fits each tree to the residuals of the previous tree
- On the first iteration, residuals are the actual label values

Model 1			Model 2			Final Prediction	
Y	Prediction	Residual	Y	Prediction	Residual	Y	Prediction
40	35	5	5	3	2	40	38
60	67	-7	-7	-4	-3	60	63
30	28	2	2	3	-1	30	31
33	32	1	1	0	1	33	32

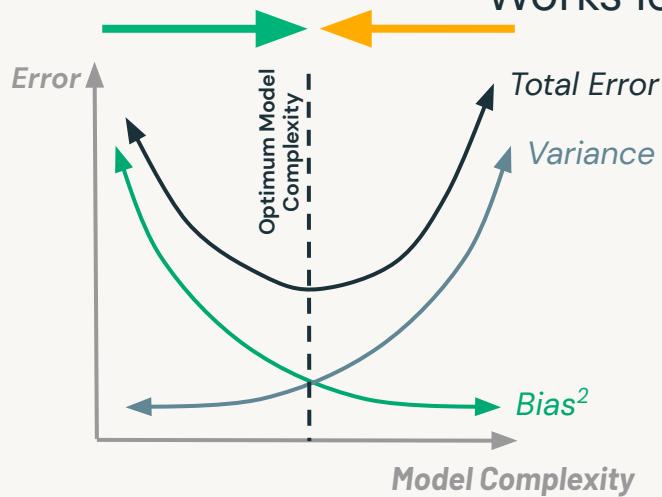
Boosting vs. Bagging

GBDT

- Starts with high bias, low variance
- Works right

RF

- Starts with high variance, low bias
- Works left



Gradient Boosted Decision Trees Implementations

- Spark ML
 - Built into Spark
 - Utilizes Spark's existing decision tree implementation
- XGBoost
 - Designed and built specifically for gradient boosted trees
 - Regularized to prevent overfitting
 - Pre-installed in Databricks Runtime for ML (Python & Scala APIs)



XGBOOST DEMO



Appendix



MLlib Deployment Options



Data Science vs. Data Engineering

- Data Science != Data Engineering
- Data Science
 - Scientific
 - Art
 - Business problems
 - Model mathematically
 - Optimize performance
- Data Engineering
 - Reliability
 - Scalability
 - Maintainability
 - SLAs



Model Operations (ModelOps)

- DevOps
 - Software development and IT operations
 - Manages deployments
 - CI/CD of features, patches, updates, and rollbacks
 - Agile vs. waterfall
- ModelOps
 - Data modeling and deployment operations
 - Java environments
 - Containers
 - Model performance monitoring

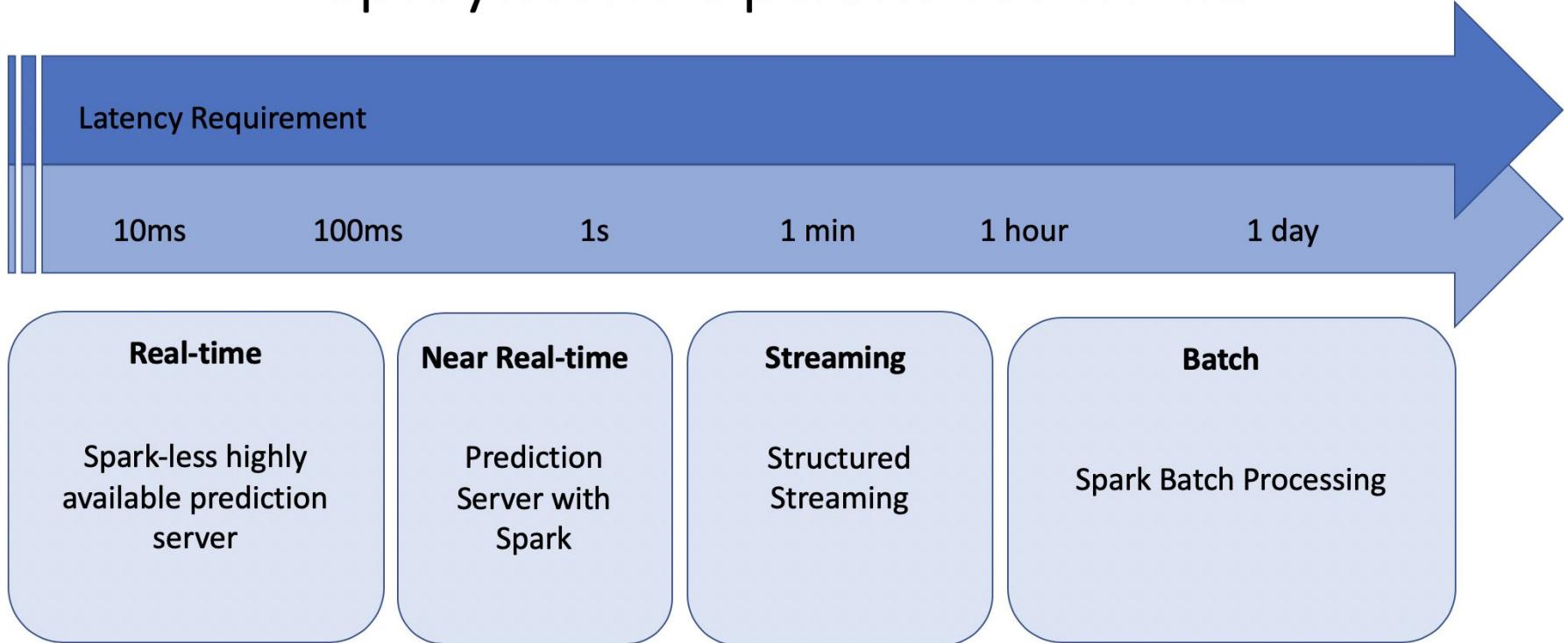


The Four ML Deployment Options

- Batch
 - 80–90 percent of deployments
 - Leverages databases and object storage
 - Fast retrieval of stored predictions
- Continuous/Streaming
 - 10–15 percent of deployments
 - Moderately fast scoring on new data
- Real-time
 - 5–10 percent of deployments
 - Usually using REST (Azure ML, SageMaker, containers)
- On-device



Deployment Options for MLlib



Overview of a typical Databricks CI/CD pipeline



[See CI/CD Templates for a starting point](#)



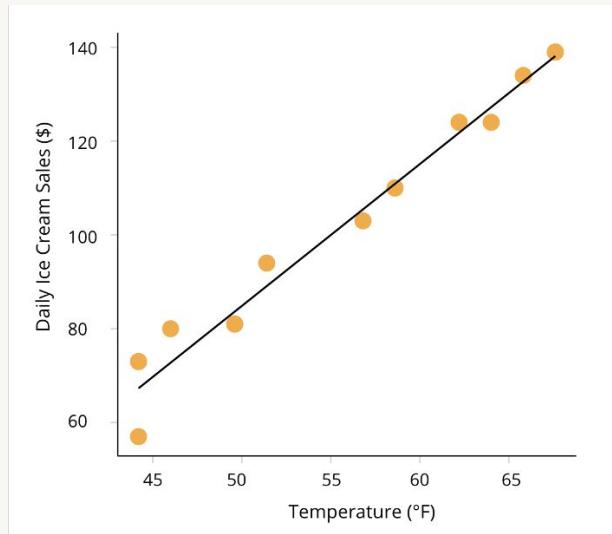
Logistic Regression



Types of Supervised Learning

Regression

- Predicting a continuous output



Classification

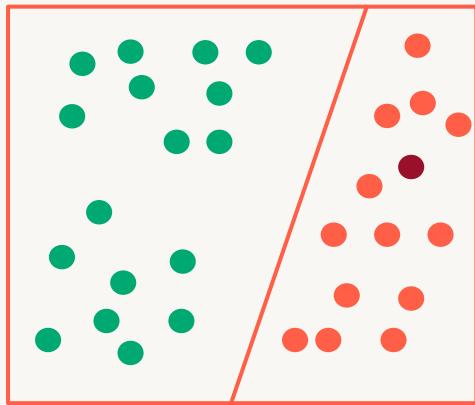
- Predicting a categorical/discrete output



Types of Classification

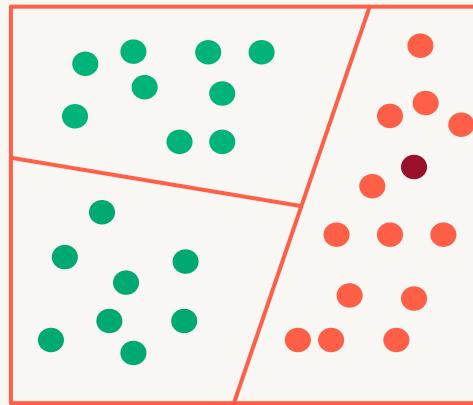
Binary Classification

Two label classes



Multiclass Classification

Three or more label classes



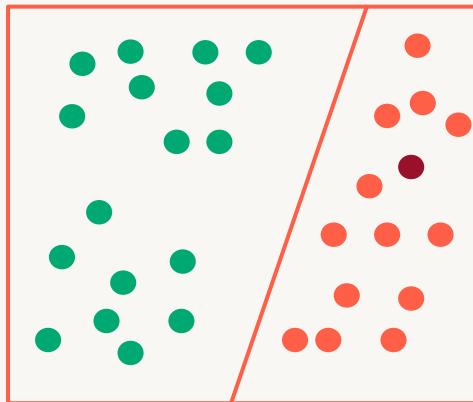
Model output is commonly the **probability** of a record belonging to **each of the classes**.



Binary Classification

Binary Classification

Two label classes

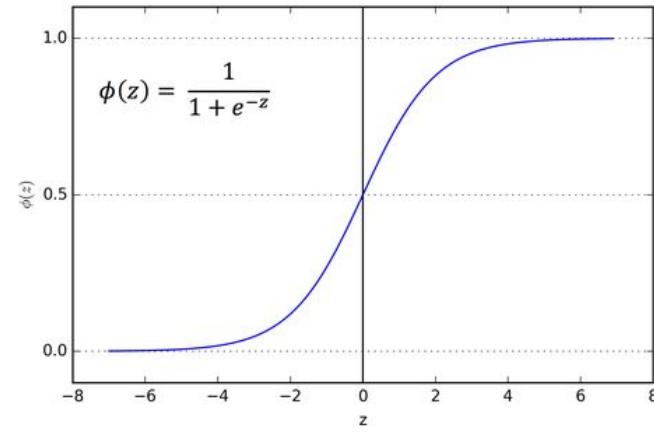


- Outputs:
 - Probability that the record is **Green** given a set of features
 - Probability that the record is **Red** given a set of features
- Reminders:
 - Probabilities are bounded between 0 and 1
 - And linear regression returns any real number



Bounding Binary Classification Probabilities

How can we keep model outputs between 0 and 1?



- Logistic Function:
 - Large positive inputs $\rightarrow 1$
 - Large negative inputs $\rightarrow 0$



Converting Probabilities to Classes

- In binary classification, the class probabilities are directly complementary
- So let's set our **Red** class equal to 1, and our **Blue** class equal to 0
- The model output is $P[y = 1 | x]$ where x represents the features

But we need **class** predictions, not **probability** predictions

- Set a threshold on the probability predictions
 - $P[y = 1 | x] < 0.5 \rightarrow y = 0$
 - $P[y = 1 | x] \geq 0.5 \rightarrow y = 1$



Evaluating Binary Classification Models

- How can the model be wrong?
 - Type I Error: False Positive
 - Type II Error: False Negative
- Representing these errors with a **confusion matrix**.

		Prediction	
		Positive	Negative
Actual	Positive	True Positive	False Negative
	Negative	False Positive	True Negative



Binary Classification Metrics

Accuracy

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Precision

$$\frac{TP}{TP + FP}$$

Recall

$$\frac{TP}{TP + FN}$$

F1

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

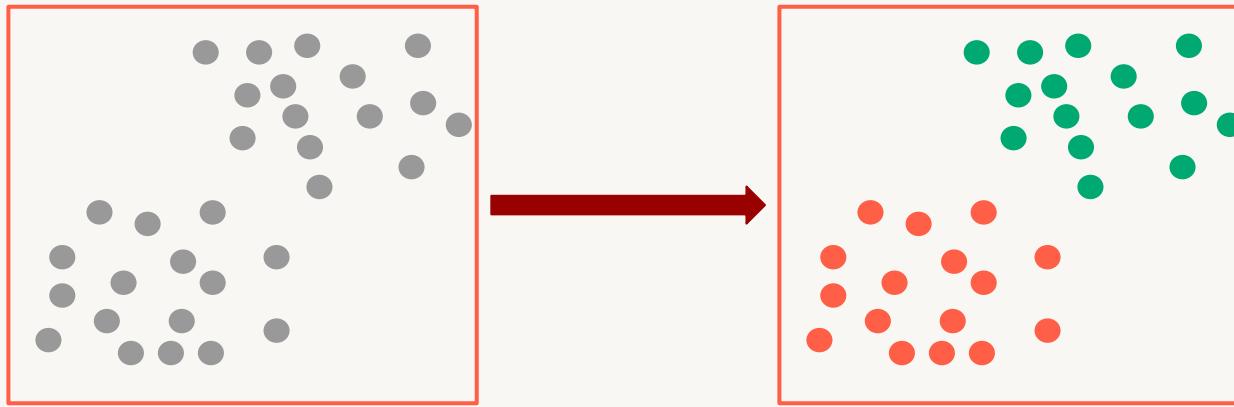


K-Means



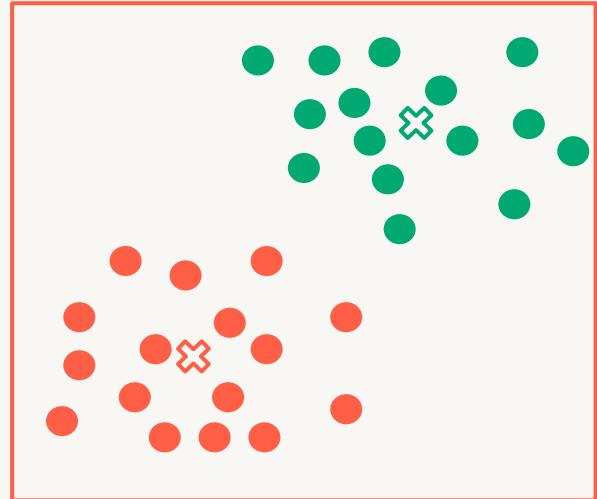
Clustering

- Unsupervised learning
- Unlabeled data (no known function output)
- Categorize records based on features



K-Means Clustering

- Most common clustering algorithm
- Number of clusters, K , is manually chosen
- Each cluster has a centroid
- Objective of minimizing the total distance between all of the points and their assigned centroid

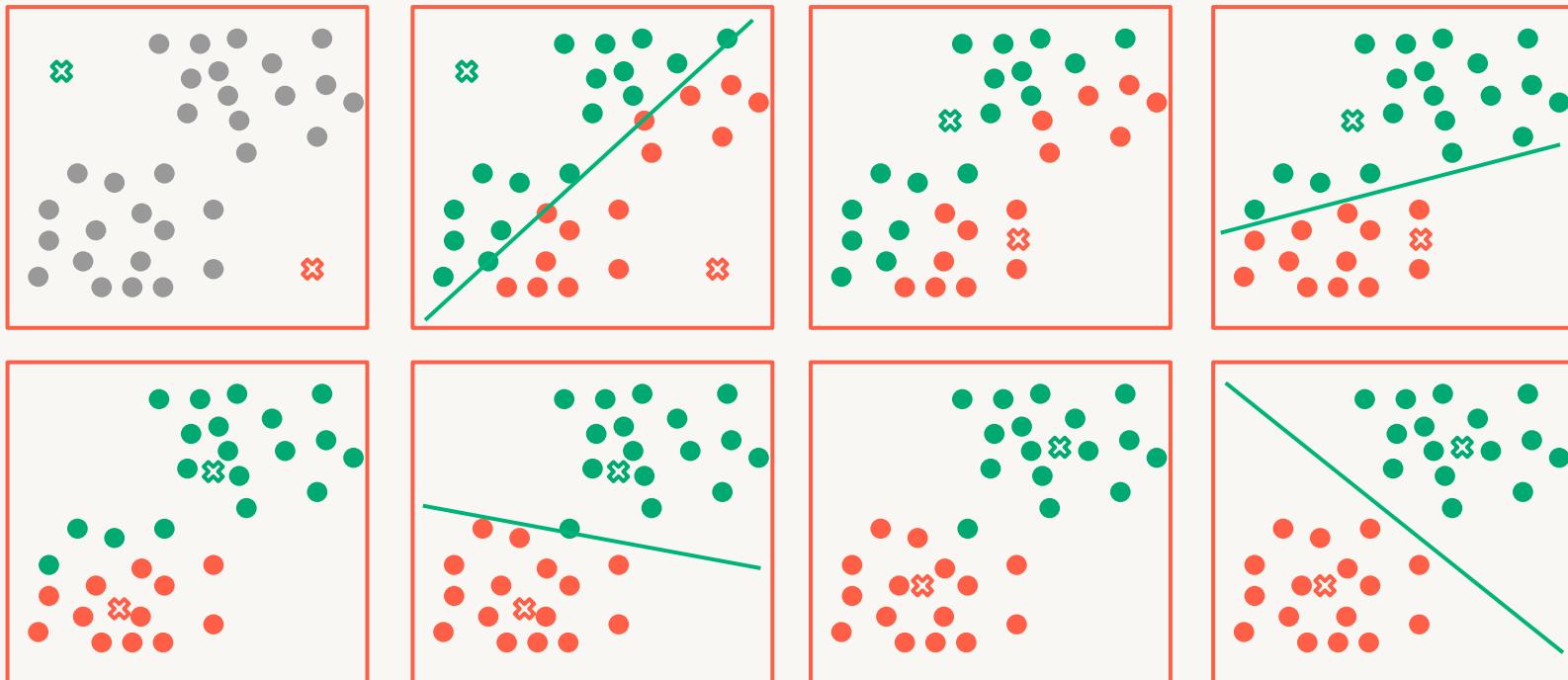


K-Means Algorithm

- **Step 1:** Randomly create centroids for k clusters
- Repeat until convergence/stopping criteria:
 - **Step 2:** Assign each data point to the cluster with the closest centroid
 - **Step 3:** Move the cluster centroids to the average location of their assigned data points



Visualizing K-Means



Choosing the Number of Clusters

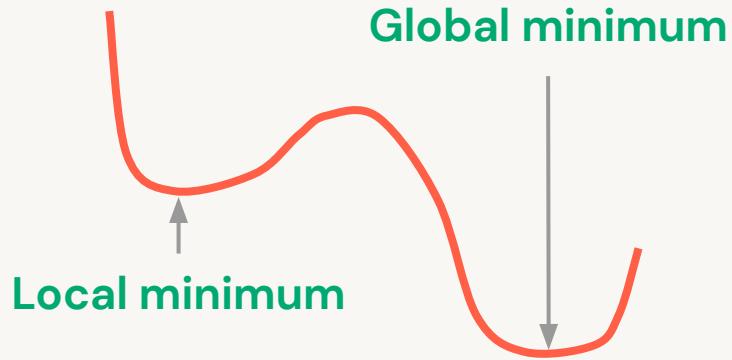
- K is a hyperparameter
- Methods of identifying the optimal K
 - Prior knowledge
 - Visualizing data
 - Elbow method for within-cluster distance

Note: Error will always decrease as K increases, unless a penalty is imposed.

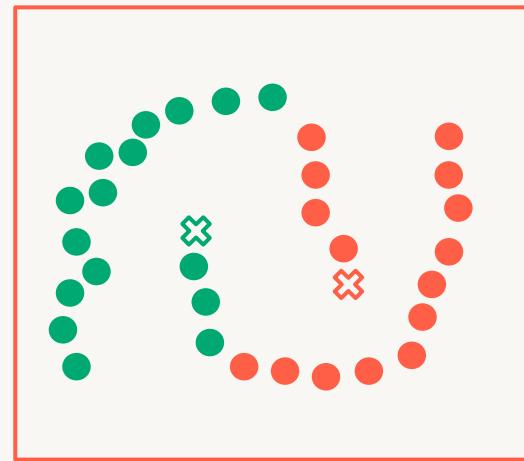


Issues with K-Means

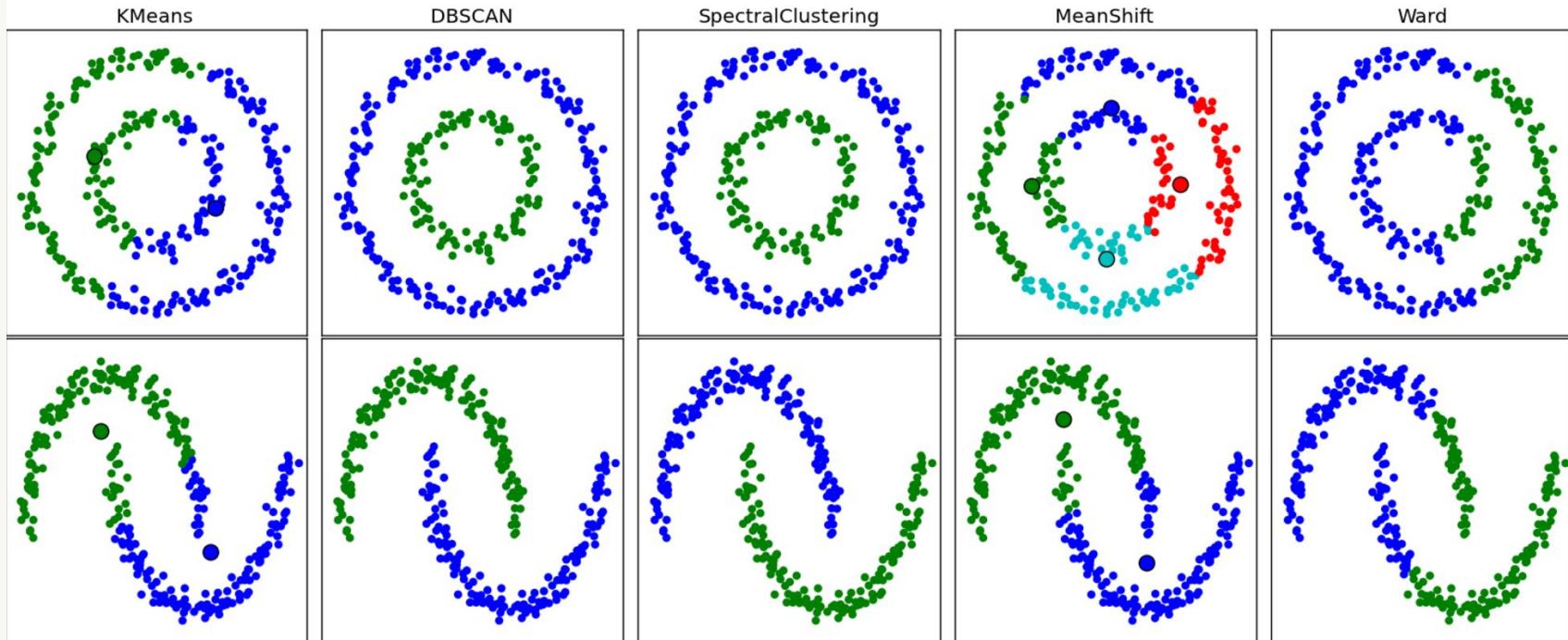
Local optima vs. global optima



Straight-line distance



Other Clustering Techniques



Collaborative Filtering



Recommendation Systems



Naive Approaches to Recommendation

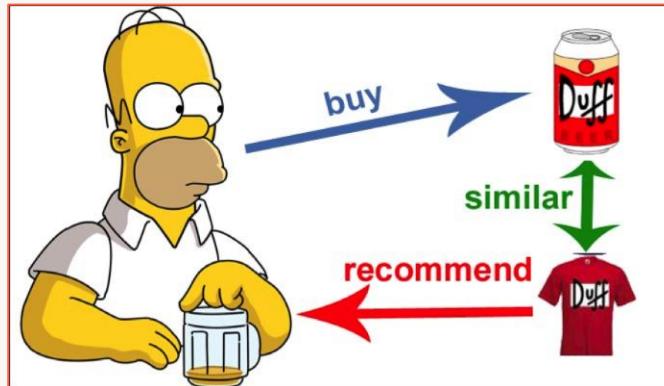
- Hand-curated
- Aggregates

Question: What are problems with these approaches?



Content-based Recommendation

- Idea: Recommend items to a customer that are *similar* to other items the customer liked
- Creates a profile for each user or product
 - User: demographic info, ratings, etc.
 - Item: genre, flavor, brand, actor list, etc.



Content-based Recommendation

- Advantages
 - No need for data from other users
 - New item recommendations
- Disadvantages
 - Cold-start problem
 - Determining appropriate features
 - Implicit information



Collaborative Filtering

- Idea: Make recommendations for one customer (filtering) by collecting and analyzing the interests of many users (collaboration)
- Advantages over content-based recommendation
 - Relies only on past user behavior (no profile creation)
 - Domain independent
 - Generally more accurate
- Disadvantages
 - Extremely susceptible to cold-start problem (user and item)



Types of Collaborative Filtering

- **Neighborhood Methods:** Compute relationships between items or users
 - Computationally expensive
 - Not empirically as good
- **Latent Factor Models:** Explain the ratings by characterizing items and users by small number of inferred factors
 - Matrix factorization
 - Characterizes both items and users by vectors of factors from item-rating pattern
 - Explicit feedback: sparse matrix
 - Scalable



Latent Factor Approach

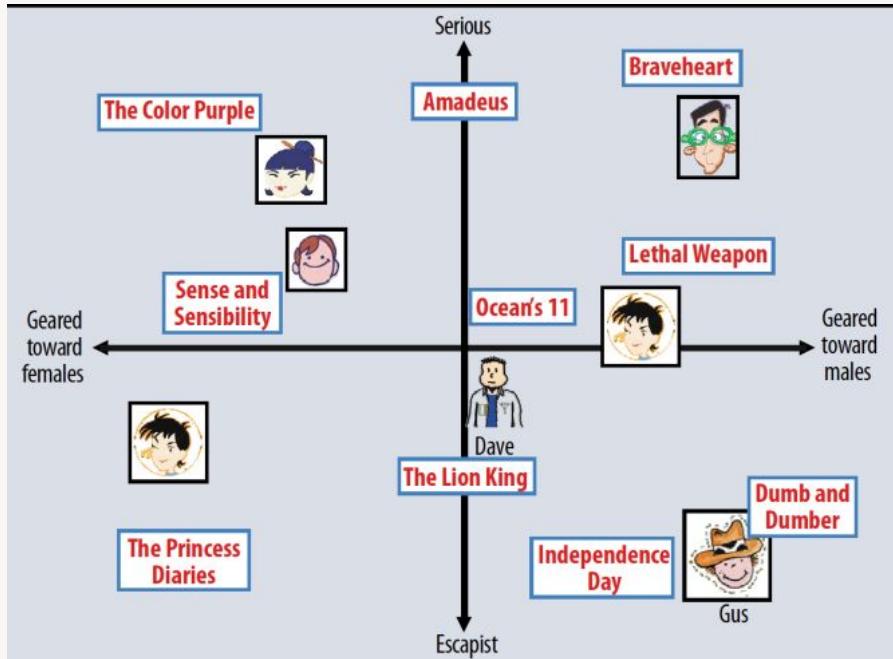
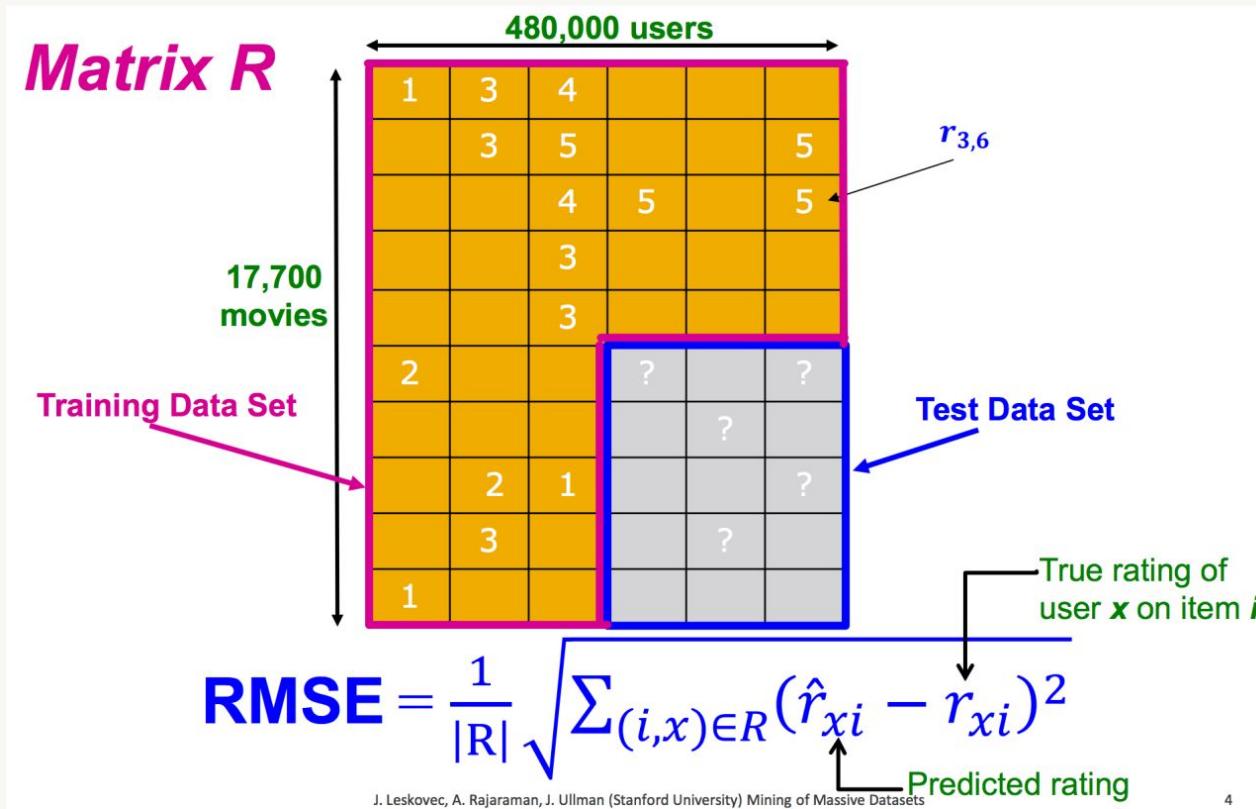


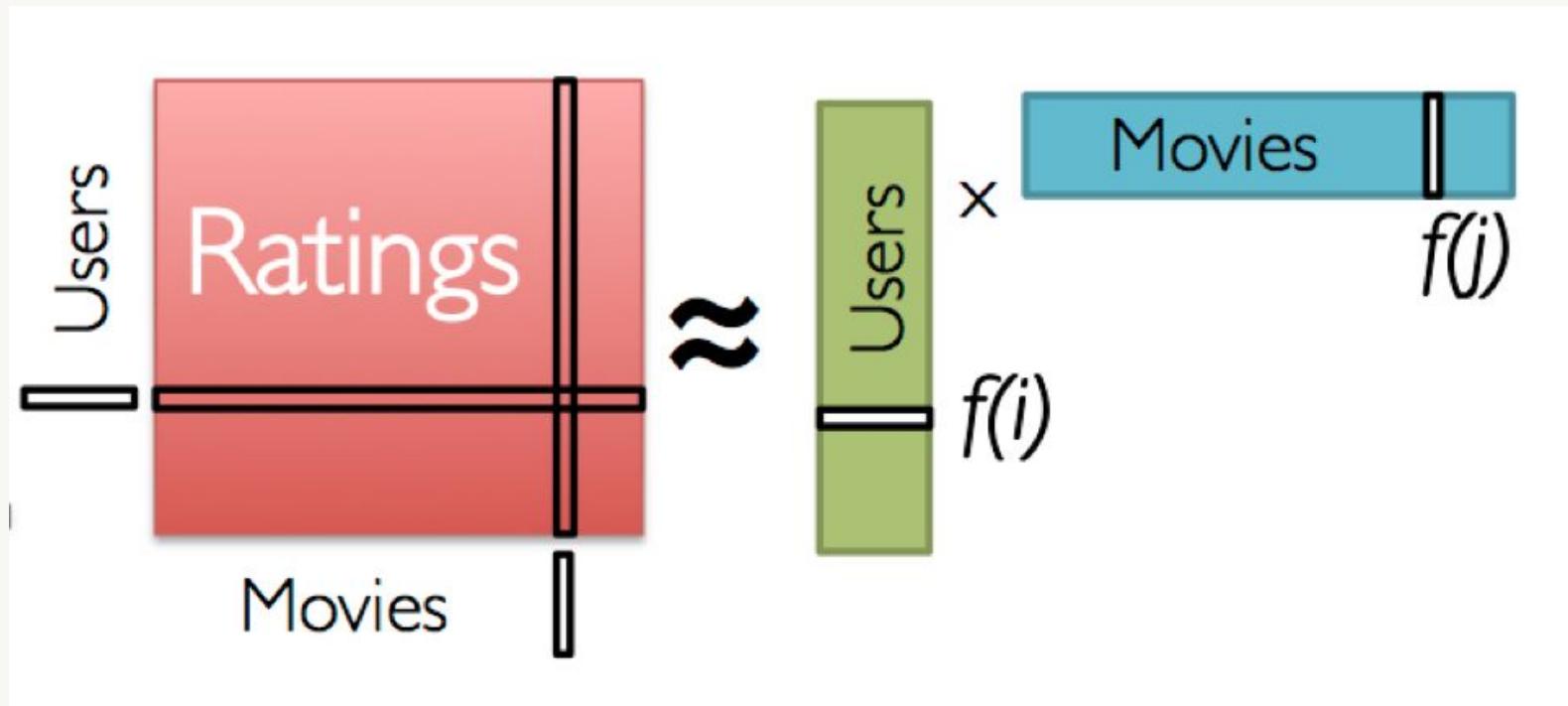
Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.



Ratings Matrix



Matrix Factorization



Alternating Least Squares

- **Step 1:** Randomly initialize user and movie factors
- **Step 2:** Repeat the following
 1. Fix the movie factors, and optimize user factors
 2. Fix the user factors, and optimize movie factors

$$\min_{q^*, p^*} \sum_{(u,i) \in R} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

