Day 13

Model Comparison & Feature Engineering

Databricks 14-Day AI Challenge

Yash Kavaiya

Agenda

- **Training Multiple Models**
  - Why compare different algorithms?
  - Common regression models
- **Hyperparameter Tuning**
  - Parameters vs Hyperparameters
  - Tuning techniques
- **Feature Importance**
  - Methods to calculate importance
  - Interpretation and use cases
- **Spark ML Pipelines**
  - Pipeline components
  - Building reproducible workflows
- **MLflow Integration**
  - Experiment tracking
  - Model comparison
- **Best Practices**
  - Model selection guide
  - Production deployment

Introduction

Model Comparison & Feature Engineering

Critical steps in the machine learning lifecycle that determine production success.

What We'll Cover:

- Training multiple regression models
- Tracking experiments with MLflow
- Building ML pipelines with Spark
- Understanding feature importance

Why It Matters:

- ▷ Rarely deploy the first model
- ▷ Systematic comparison needed
- ▷ Better model = better business outcomes

Why Train Multiple Models?

The "No Free Lunch" Theorem

No single algorithm works best for every problem — each has unique strengths!

1. Compare Performance

2. Trade-off Analysis

3. Reduce Bias

4. Build Ensembles

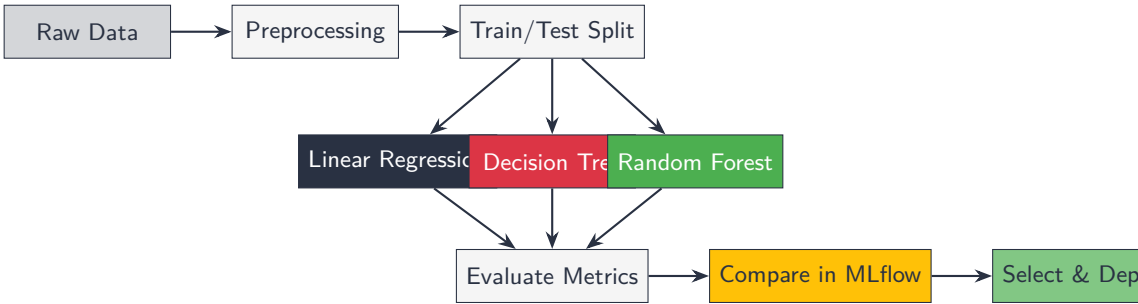
- See which algorithm captures patterns best
- Balance accuracy vs. interpretability
- Avoid single algorithm assumptions
- Combine models for better predictions

Common Regression Models

Model	Type	Strengths	Weaknesses	Interpretability
Linear Regression	Parametric	Fast, Simple	Linear assumption	High
Decision Tree	Non-parametric	Handles non-linearity	Prone to overfit	Medium
Random Forest	Ensemble	Reduces overfitting	Slower, complex	Low

Key Insight: Start simple (Linear Regression) and increase complexity only when needed!

Model Selection Strategy



What are Hyperparameters?

Definition

Configuration settings that control the learning process — must be set **before** training begins.

Aspect	Parameters	Hyperparameters
Definition	Learned from data	Set before training
Examples	Coefficients, weights	Learning rate, tree depth
How Set	Optimization algorithm	Manual or automated search
Location	Inside the model	Outside the model

Key Hyperparameters by Model

- `max_depth`
  - Maximum depth of tree
  - Controls complexity
- `min_samples_split`
  - Min samples to split node
- `min_samples_leaf`
  - Min samples at leaf

Random Forest Regressor

- `n_estimators`
  - Number of trees in forest
- `max_depth`
  - Max depth of each tree
- `max_features`
  - Features for best split

Hyperparameter Tuning Techniques

Grid Search

Exhaustive search through specified parameter grid

✓ Finds best combination
× Computationally expensive

Random Search

Samples random combinations from distributions

✓ More efficient
× May miss optimal

Bayesian Opt

Probabilistic models learn from evaluations

✓ Most efficient
× Complex setup

Grid Search Example

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    DecisionTreeRegressor(),
    param_grid,
    cv=5,  # 5-fold cross-validation
    scoring='r2'
)

grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
```

Understanding Feature Importance

What is Feature Importance?

Quantifies how much each input feature contributes to the model's predictions.

Why It Matters:

- **Understand the model**
  - Know what drives predictions
- **Feature selection**
  - Remove irrelevant features
- **Business insights**
  - Identify key factors
- **Model debugging**
  - Detect unexpected reliance

Methods to Calculate Feature Importance

For Linear Regression:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Importance of feature i: $|\beta_i|$

Gini Importance

Mean Decrease in Impurity (MDI):

$$\text{Imp}(f) = \sum_t \frac{n_t}{N} \cdot \Delta\text{impurity}_t$$

Permutation Importance

Decrease in performance when feature shuffled:

$$\text{Imp}(f) = \text{Score}_{orig} - \text{Score}_{perm}$$

Model-agnostic: Works with any model
Robust: Based on actual predictions

What is a Spark ML Pipeline?

Definition

A sequence of stages that transform data and train models — chains transformers and estimators into a single workflow.

Reproducibility
Same transformations consistently

Simplicity
Complex workflows as single objects

No Data Leakage
Transformations fit only on train

Pipeline Components

Convert one DataFrame into another (no learning)

`VectorAssembler`	Combine cols → vector
`StringIndexer`	Strings → numeric
`StandardScaler`	Mean=0, Std=1
`OneHotEncoder`	One-hot vectors

Estimators

Learn from data, produce a Model

`LinearRegression`	→ LRModel
`DecisionTreeRegressor`	→ DTModel
`RandomForestRegressor`	→ RFModel

VectorAssembler Example:

`[views=100, cart_adds=5]` → `features=[100.0, 5.0]`

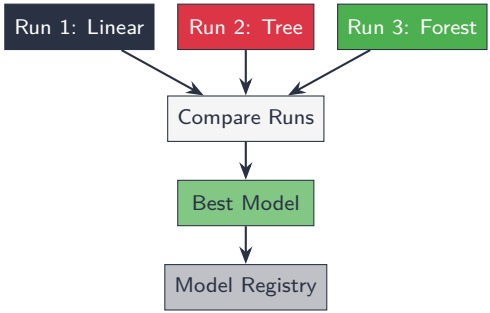
Building a Spark ML Pipeline

```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression as SparkLR

# Stage 1: Combine features into vector
assembler = VectorAssembler(
    inputCols=["views", "cart_adds"],
    outputCol="features"
)

# Stage 2: Linear Regression
lr = SparkLR(featuresCol="features", labelCol="purchases")

# Create and fit pipeline
pipeline = Pipeline(stages=[assembler, lr])
model = pipeline.fit(train_df)

# Make predictions
predictions = model.transform(test_df)
```

MLflow Integration for Model Tracking

MLflow Provides:

- **Experiment Tracking**
  - Log params, metrics, artifacts
- **Model Registry**
  - Version and manage models
- **Model Deployment**
  - Deploy as REST APIs



Key MLflow Functions

Function	Purpose	Example
`start_run()`	Start a new run	`with mlflow.start_run():`
`log_param()`	Log a parameter	`mlflow.log_param("depth", 5)`
`log_metric()`	Log a metric	`mlflow.log_metric("r2", 0.85)`
`log_artifact()`	Log a file	`mlflow.log_artifact("plot.png")`
`sklearn.log_model()`	Log sklearn model	`mlflow.sklearn.log_model(...)`

Tip: Use `with mlflow.start_run():` for automatic run closure!

Training Loop with MLflow

```python
models = {
    "linear": LinearRegression(),
    "decision_tree": DecisionTreeRegressor(max_depth=5),
    "random_forest": RandomForestRegressor(n_estimators=100)
}

for name, model in models.items():
    with mlflow.start_run(run_name=f"{name}_model"):
        # Log model type
        mlflow.log_param("model_type", name)

        # Train and evaluate
        model.fit(X_train, y_train)
        score = model.score(X_test, y_test)

        # Log metrics and model
        mlflow.log_metric("r2_score", score)
        mlflow.sklearn.log_model(model, "model")

        print(f"{name}: R2 = {score:.4f}")
```

Understanding R^2 (Coefficient of Determination)

Formula

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

Calculation Steps:

1. Calculate mean: $\bar{y} = \frac{1}{n}\sum y_i$
2. Total variance: SS_{tot}
3. Residual variance: SS_{res}
4. Compute R^2

Interpretation:

- $R^2 = 1.0$: Perfect predictions
- $R^2 = 0.0$: Predicts mean only
- $R^2 < 0$: Worse than mean

Regression Metrics Comparison

Metric	Formula	Use Case
MSE	$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$	Penalizes large errors
RMSE	$\sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$	Same units as target
MAE	$\frac{1}{n}\sum\lvert y_i - \hat{y}_i\rvert$	Robust to outliers
MAPE	$\frac{100}{n}\sum\left\lvert\frac{y_i - \hat{y}_i}{y_i}\right\rvert$	Percentage error

Pro Tip: Use RMSE when large errors are particularly undesirable; use MAE when you want robustness to outliers.

Model Comparison Example

Model	R^2 Score	Training Time	Interpretability
Linear Regression	0.7234	0.5s	High
Decision Tree	0.7856	1.2s	Medium
Random Forest	**0.8421**	8.5s	Low

Selection Factors:

- Performance (R^2 score)
- Training speed
- Interpretability needs
- Business requirements

Winner:
Random Forest
(if accuracy is priority)

Best Practices

Data Preparation

- Split data before preprocessing
- Use cross-validation
- Handle missing values consistently

Model Training

- Start simple before complex
- Use consistent random seeds
- Log everything with MLflow

Feature Engineering

- Use domain knowledge
- Remove correlated features
- Scale when needed

Pipeline Design

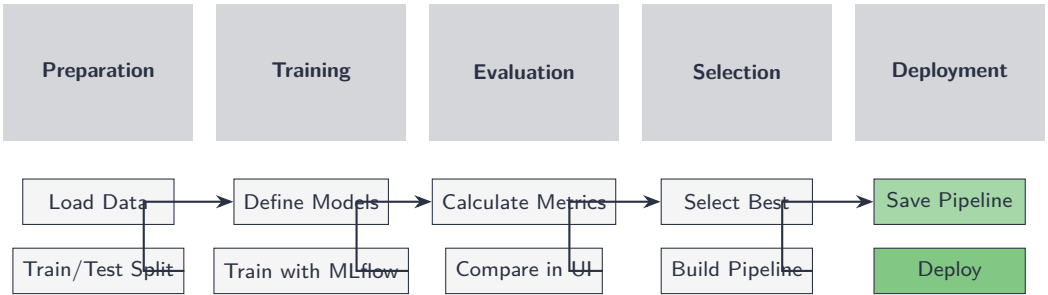
- Include all preprocessing
- Save fitted pipelines
- Version your pipelines

Quick Reference: Model Selection Guide

Situation	Recommended Model
Need interpretability	Linear Regression
Non-linear relationships	Decision Tree / Random Forest
High accuracy required	Random Forest (with tuning)
Large distributed data	Spark ML Pipeline
Quick baseline	Linear Regression

Golden Rule: Always start with a simple baseline model, then increase complexity only when justified by significant performance improvements.

Complete Workflow Summary



Key Takeaways

1. Multiple Models No single model works best for all problems

2. MLflow Tracking Enables reproducibility and comparison

3. Spark ML Pipelines Chain transformations for production

4. Feature Importance Understand what features matter

5. Hyperparameters `max_depth` and `n_estimators` control complexity

Thank You!

Day 13: Model Comparison & Feature Engineering

Connect with me:

LinkedIn: Yash Kavaiya

Gen AI Guru

Easy AI Labs