**Databricks ML Issues with Mitigations**

**Databricks ML Issues Comparison Table with Mitigations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | ML Issue | One-Liner Description | Where It Typically Arises | ML Areas Most Affected | Mitigation Strategies |
| 1 | Environment Drift | Inconsistent Python libraries and environment dependencies break model reproducibility. | Model training and serving | Experiments, Model Registry | Use MLflow conda environments; pin dependency versions explicitly; validate environments before promotion. |
| 2 | Model Version Confusion | Teams lose track of which model is the latest or production-ready version. | Multi-stage model development | Model Registry, Deployment Pipelines | Register all models in MLflow; use stage transitions (Staging, Production) with approval workflows. |
| 3 | Data Leakage | Inadvertent use of target variables or future data skews model performance metrics. | Feature engineering, training pipelines | Training Data, Feature Store | Implement strict data splits; validate feature generation pipelines for leakage risks. |
| 4 | Feature Inconsistency | Discrepancies between training and serving features cause prediction errors. | Feature Store, real-time inference | Model Serving, Feature Store | Use Feature Store to centralize feature logic; ensure online and offline consistency. |
| 5 | Inefficient Experiment Tracking | Experiments lack clear metadata and parameter tracking, complicating reproducibility and comparison. | Ad hoc experiments | MLflow Experiments | Log all runs to MLflow with parameters, metrics, artifacts, and tags. |
| 6 | Overfitting | Models perform well on training data but fail to generalize to new data. | Model training | Experiments, Model Registry | Use cross-validation and regularization; monitor validation performance metrics carefully. |
| 7 | Resource Overconsumption | Large-scale training jobs exhaust cluster resources, delaying other workloads. | Distributed training | Clusters, Workflows | Use autoscaling clusters with limits; monitor resource usage; schedule training during off-peak hours. |
| 8 | Inconsistent Model Deployment | Manual deployment steps lead to mismatched environments and errors in production. | Model serving, API deployment | Model Serving Endpoints | Automate deployment pipelines with MLflow Model Registry; validate environments during promotion. |
| 9 | Lack of Monitoring and Drift Detection | No visibility into prediction accuracy or data drift after deployment. | Production model serving | Serving Endpoints, Monitoring | Implement prediction logging and model performance monitoring; set alerts on drift thresholds. |
| 10 | Poor Governance and Access Control | Model artifacts and experiment data are accessible without appropriate controls. | Multi-team environments | Model Registry, Experiments | Use Unity Catalog and RBAC to control access to ML artifacts and training data. |

**Quick Reference**

* **MLflow:** Databricks-native framework for experiment tracking, model management, and deployment.
* **Model Registry:** Central hub for model versioning, staging, and promotion.
* **Feature Store:** Managed repository for storing and serving features.
* **Serving Endpoint:** Production API for real-time inference.
* **Drift Detection:** Monitoring for data distribution changes over time.

**Example Mitigation Commands and Configurations**

**Track Experiments with MLflow:**

python

CopyEdit

import mlflow

with mlflow.start\_run():

mlflow.log\_param("learning\_rate", 0.01)

mlflow.log\_metric("accuracy", 0.92)

mlflow.log\_artifact("model.pkl")

**Register and Transition Models:**

python

CopyEdit

result = mlflow.register\_model(

"runs:/<run-id>/model",

"credit\_risk\_model"

)

from mlflow.tracking import MlflowClient

client = MlflowClient()

client.transition\_model\_version\_stage(

name="credit\_risk\_model",

version=result.version,

stage="Staging"

)

**Log Features to Feature Store:**

python

CopyEdit

from databricks.feature\_store import FeatureStoreClient

fs = FeatureStoreClient()

fs.create\_table(

name="churn\_features",

primary\_keys=["customer\_id"],

schema=df.schema,

description="Features for churn prediction"

)

**Enable Model Serving:**

* In the UI: *Models > Serving > Enable Endpoint*
* Or via API: Use MLflow to deploy as REST endpoint.

**Monitor Prediction Drift:**

* Export predictions to Delta tables.
* Use scheduled notebooks to compare distributions over time.

**Restrict Access with Unity Catalog:**

sql

CopyEdit

GRANT SELECT ON FEATURE STORE churn\_features TO `ml\_team`;