Quest Journals
Journal of Research in Business and Management
Volume 13 ~ Issue 10 (October 2025) pp: 48-52
ISSN(Online):2347-3002
www.questjournals.org

Research Paper

Efficient and Accelerated ML model development using Databricks Feature Store

Vamshi Krishna Dasarraju
Independent researcher

Abstract
Feature engineering is the vital step in machine learning, it is a process of converting raw data into features that can be used to train and infer machine learning models. Improving the features used for training and for inference has huge impact on model quality and accuracy in its predictions. In feature engineering many data challenges are faced such as data silos, offline/online skew, client configurations and data freshness. To address these challenges this paper proposes the implementation of Databricks FeatureStore. A FeatureStore is a centralized repository of features across an organization. It helps to discover and share features along with tracking their lineage. The results of adopting the FeatureStore in machine learning pipelines include removal of duplicate feature computation, faster feature discovery due to centralized storage, eliminates training/serving skew and accurate historical lookups. As organizations adopt distributed and cloud-based analytics platforms, Databricks FeatureStore helps them by providing the unified, scalable and reproducible approach to manage features across the organization.

Keywords: *Databricks, FeatureStore, Feature Engineering, Unity Catalog, Datalake, MLFlow, Machine Learning, Delta Lake*

I. Introduction

Before we delve deeper into feature engineering, let's first clarify exactly what features are.
Features: These are the fundamental inputs that machine learning algorithms use to learn patterns and make predictions about new data. The quality and relevance of features directly impact how accurate a model's predictions will be.
Below are the types of features:

- **Continuous Features:** Numerical values that fall within a range, such as a house's square footage or temperature.
- **Categorical Features:** Distinct categories that represent different groups, like the number of bedrooms in a house.
- **Ordinal Features:** Categories with a specific, ranked order, such as the age of a house or age group.

Feature engineering: It is the process of turning raw data into useful features that help improve the performance of machine learning models and to extract patterns effectively. It involves selecting, extracting, and sometimes creating new features from the available data to improve the performance of machine learning models[[4]].

Below are the steps involved in the feature engineering process:

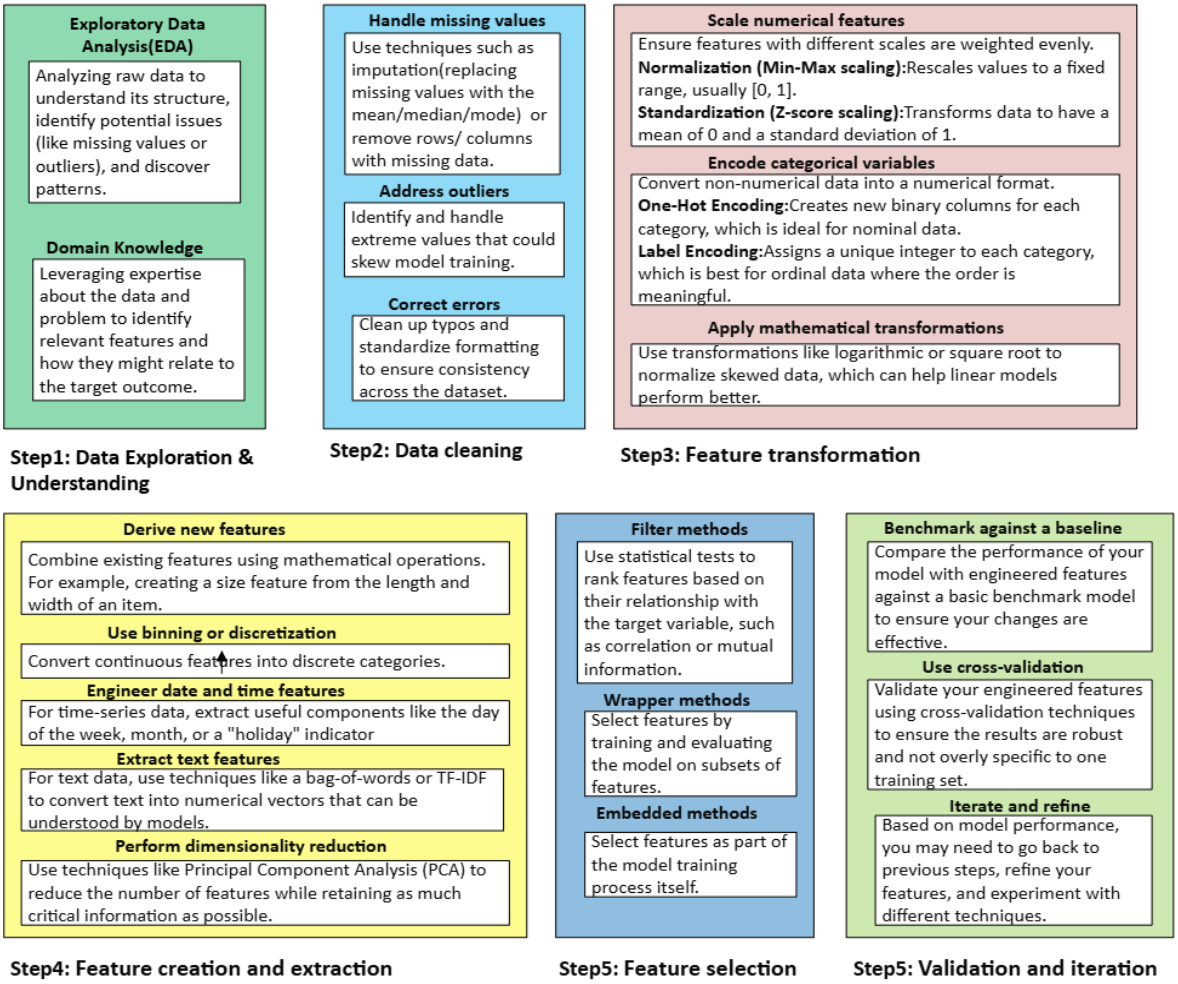

Figure 1:Feature Engineering Steps

Traditional Feature Engineering tools

- Data Storage and Management Systems (Data Warehouses, Data Lakes, File Storage, Block Storage, Object Storage)
- Open Transformation Languages (SQL, Python, Spark, etc.)
- Compute to run the transformations (Microsoft Azure, AWS, GCP etc.)
- Feature engineering packages & libraries (Scikit-learn, Feature-engine, Featuretools, Pandas, tsfresh, MLxtend etc.)
- Data Version Control tools

However, the above specified traditional feature engineering process and tooling is fragmented, leading to inefficiencies, duplication, and governance issues.

Databricks Feature Store

The Databricks Feature Store is a centralized repository for machine learning features, allowing data scientists to discover, share, and reuse features across different models and teams. Key capabilities include automatic lineage tracking, online and offline feature serving for real-time and batch predictions, and integration with Unity Catalog for built-in governance and discovery[[1]].

The Databricks Feature Store with its deep integration with Delta Lake, MLflow, Unity Catalog and Databricks workflows provides reliable and scalable feature engineering.

Let's explore the key challenges of managing ML data in depth and examine how Databricks Feature Store, together with Delta Lake, MLflow, Unity Catalog, and Databricks Workflows, effectively addresses and overcomes these data challenges.

Data Challenges and Solution Descriptions
Challenge 1-Data Silos:
Challenge Description
To train a ML model raw data sources are queried, joined aggregated and transformed into features. However, after ML model training, these transformations and raw data sources are often discarded. But these assets are essential for model reproducibility and model retraining. Features often remain undocumented and are not easily discoverable or reusable across the organization. This results in redundant effort and ends up in technical debt.
Different teams store their features in different locations and different file formats. Sometimes inaccessible to other teams. The result is redundant featurization pipelines.
Data silos significantly complicate feature engineering for machine learning by fragmenting valuable information and creating inconsistent, incomplete, and difficult-to-access datasets.

Solution Description:
Databricks FeatureStore addresses this challenge by providing a centralized and standardized platform for managing and Sharing Features. The FeatureStore acts as a single source of truth for features preventing duplication of feature creation effort and ensures consistency[[3]].
Databricks' Feature Store integrates with Unity Catalog to provide a centralized and governed approach to managing features. Fosters collaboration between various teams and helps them to leverage the pre-computed features which results in acceleration of model development and deployment[[5]].
Databricks FeatureStore lineage tracking provides us the visibility into the origin of features, its usage and which job or notebook making changes to features. It also ensures compliance to regulatory requirements by providing comprehensive lineage trail.

Feature's lineage information is stored in Unity Catalog and can be retrieved by querying the following tables:

- **system.access.table_lineage:** This table creates a record for each read or write event on a Unity Catalog Feature table. This includes, but is not limited to, job runs, notebook runs, and dashboards updated with the read or write event.
- **system.access.column_lineage:** This table provides column-level lineage information. Allowing users to track the flow of data at a granular level.

Challenge 2-Online/Offline Skew:
Challenge Description
Once the ML models are trained, they will be deployed for real time inference for use cases such as fraud detection, fetching features at low latency is critical for those kinds of use cases. Models are required to use the same features at inference time as used at training time. Most of the time featurization codes written for the purpose of training will not be performant when used for inference purposes. Hence the development team will end up replicating the feature transformation code for low latency inference. Both versions of featurization code should be exactly same, otherwise models will end up performing very well during training & evaluation phases but may perform very poor during online inference. This problem is called Online (Realtime)/Offline (Training time) skew.

Solution Description:
The Databricks Feature Store addresses online-offline skew by ensuring consistency in feature computation and usage across the entire machine learning lifecycle. This means that the same code and logic used to generate features for model training (offline) are also used to retrieve features for real-time inference (online).
Below are the steps to be followed to ensure same feature computation is used during the model training stage and inference stage:

1. Initially define a Feature Computation Wrapper class inherited from mlflow.pyfunc.PythonModel base class
2. Override the base class fit method which includes a call to the custom feature transformation function first followed by train function
3. Override the base class predict function which takes inference input as a parameter and call the custom feature transformation function followed by predict function which takes transformed input as the parameter

```
# Define the model class with on-demand computation model wrapper
class OnDemandComputationModelWrapper(mlflow.pyfunc.PythonModel):

    def fit(self, X_train: pd.DataFrame, y_train: pd.DataFrame):
        try:
            X_train = self._preprocessing(X_train)

            self.model = lgb.train(
              {"num_leaves": 32, "objective": "binary"},
              lgb.Dataset(X_train, label=y_train.values),
              5)
            logging.info(f"trained model on dataframe : {X_train}")
        except Exception as e:
            logging.error(e)

    # Add preprocessing code for on-demand Feature computation
    def _preprocessing(self, model_input: pd.DataFrame)->pd.DataFrame:
      try:
          col_name = "impression_durations"
          model_input["impression_durations"] = model_input[col_name].apply(lambda x: sum(x) / len(x))
      except Exception as e:
            logging.error("inside custom model _preprocessing")
            logging.error(e)
            raise e
      return model_input

    def predict(self, context, model_input):
        new_model_input = self._preprocessing(model_input)
        return  self.model.predict(new_model_input)
```

Figure 2: Feature Computation Wrapper Python Code

4. Prepare the training data which includes all the features and corresponding labels.

```
import cloudpickle
import sklearn
import mlflow
import mlflow.pyfunc

# Save the MLflow Model
features_and_label = training_df.columns
training_data = training_df.toPandas()[features_and_label]

X_train = training_data.drop(["purchased"], axis=1)
y_train = training_data.purchased.astype(int)
```

Figure 3: Training Data split

5. Instantiate Feature Computation Wrapper class and call its fit function by passing the training_data(labels and features as separate objects) from step4
6. Instantiate FeatureStoreClient() object and call log_model function on the same.

```
fs.log_model(
  pyfunc_model,
  artifact_path="model",
  flavor = mlflow.pyfunc,
  training_set=training_set,
  registered_model_name="destination_recommendations"
)
```

Figure 4: Register the model with featurestore client

For performing low-latency predictions, we need a high-performance, scalable solution for serving feature data to online applications and real-time machine learning models. For this kind of use case fast key-value stores like DynamoDB can be used as an online store[[2]]. Features can be published to online store from offline store by calling publish_table function on FeatureStoreClient object[[6]].

Challenge 3-Client Configuration & Data Freshness:
Challenge Description
Different clients or different use cases require different model versions. Maintaining separate versions allows for serving these diverse needs simultaneously. Different versions also allow for exact replication of past results, which is essential for debugging, validating findings, and ensuring consistency. Sometimes New models are not always perfect. If a newly deployed model performs poorly or introduces bugs, having previous stable versions readily available allows for quick rollbacks to minimize service disruption. These different model versions require different versions of data. It is essential that a particular model version should look up the exact data needed by that model version for inference. If the right version of data is not provided, then the features will not be the same on which model got trained and model will perform poorly during inference.

The statistical properties of the input features can change over time, leading to mismatch between the features on which the ML model trained on and the features it encounters during inference. This kind of mismatch results in poor predictions and bad decisions. Sometimes even the relationship between input features and desired output variables may change over time. This change degrades the model's predictive performance because the underlying patterns it was trained on are no longer valid.

Solution Description:
Databricks Feature Store serves different "versions" of features by leveraging Delta Lake's versioning for historical data retrieval and by automatically linking trained models to the specific feature data used during training.
When you train a model using features from the Databricks Feature Store, the model automatically tracks the lineage to the specific features and their values used during that training run. This metadata is stored with the registered MLflow model.
When features evolve or their definitions change, versioning allows for tracking these changes and ensuring that models are trained and served with the correct feature definitions.

II. Conclusion

Feature engineering defines the success of machine learning systems. The Databricks Feature Store revolutionizes this process by providing a scalable, collaborative, and governed approach to managing features. By integrating with MLflow, Delta Lake, and Unity catalog, it establishes a foundation for reproducible ML workflows that accelerate innovation.

References

[1]. Rayarao, S. R., & Donikena, N. Databricks Data Intelligence Platform: A Comprehensive Analysis of Machine Learning Capabilities and Data Management Features.
[2]. Modak, Rahul and Avula, Venu Gopal, Efficient Feature Store Architectures for Real-time Machine Learning Model Deployment in High-Throughput Systems (August 08, 2020).
[3]. Lamer, A., Saint-Dizier, C., Paris, N., & Chazard, E. (2024). Data lake, data warehouse, datamart, and feature store: Their contributions to the complete data reuse pipeline. *JMIR medical informatics*, *12*, e54590.
[4]. Ozdemir, S., & Susarla, D. (2018). *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd.
[5]. Databricks Feature Store | Databricks on AWS
[6]. Databricks Online Feature Stores | Databricks on AWS