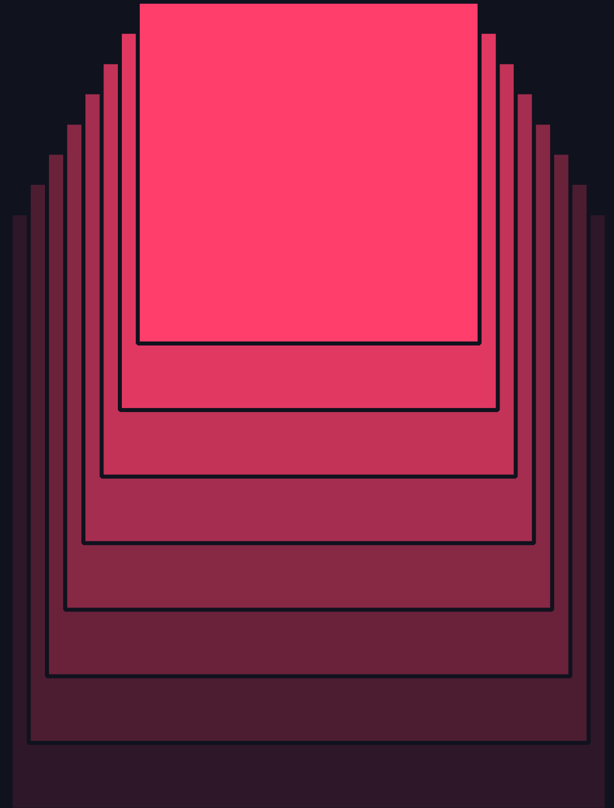
DATA+AI SUMMIT
BY databricks

Many Model Forecasting in Real-Time



Anastasia Prokaieva
13 May 2024

©2024 Databricks Inc. — All rights reserved

Meet your Speaker

Let's connect!

Anastasia Prokaieva

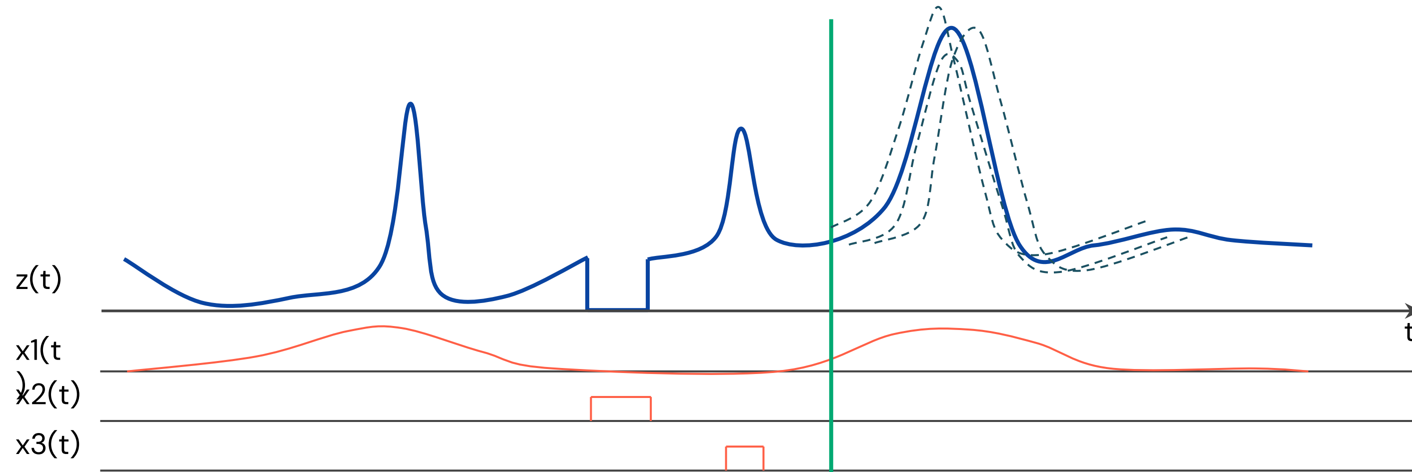
- Specialist Architect - AI and GeoSpatial
- Databricks since 2021, Global SME on AI and product champion on Model Serving
- Background in Physics & Applied Mathematics
- Book co-Author
  - **"Databricks ML in Action"** by Packt

Anastasia Prokaieva
🚀 AI Geek | Book Author | Speaker 🚀

©2024 Databricks Inc. — All rights reserved

Problem Statement

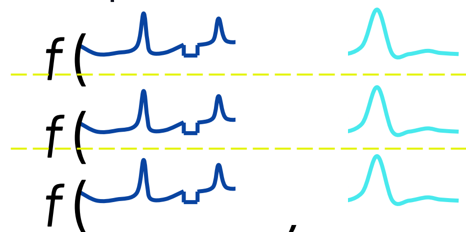
Time Series Forecasting



Types of Forecasting Algorithms

Local Models

Predicting individual time series separately. Each model is trained and applied to a specific time series, making it suitable for forecasting at a granular level, such as product-level sales forecasting in a large enterprise.



Takes only one time series at a time

Global Models

Consider multiple time series collectively. They forecast across a broader set of data. Global models are useful for capturing complex dependencies between different time series, making them valuable for broader, cross-entity forecasting tasks.

Learns parameters for multiple time series

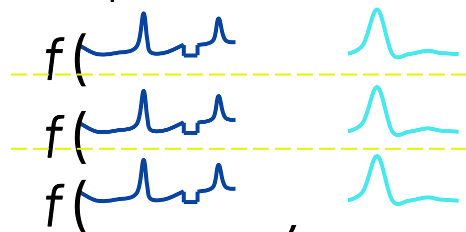
©2024 Databricks Inc. — All rights reserved

Types of Forecasting Algorithms

Local Models

Predicting individual time series separately. Each model is trained and applied to a specific time series, making it suitable for forecasting at a granular level, such as product-level sales forecasting in a large enterprise.

Our Focus today



Takes only one time series at a time

Global Models

Consider multiple time series collectively. They forecast across a broader set of data. Global models are useful for capturing complex dependencies between different time series, making them valuable for broader, cross-entity forecasting tasks.

Learns parameters for multiple time series

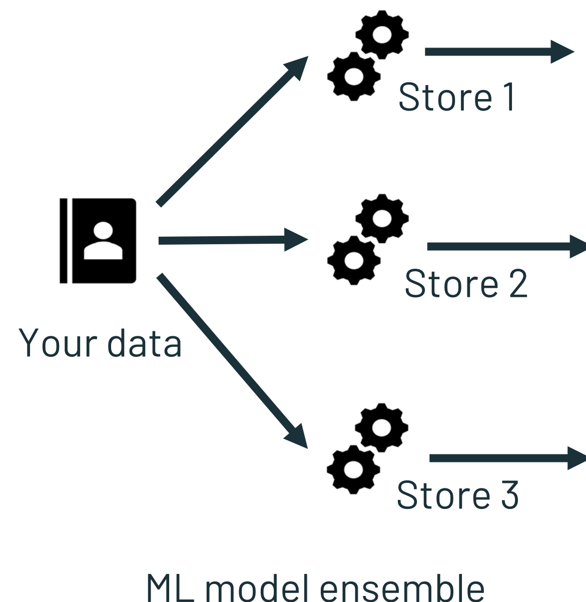
©2024 Databricks Inc. — All rights reserved

Our use case

Retailer that operates hundreds of thousands stores and want to bring operational forecasting of sales across all stores with all the available data in real-time taking into account metadata available.

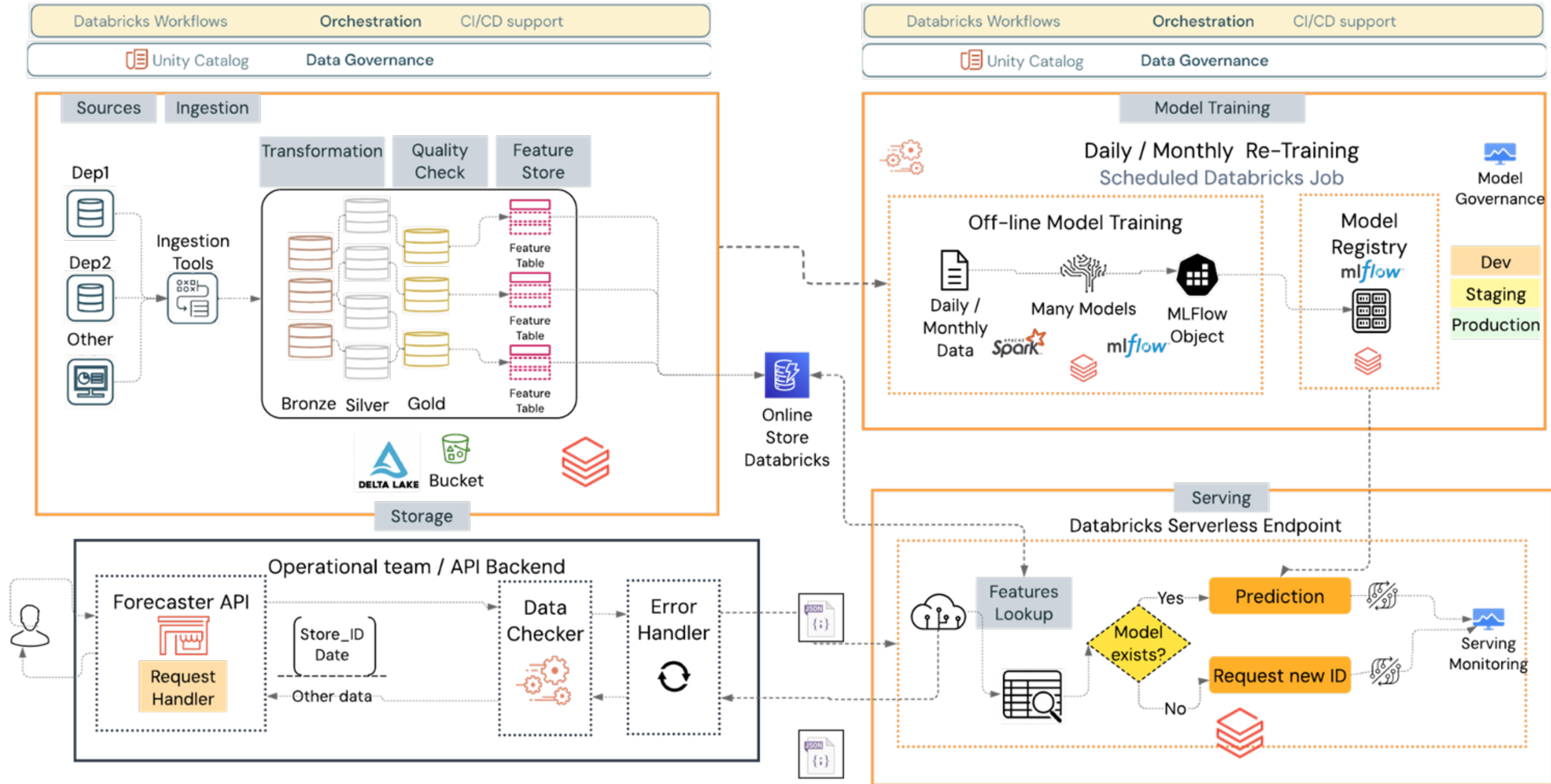
Key problems today:

- Training takes weeks
- Problems on joining freshly arriving features (weather, promos, marketing campaigns etc.)
- Data volumes are hard to maintain
- Requires to deliver updated forecasts per demand
- Would like to standardize on MLOps



©2024 Databricks Inc. — All rights reserved

Your Final Architecture(one of many)

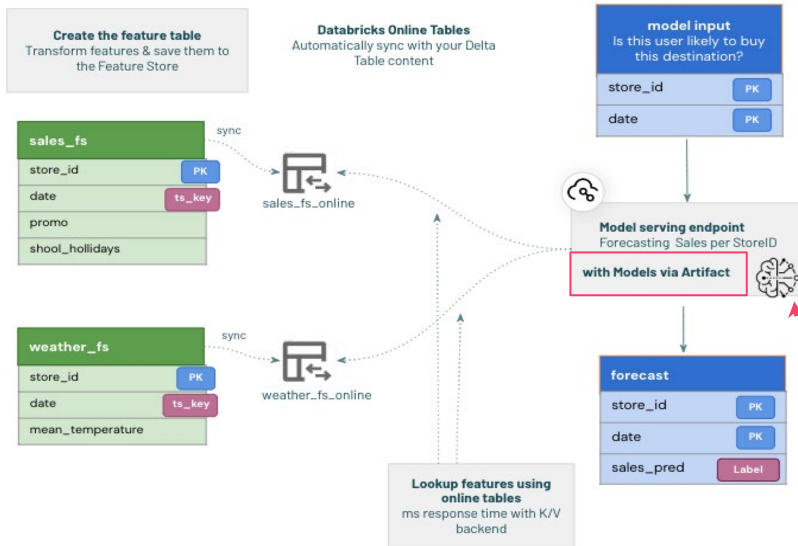


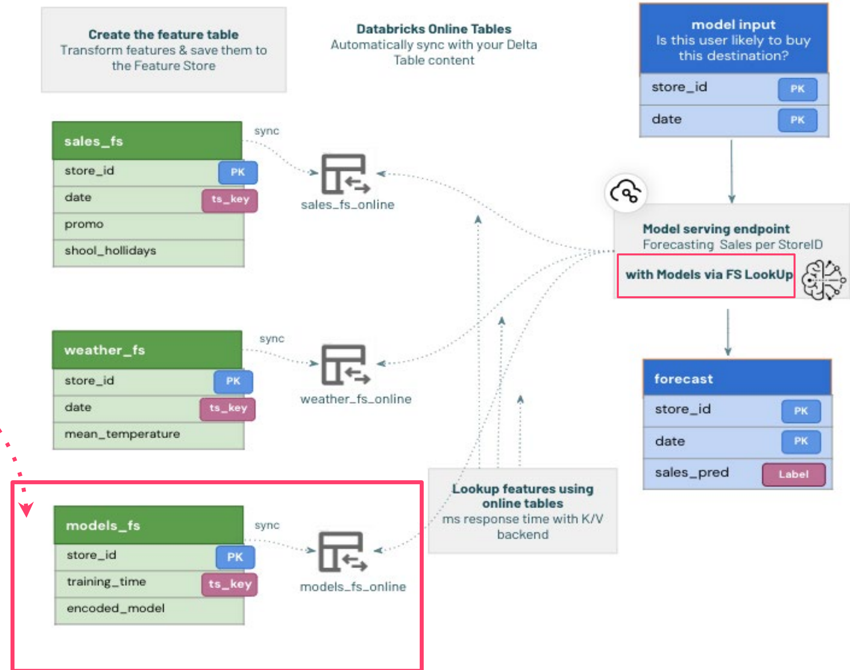
©2024 Databricks Inc. — All rights reserved

Your Final Architecture FS (zoom in)

A. Model Serving with Online store for Feature LookUp & all models are inside a container

B. Model Serving with Online store for Feature & MODELS LookUp





C. Model Serving with Online store for Feature LookUp & all models are inside Online Store

©2024 Databricks Inc. — All rights reserved

Part 1.a Creating a FS Training Dataset

```sql
ALTER TABLE ${catalog}.${schema}.sales_model_table_v2 ALTER COLUMN Store SET NOT NULL;
ALTER TABLE ${catalog}.${schema}.sales_model_table_v2 ADD CONSTRAINT sales_model_table_v2_pk PRIMARY KEY(Store);
```

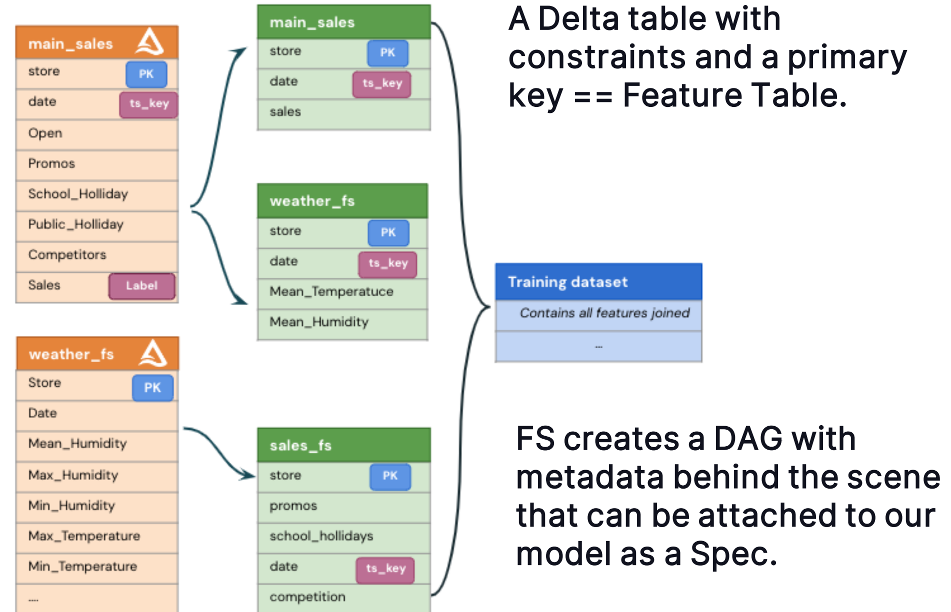
```python
fe = FeatureEngineeringClient()

training_df = (spark
              .table(f"{database_name}.main_sales_fs")
              .select("Store","Date","Sales"))

feature_lookups = [
  FeatureLookup(
      table_name=f"{database_name}.main_features_sales_fs",
      lookup_key=["Store","Date"],
      feature_names=["SchoolHoliday", "Promo"]
  ),
  FeatureLookup(
      table_name=f"{database_name}.extra_sales_weather_fs",
      lookup_key=["Store","Date"],
      feature_names=["Mean_TemperatureC"]
  ),
]

training_set = fe.create_training_set(
    df=training_df,
    feature_lookups=feature_lookups,
    label='Sales',
    timestamp_key = "Date"
)

training_df = training_set.load_df()
```

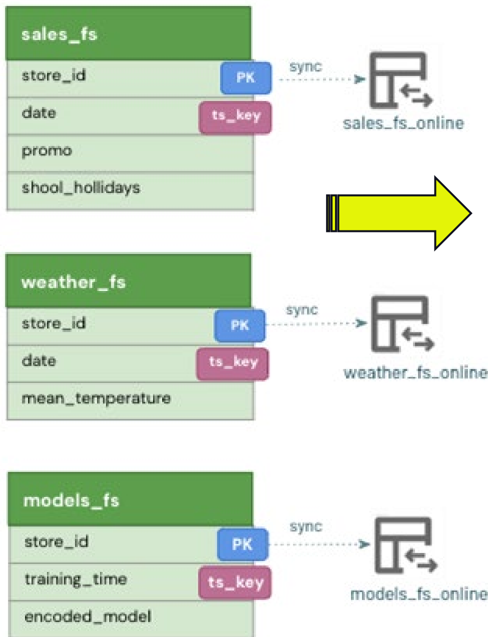


A Delta table with constraints and a primary key == Feature Table.

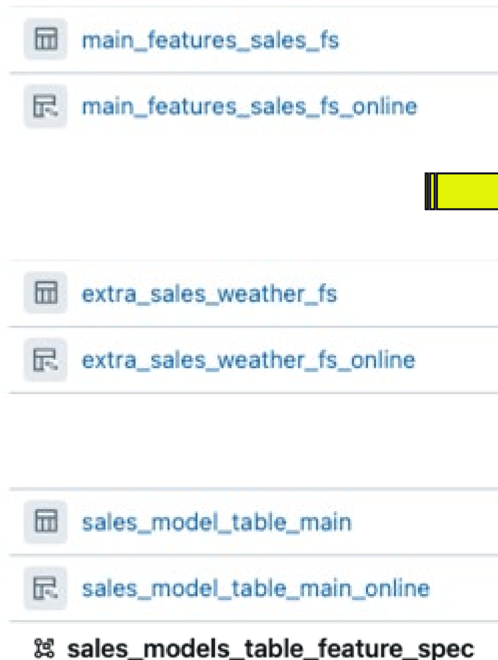
FS creates a DAG with metadata behind the scene that can be attached to our model as a Spec.

Part 1.b Publish Features to Online Store

Your Delta Tables with PKs and Change Data Feed enabled



Your Delta Tables with PKs published to Online Store that syncs to tables



You can serve your features externally with low - latency - Feature Serving via published Spec

Name	State	Compute
sales_models_table_feature_spec	Ready	**CPU**, **Small** 0-4 concurrency (0-4 DBU)

Query endpoint ×

Browser Curl Python SQL

Request

{"dataframe_split": {"index": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], "columns": ["Store", "Date"], "data": [["1", "2015-01-30"], ["1", "2015-01-29"], ["1", "2015-01-28"], ["1", "2015-01-27"], ["1", "2015-01-26"], ["1", "2015-01-25"], ["1", "2015-01-24"], ["1", "2015-01-23"], ["1", "2015-01-22"], ["1", "2015-01-21"], ["1", "2015-01-20"], ["1", "2015-01-19"], ["1", "2015-01-18"], ["1", "2015-01-17"], ["1", "2015-01-16"], ["1", "2015-01-15"], ["1", "2015-01-14"], ["1", "2015-01-13"], ["1", "2015-01-12"], ["1", "2015-01-11"], ["1", "2015-01-10"], ["1", "2015-01-09"], ["1", "2015-01-08"], ["1", "2015-01-07"], ["1", "2015-01-06"], ["1", "2015-01-05"], ["1", "2015-01-04"], ["1", "2015-01-03"], ["1", "2015-01-02"], ["1", "2015-01-01"]]}}

Response
from **sales_models_table_feature_spec**

{
  "outputs": [
    {
      "Store": "1",
      "encoded_model": [

"gASVmy0BAAAAAACMDXdyYXBwZXJfbW9kZWyUjBdGb3JlY2F
zdGluZ01vZGVsUHJvcGhldJSTlCmBlH2UKIwNbWFpbl9mZWF
0dXJlc5RdlCiMBVN0b3JllIwERGF0ZZSMBVNhbGVzlGWMCGZ
lYXR1cmVzlF2UKIwNU2Nob29sSG9saWRheZSMBVByb21vlIw
RTWVhbl9UZW1wZXJhdHVyZUOUZYwHaG9yaXpvbpRLHowFbW9
kZWyUjBJwcm9waGV0LmZvcmVjYXN0ZXKUjAdQcm9waGV0lJO
UKYGUfZQojAZncm93dGiUjAZsaW5lYXKUjAxjaGFuZ2Vwb21
udHOUjBJwYW5kYXMuY29yZS5zZXJpZXOUjAZTZXJpZXOUk5Q
pgZR9lCiMBF9tZ3KUjB5wYW5kYXMuY29yZS5pbnRlcm5hbHM
ubWFuYWdlcnOUjBJTaW5nbGVCbG9ja01hbmFnZXKUk5QpgZQ
oXZSMGHBhbmRhcy5jb3JlLmluZGV4ZXMuYmFzZZSMCl9uZXd
fSW5kZXiUk5SMG3BhbmRhcy5jb3JlLmluZGV4ZXMubnVtZXJ
pY5SMCkludDY0SW5kZXiUk5R9lCiMBGRhdGGUjBVudW1weS5

Send request Reset example

©2024 Databricks Inc. — All rights reserved

Part 2.a Training our models on scale

1) Make sure to return the same type as the provided schema - otherwise will cause a type problem.

2) **applyInPandas** will apply your function to the grouped data, the function gets a pdDF as input.

```
trainReturnSchema = t.StructType([
    t.StructField("Store", t.StringType()),
    t.StructField("model_path", t.StringType()),
    t.StructField("encoded_model", t.ArrayType(t.StringType())),])

with mlflow.start_run() as run:
    modelDirectoriesDF = (
        df_stores
        .withColumn("run_id", F.lit(run.info.run_id))
        .withColumn("experiment_id", F.lit(experiment_id))
        .groupby("Store")
        .applyInPandas(fit_final_model_udf, schema=trainReturnSchema))
```

```
def fit_final_model_udf(df_pandas: pd.DataFrame) -> pd.DataFrame:

    import prophet as Prophet
    model = ForecastingModelProphet()

    X = df_pandas[["Store", "Date", "Sales", "SchoolHoliday", "Promo", "Mean_TemperatureC"]]
    y = df_pandas.loc[:, ["Sales"]]
    ...
    # Optional!
    with mlflow.start_run(run_id=run_id, experiment_id=experiment_id) as outer_run:
        with mlflow.start_run(run_name=f"store_{store}", nested=True, experiment_id=experiment_id ) as run:
            model.fit(X)
            mlflow.pyfunc.log_model(artifact_path=artifact_name,python_model=model,)

    model_encoder = str(urlsafe_b64encode(pickle.dumps(model)).decode("utf-8"))
    return pd.DataFrame([[store, artifact_uri, [model_encoder]]], columns = ["Store", "model_path", "encoded_model"],)
```

1) Make sure to pass a class otherwise Spark does not serialise this properly.

2) Log your models, parameters, errors into MLFlow with a nested run.

3) We serialise our object into a str and return array of strings!

Part 3.a Wrap your model with Artifact

	Store	encoded_model	training_date
1	1024	gASVmy0BAAAAAACMDXdyYXBwZXJfbW9kZWyUjBdGb3JlY2FzdGluZ01vZGVsUHJvcGhldJSTlCmBlH2UKIwNbWFpbl9mZW...	2024-05-27
2	179	gASVmy0BAAAAAACMDXdyYXBwZXJfbW9kZWyUjBdGb3JlY2FzdGluZ01vZGVsUHJvcGhldJSTlCmBlH2UKIwNbWFpbl9mZW...	2024-05-27
3	409	gASVmy0BAAAAAACMDXdyYXBwZXJfbW9kZWyUjBdGb3JlY2FzdGluZ01vZGVsUHJvcGhldJSTlCmBlH2UKIwNbWFpbl9mZW...	2024-05-27

```
class MultiModelPyfunc(mlflow.pyfunc.PythonModel):
    def __init__(self, model_list = []):
        self.model_list = model_list

    def load_context(self, context):
        model_list = pd.DataFrame.from_records(
            mlflow.artifacts.load_dict(context.artifacts['model_list'])
        )
        model_list["model_artifact"] = [pickle.loads(urlsafe_b64decode(artifact.encode("utf-8")))
                                        for artifact in model_list['encoded_model']
                                        ]
        self.model_list = model_list.set_index('Store')

    def predict(self, context, model_input):…
```

```
fe.log_model(
    artifact_path = "model",
    model = MultiModelPyfunc(),
    flavor= mlflow.pyfunc,
    pip_requirements= reqs,
    artifacts= artifacts,
    training_set=training_set,
    registered_model_name=model_name,
    code_path = ['wrapper_model.py']
)
```


©2024 Databricks Inc. — All rights reserved

Part 3.b Wrap your model with Online Store

```
models_lookups = [FeatureLookup(
        table_name=f"{database_name}.sales_model_table_main",
        lookup_key=["Store"],
        feature_names=["encoded_model"]
    ),
]

features_set = fe.create_training_set(
    df=label_df,
    feature_lookups=feature_lookups+models_lookups,
    label='Sales',
)
```

```
fe.log_model(
    artifact_path = "model",
    model = MultiModelPyfunc(),
    flavor= mlflow.pyfunc,
    pip_requirements= reqs,
    training_set=features_set,
    registered_model_name=model_name,
    code_path = ['wrapper_model.py']
)
```

DAG behind the scene is attached to the metadata of FS. When you evoke the model on batch/serving the features will be “looked” and joined to the dataset on PK.

```
class MultiModelPyfunc(mlflow.pyfunc.PythonModel):
    def __init__(self):
        super().__init__()

    def predict(self, context, model_input):
        output_df = pd.DataFrame()
        for store_id, pd_store_df in model_input.groupby('Store'):
            model_pickled = pd_store_df.iloc[0,:]["encoded_model"][0]
            model = pickle.loads(urlsafe_b64decode(model_pickled.encode("utf-8")))
            predict_df = model.predict(context=None,
                                       data_input = pd_store_df[[
                                        "Store", "Date", "SchoolHoliday", "Promo", "Mean_TemperatureC"
                                        ]])
            output_df = pd.concat([output_df, predict_df], ignore_index=True)
        return output_df
```


©2024 Databricks Inc. — All rights reserved

Part 4 Serving our models on scale

- ap
  - default
  - forecast
    - Tables (10)
    - Volumes (1)
      - dais_ts
    - Functions (1)
      - sales_models_table_feature_spec
    - Models (7)
      - model_wrapper_serving
      - model_wrapper_serving_fsa
      - model_wrapper_serving_fsm
      - model_wrapper_serving_mt

Catalogs › ap › forecast ›

model_wrapper_serving_fsm

Serve this model

Overview Details Permissions

Description: Add description

Versions

Status	Version	Time registered	Tags	Aliases	Registered by	Comment
	Version 3	2024-06-05 14:56:22			anastasia.prokaieva@dat...	
	Version 2	2024-06-05 09:23:52			anastasia.prokaieva@dat...	
	Version 1	2024-06-04 20:14:36			anastasia.prokaieva@dat...	

Query endpoint

Browser Curl Python SQL

Request

{"dataframe_split": {"index": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], "columns": ["Store", "Date"], "data": [["1", "2015-01-30"], ["1", "2015-01-29"], ["1", "2015-01-28"], ["1", "2015-01-27"], ["1", "2015-01-26"], ["1", "2015-01-25"], ["1", "2015-01-24"], ["1", "2015-01-23"], ["1", "2015-01-22"], ["1", "2015-01-21"], ["1", "2015-01-20"], ["1", "2015-01-19"], ["1", "2015-01-18"], ["1", "2015-01-17"], ["1", "2015-01-16"], ["1", "2015-01-15"], ["1", "2015-01-14"], ["1", "2015-01-13"], ["1", "2015-01-12"], ["1", "2015-01-11"], ["1", "2015-01-10"], ["1", "2015-01-09"], ["1", "2015-01-08"], ["1", "2015-01-07"], ["1", "2015-01-06"], ["1", "2015-01-05"], ["1", "2015-01-04"], ["1", "2015-01-03"], ["1", "2015-01-02"], ["1", "2015-01-01"]]}}

Response
from **model_wrapper_serving_fsa-6**

{
  "predictions": [
    {
      "Date": "2015-01-01T00:00:00",
      "Store": "1",
      "Sales_Pred": 3738.813755710591
    },
    {
      "Date": "2015-01-02T00:00:00",
      "Store": "1",
      "Sales_Pred": 4142.608373035108
    },
    {
      "Date": "2015-01-03T00:00:00",
      "Store": "1",
      "Sales_Pred": 5209.715878546838
    },
    {
      "Date": "2015-01-04T00:00:00",

Send request

- all 3 models can be queried using same schema
- you can pass data, and it will be replaced

©2024 Databricks Inc. — All rights reserved

Conclusions

What have we learned by doing?

- Feature Engineering Client and Online Tables from Databricks combined with Model Serving significantly simplifies features lookups and joints with a TimeStamp dependency on features updates.
- We can store various type of data under Online Tables, e.g serialized models for real-time calls.
- Feature Engineering Client and Online tables can be used across any project like Forecasting, Recommender Systems, GenAI Agents etc

©2024 Databricks Inc. — All rights reserved

Warnings/Limitations

Few tricks and tips to make it successful

Fixed Container Memory

- Limitation:
  - 4 Gb RAM for a CPU container
- Solution:
  - Will be lifted, if needed contact your Dbx team
  - Use small GPU container
  - Move into pure Online store solution

Online Store Str size

- Limitation:
  - 65Kb of a string type per row
- Solution
  - Publish your serialized model as array(string)
  - Use smaller models
  - Compress your model

MLOps

- Limitation:
  - To update models under an artifact have to redeploy a model container
- Solution:
  - Use pure Online Store solution with a TimeStamp Key on model updates

©2024 Databricks Inc. — All rights reserved