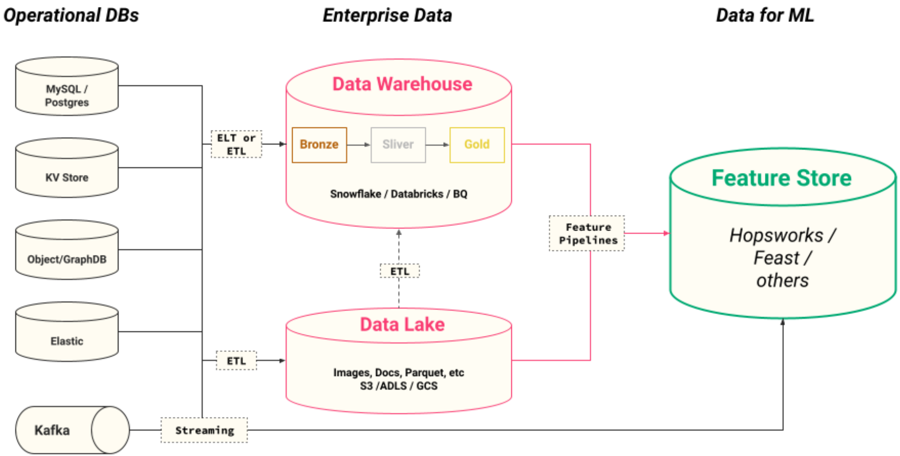
Feature Stores for Machine Learning

Jim Dowling
jdowling@kth.se

Enterprise Data and Feature Store



Example e-commerce marketing data model

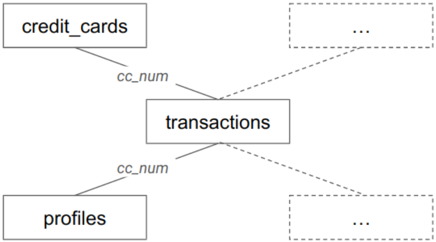
- **Facts**
  - Impressions
  - Clicks
  - Email sends
  - Email opens
  - Website Visits
  - Website Visitors
    - Cost
    - Add-to-Carts
    - Conversions
    - Revenue
    - Profit
- **Dimensions**
  - Campaign
  - Channel
  - Product Family
  - Product
  - User Profile
    - Opt Out
    - GDPR
    - Location
    - Persona

Data modelling: Fact and Dimension Tables

- A popular Data Model for Data Warehouses is to have Fact and Dimension Tables
- Examples of Facts: purchases, user clicks, user searches, songs played, embeddings (recent user searches/sessions)
- Examples of Dimensions: click dimension, location dimension, time dimension, customer dimension, song dimension
- Business events are modelled as Facts (aka measurements)
- Idenify and save dimensions for your facts that are useful for analysis or prediction services
- Dimensions can be thought of as the columns you would expect to "group by"

Example credit-card fraud facts and dimensions in a Star Schema

- **Fact table**
    - transactions
- **Dimension tables**
    - profiles
    - credit_cards
- **This data modeling approach is known as building a Star Schema**
    - Easy to add new Dimension tables
    - A Snowflake schema just hangs more dimensions off the Dimension tables



Example credit-card fraud tables

Data for a prediction service to identify if a credit card transaction is **suspected of fraud or not.**

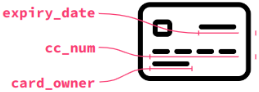
transactions table

tid	datetime	cc_num	category	amount	long/lat
1	2022-09-01 11:24	1111 …	food	45.33	53N,6W
2	2022-09-01 13:24	1111 …	clothing	183.12	52N,6W

user_profile table

Name
Sex
Date of Birth
City

credit_card_details table



Our credit-card fraud tables are updated at different cadences

Updated once/hour

transactions

tid	cc_num	datetime	category	amount	long/lat	fraud
1	11111	2022-09-01 11:24	food	45.33	53N,6W	No
2	11111	2022-09-02 09:17	clothing	183.12	52N,6W	No
3	11111	2022-09-04 19:33	entertain	63.33	51N,7W	yes
..	…	…	…	…	…	…

+1w

Updated once/week

credit_cards

cc_num	provider	expires
11111	visa	24/05
…	…	…

+1d

Updated once/day

profiles

cc_num	name	sex	DoB	city
11111	Jim D	M	26/09/74	Dublin
…	…	…	…	…

Our credit-card fraud tables are in 3rd normal form

transactions

tid	cc_num	datetime	category	amount	long/lat	fraud
1	11111	2022-09-01 11:24	food	45.33	53N,6W	No
2	11111	2022-09-02 09:17	clothing	183.12	52N,6W	No
3	11111	2022-09-04 19:33	entertain	63.33	51N,7W	yes
..	…	…	…	…	…	…

cc_num *Join Key*

credit_cards

cc_num	provider	expires
11111	visa	24/05
…	…	…

profiles

cc_num	name	sex	DoB	city
11111	Jim D	M	26/09/74	Dublin
…	…	…	…	…

Note: the Join Key is a part of the Primary Key of all our tables

The primary keys for our credit-card fraud tables

transactions (Composite Primary Key: tid, cc_num)

tid	cc_num	datetime	category	amount	long/lat	fraud
1	11111	2022-09-01 11:24	food	45.33	53N,6W	No
2	11111	2022-09-02 09:17	clothing	183.12	52N,6W	No
3	11111	2022-09-04 19:33	entertain	63.33	51N,7W	yes
..	…	…	…	…	…	…

credit_cards (Primary Key: cc_num)

cc_num	provider	expires
11111	visa	24/05
…	…	…

profiles (Primary Key: cc_num)

cc_num	name	sex	DoB	city
11111	Jim D	M	26/09/74	Dublin
…	…	…	…	…

Note: the Join Key is a part of the Primary Key of all our tables

Credit card number - the Join key for our credit-card fraud tables

Composite Primary Key

transactions

tid	cc_num	datetime	category	amount	long/lat	fraud
1	11111	2022-09-01 11:24	food	45.33	53N,6W	No
2	11111	2022-09-02 09:17	clothing	183.12	52N,6W	No
3	11111	2022-09-04 19:33	entertain	63.33	51N,7W	yes
..	…	…	…	…	…	…

cc_num ***Join Key***

Primary Key

credit_cards

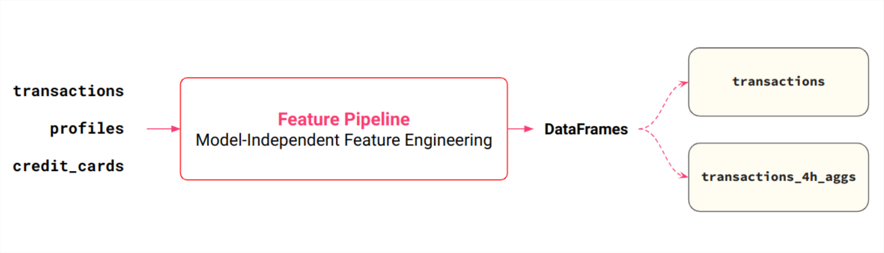
cc_num	provider	expires
11111	visa	24/05
…	…	…

Primary Key

profiles

cc_num	name	sex	DoB	city
11111	Jim D	M	26/09/74	Dublin
…	…	…	…	…

Feature Pipeline for Credit Card Fraud Features



Credit Card Transactions Feature Group

Key	Primary Key	Event Time	Features			Label	Features			
tid	cc_num	datetime	category	amount	long/lat	fraud	days_until_card_expires	age_at_transaction	sex	lives_city
1	1111 ...	2022-09-01 11:24	food	45.33	53N,6W	No	1011	47	M	Dublin
2	1111 ...	2022-09-02 09:17	clothing	183.12	52N,6W	No	1010	47	M	Dublin
3	1111 ...	2022-09-04 19:33	entertain	63.33	51N,7W	yes	1008	47	M	Dublin
..	...	...	...	...	...	...	...	...	...	...

Credit Card Transactions Feature Group - One Big Table

Our main Feature group is `transactions`. The columns for this one big table (OBT) have different data sources, that are updated at different cadences. It is inefficient to update all columns in every update, but less JOINs will be required for training data.

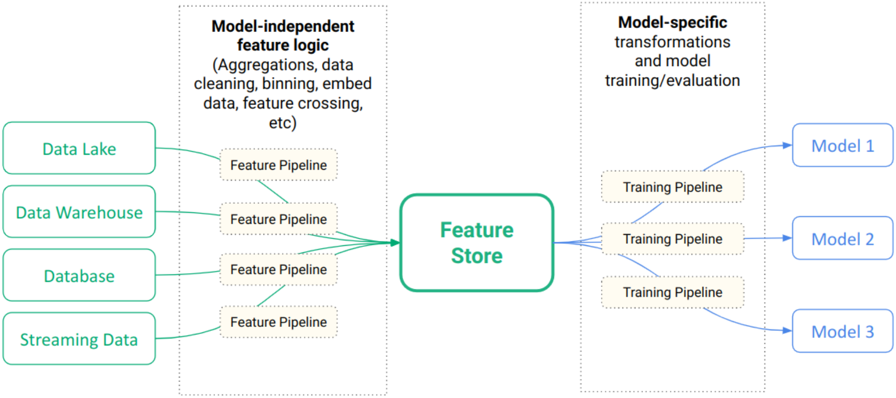
Updated Frequently							*Updated Less Frequently*			
Source: cc_transactions							*Source: credit_card*	*Source: profile*		
tid	cc_num	datetime	category	amount	long/lat	fraud	days_until_card_expires	age_at_transaction	sex	lives_city
1	1111 ...	2022-09-01 11:24	food	45.33	53N,6W	No	1011	47	M	Dublin
2	1111 ...	2022-09-02 09:17	clothing	183.12	52N,6W	No	1010	47	M	Dublin
3	1111 ...	2022-09-04 19:33	entertain	63.33	51N,7W	yes	1008	47	M	Dublin
..	...	...	...	...	...	...	...	...	...	...

Credit Card Transactions - 4 hr Aggregations Feature Group

`transactions_4h_aggs` contains aggregated features computed over a 4h time window for each credit card

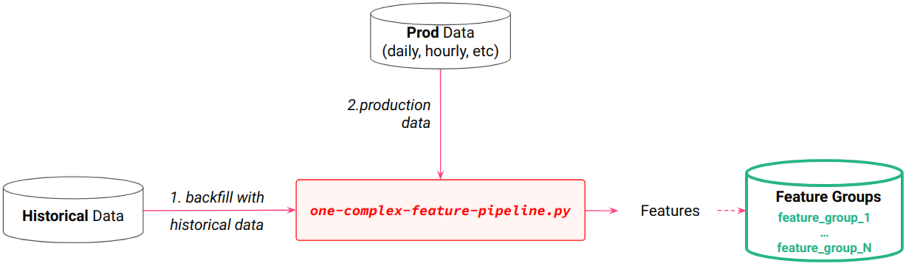
cc_num	datetime	loc_delta_mavg	trans_volume_mstd	trans_volume_mavg	trans_freq
Primary Key	Event Time	Features			
1111 ...	2022-09-01 11:24	53N,6W	3.4	8	6
1111 ...	2022-09-02 09:17	52N,6W	3.6	3	5
1111 ...	2022-09-04 19:33	51N,7W	4.1	33	45
...	...	...	...	...	...

Decouple feature pipelines from Models with a Feature Store

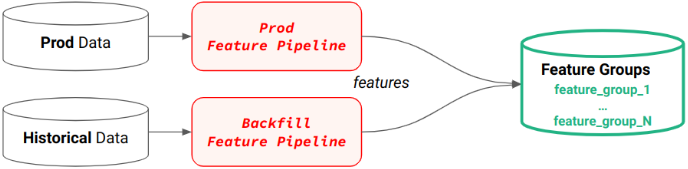


The number of models is independent of the number of feature pipelines - features can be reused in different models.

One complex feature pipeline with both backfill and production



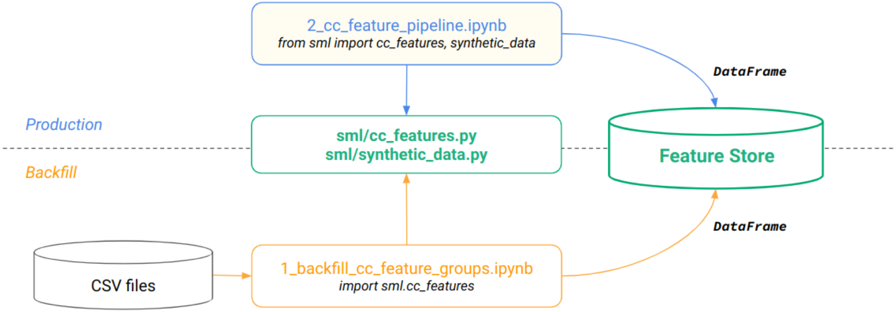
Separate feature pipeline for backfill and production



DRY code warning!
Do not re-implement (or copy!) the feature logic from you backfill feature pipeline to your production (prod) data feature pipeline, as there is a risk of them becoming inconsistent over time.

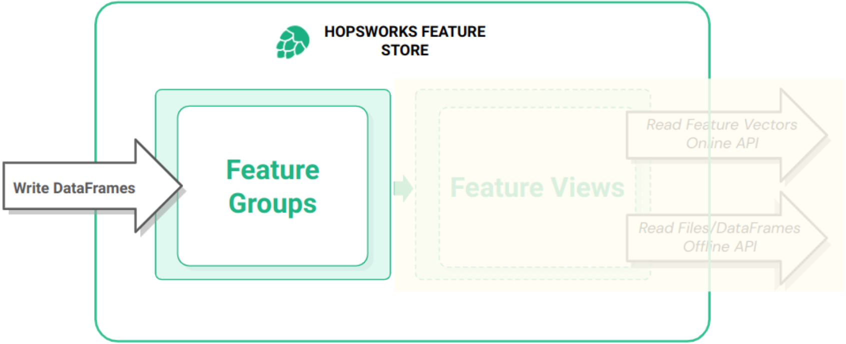
Separate feature pipeline for backfill and production with shared code

1. Move all feature engineering code to shared Python module(s)
2. Write features to the same feature groups from backfill and production feature pipelines



Feature pipelines write DataFrames to Feature Groups

- A Feature Group is a table that stores feature data and metadata in a Feature Store
- Feature pipelines use DataFrames to insert/update/delete rows in Feature Groups
- Feature Groups are versioned - breaking schema changes requires a new version



Create a Feature Group in Hopsworks with Python

```
fg = feature_store.create_feature_group(name="transactions",
    version=1,
    description="Credit Card Holder Details",
    online_enabled=True,
    primary_key=['cc_num'],
    partition_key=['city'],
    event_time='datetime',
    statistics_config={
        "enabled": True,
        "histograms": True,
        "correlations": True,
        "exact_uniqueness": False,
        "columns": [“amount”, “category”]
    }
)

fg.insert(df)      # The DataFrame provides the Schema
```

Feature Group - primary keys

- A Feature Group should define one or more columns as its **primary key**, such that every row in the table can be uniquely identified
- A primary key prevents duplicate data as each row is unique
- A primary key enables a row of features to be retrieved with the Online API

```
fg = feature_store.create_feature_group(name="transactions",
…
    primary_key=['cc_num'],
)
```

Feature Group - Event Time

Rows can be updated, but *event_time* columns enables a history of their values over time.

cc_num	datetime	sex	lives_city
1111 2222	1974-09-26 06:00	M	Dublin
1111 2222	2005-10-01 00:00	M	Stockholm
1111 2222	2023-01-10 10:00	F	Stockholm

We can now make **time-travel queries** about our credit card holder:

- Where did the cc holder live on 2000-01-01?
- What was the cc holder's gender on 2022-01-1?

```
fg = feature_store.create_feature_group(name="transactions",
…
     event_time='datetime'
)
```

Note: with time-travel, the primary key no longer uniquely identifies each row. Now, you need the combination of *(primary_key, event_time)*. For this reason, we often call the primary key the ***entity ID***.

Feature Group - Event Time is not Ingestion Time

Event time is not the same as ingestion time

cc_num	datetime	ingestion_time	sex	lives_city
1111 2222	1974-09-26 06:00	1994-10-10 11:15	M	Dublin
1111 2222	2005-10-01 10:00	2005-10-12 00:00	M	Stockholm
1111 2222	2023-01-10 10:00	2023-10-10 11:00	F	Stockholm

Move from Dublin to Stockholm
`2005-10-01 10:00 | event_time`

Update my details
(including `event_time` for moving from `Dublin` to `Stockholm`)

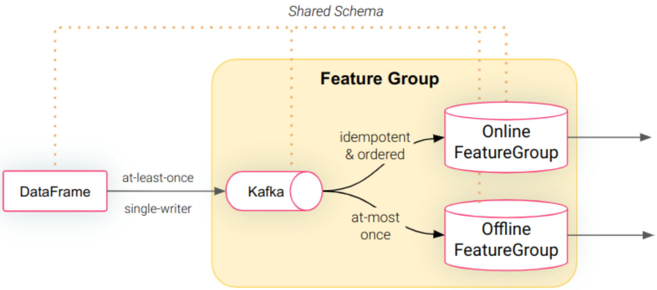
`2005-10-12 00:00`
`ingestion_time`

Feature Group - Online Enabled

```
fg = feature_store.create_feature_group(name="transactions",
…
    online_enabled=True,
)
```

Tip

If your features may be accessed using Online Applications, set ***online_enabled=True***



Online API
low-latency, row-oriented, no event_time, latest values

Offline API
high throughput, batch-oriented, event_time, historical values (time-travel) supported

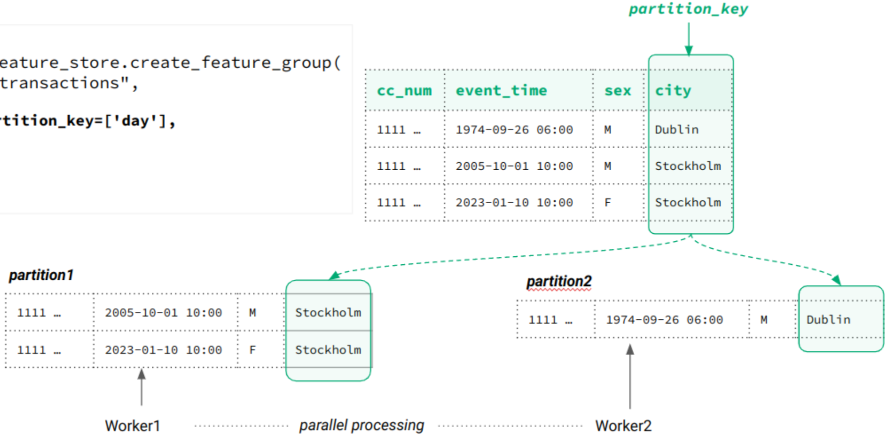
Feature Groups are stored internally with Hive (offline), MySQL (online) schemas

Pandas DType	Hive Type	MySQL Type
bool	BOOLEAN	TINYINT
int8	INT	TINYINT
uint8/16, int16/32	INT	INT
int, uint32, int64	BIGINT	BIGINT
float, float16, float32	FLOAT	FLOAT
float64	DOUBLE	DOUBLE
decimal.decimal	DECIMAL(PREC, SCALE)	DECIMAL(PREC, SCALE)
datetime64[ns]	TIMESTAMP	TIMESTAMP
object (datetime.date)	DATE	DATE
object (str), object(np.unicode)	STRING	VARCHAR(100)
object (list), object (np.ndarray)	ARRAY<TYPE>	VARBINARY(100)/BLOB
object (dict)	STRUCT<NAME: TYPE, ...>	VARBINARY(100)/BLOB
object (binary)	BINARY	VARBINARY(100)/BLOB
-	MAP<String,TYPE>	VARBINARY(100)/BLOB

Source: https://docs.hopsworks.ai/3.0/user_guides/fs/feature_group/data_types/

Partitions: Efficient Queries over Offline Feature Groups storing large amounts of data

```
fg = feature_store.create_feature_group(
name="transactions",
…
    partition_key=['day'],
)
```

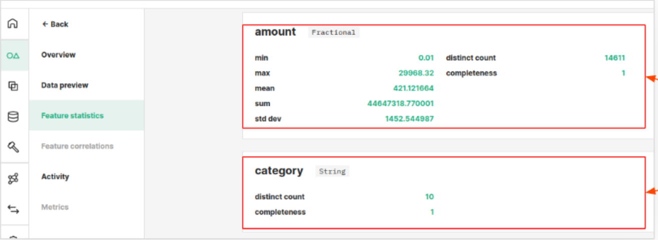


Tip

Ensure the size of the partitions is balanced or else some workers will do all the work, reducing performance.

Compute descriptive statistics over numerical features, distributions for categorical features

```
fg = feature_store.create_feature_group(name="transactions",
...

    statistics_config={
        "enabled": True,
        "histograms": True,
        "correlations": True,
        "exact_uniqueness": False,
        "columns": [“amount”, “category”]
    }
)
```



statistics computed over the ‘amount’ and ‘category’ features

Storing Labels in Feature Groups

- A Feature Group that contains labels looks like any other feature group
  - The label column is a column like any other column
- A "Label Feature Group" typically contains an **event_time** column, indicating when the label value was observed, and it is typically not *onlined_enabled.* Labels are defined in *Feature Views*.

```
fg = feature_store.create_feature_group(name="transactions",
    version=1,
    description="Credit Card Fraud Labels",
    primary_key=['tid', ‘cc_num’],
    event_time='datetime',
)
```

tid	cc_num	datetime	is_fraud
12345	1111 2222 …	1974-09-26 06:00	False
12346	1111 2222 …	2005-10-01 00:00	False
12347	1111 2222 …	2023-01-10 10:00	True

Feature Selection

month	inflation_rate	income_growth

day	electricity_price

user_id	event_time	income	age

day	weather

- Identify (1) **features with predictive power** for your prediction problem and (2) **the JOIN keys**
- Avoid **Feature Debt** – features once added to a model are rarely removed and tend to accumulate
- Feature selection is as either **part of a training pipeline** or as offline **experimentation**

Which features from which Feature Groups have predictive power for my prediction problem?

Feature Selection with Scikit-Learn

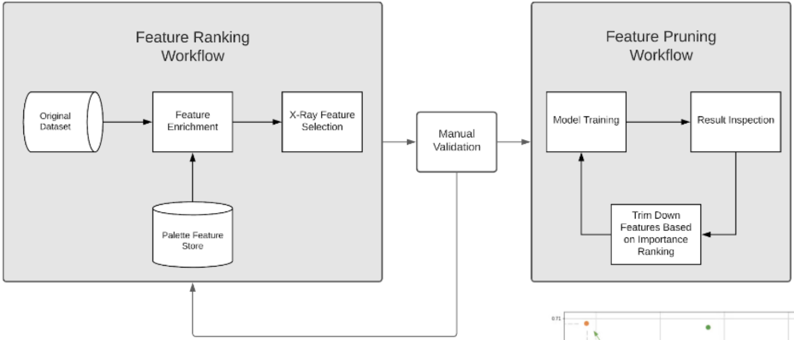
- Remove features with low variance
- Recursive feature elimination
- Feature selection using SelectFromModel
- Sequential Feature Selection

https://scikit-learn.org/stable/modules/feature_selection.html#

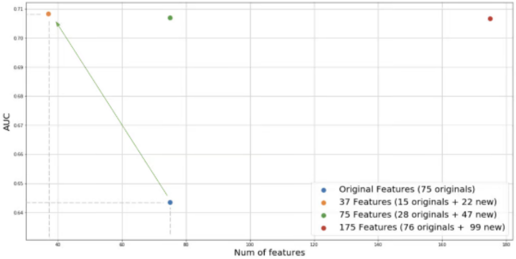
Select the best features based on univariate statistical tests

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X, y = load_iris(return_X_y=True)
X.shape
(150, 4)
X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
X_new.shape
(150, 2)

#Which 2 features were selected for the Iris Dataset?
```

Feature Selection with Uber's XRay Framework

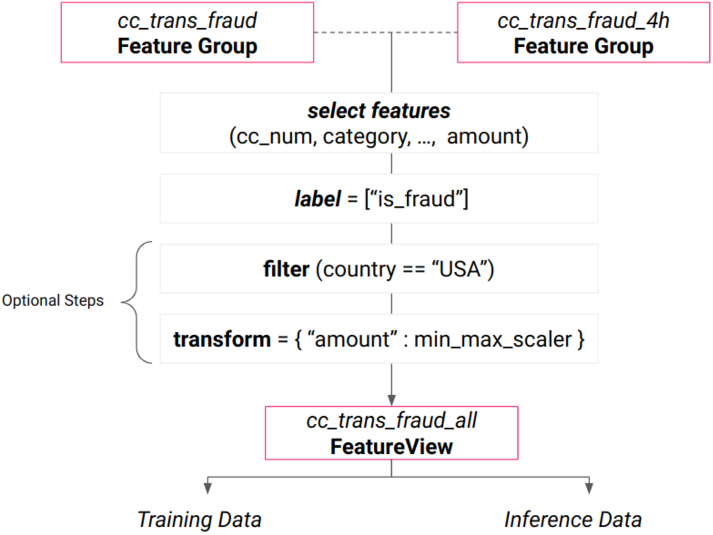


Original baseline dataset with 75 features. X-Ray evaluated 2k features from the feature store, and the best dataset had 15 features from the original dataset and 27 features from the feature store.



[Images from Uber: https://www.uber.com/en-EC/blog/optimal-feature-discovery-ml/]

Feature Selection with a Feature View



Join Features together to create a Feature View

cc_trans_fraud

cc_num	datetime	amount	category	…
1111 2222 …	2004-01-01 10:00	…	…	…
1111 2222 …	2004-01-02 11:00	…	…	…
1111 2222 …	2004-01-03 12:00	…	…	…

cc_trans_fraud_4h

cc_num	datetime	loc_delta_mavg	trans_freq
1111 2222 …	2004-01-01 00:00	…	…
1111 2222 …	2004-01-02 06:00	…	…
1111 2222 …	2004-01-03 12:00	…	…

Join on cc_num

Feature View (cc_trans_fraud_all)

	amount	category	loc_delta_mavg	trans_freq	….
Datatype	float	string	float	float	
Transformation Function	<min_max_scalar>	<none>	<none>	<min_max_scalar>	

Point-in-Time Correct Joins needed to create Training Data

cc_trans_fraud

cc_num	datetime	amount	category	...
1111 2222 ...	2004-01-01 10:00	...	...	...
1111 2222 ...	2004-01-02 11:00	...	...	...
1111 2222 ...	2004-01-03 12:00	...	...	...

cc_trans_fraud_4h

cc_num	datetime	loc_delta_mavg	trans_freq
1111 2222 ...	2004-01-01 00:00	...	...
1111 2222 ...	2004-01-02 06:00	...	...
1111 2222 ...	2004-01-03 12:00	...	...

Point-in-time (PiT) Correct JOIN
(no data leakage)

cc_trans_fraud_all

datetime	amount	category	loc_delta_mavg	trans_freq	...
2004-01-01 10:00	...	...	...	...	...
2004-01-02 11:00	...	...	...	...	...
2004-01-03 12:00	...	...	...	...	...

Training Data

Create a Feature View

```
fg_trans = fs.get_feature_group("cc_trans_fraud", version=1)
fg_trans_4h = fs.get_feature_group("cc_trans_fraud_4h", version=1)
labels = fs.get_feature_group("labels", version=1)

query = labels.select_all().join(fg_trans.select_all() \
        .join(fg_trans_4h.select_all()))
```

DSL to join features, returns a Query object

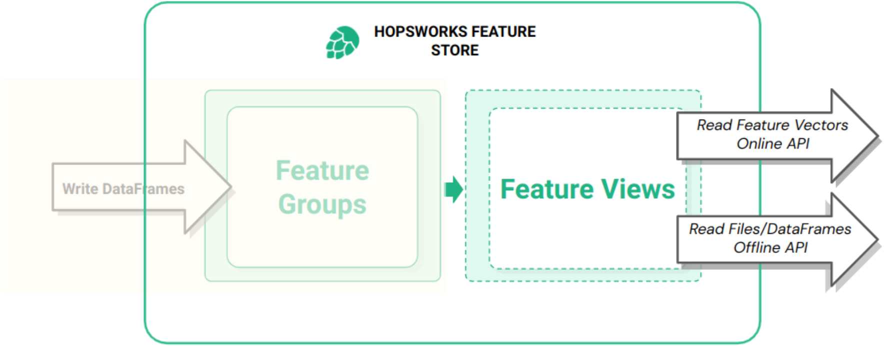
```
fv = fs.create_feature_view(name="cc_trans_fraud_all",
    version=1,
    description="Credit Card Transactions",
    label=['is_fraud],
    query=query
)
```

Both label and query object needed to create a Feature View

https://docs.hopsworks.ai/3.0/user_guides/fs/feature_view/overview/

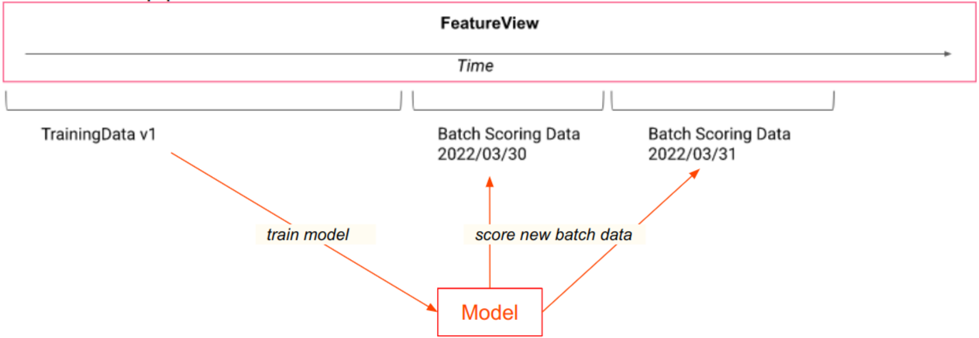
Create a Feature View from your Selected Features

- A Feature View contains a model's input features (for training and inference)
- A Feature View is metadata - the actual feature data is stored in Feature Groups
- The Feature View provides both an Offline and an Online API
    - The Offline API is a batch API for reading historical feature data
    - The Online API is a row-oriented API for reading feature vectors using a primary key



Feature View Offline API: Create Training Data or Batch Inference Data

- Create Training Data for Models
- Create Batch Inference (Scoring) Data for new data that arrives in Feature Groups via feature pipelines



Feature View Offline API: Create Training Data

- Create Training Data for Models as
  (1) Pandas DataFames or
  (2) Files
- You can also create train/validation/test splits (random or temporal)
- For (2) files, you can specify the output file format and where the files should be stored.

```
# (1) Pandas DataFrames
feature_df, label_df = feature_view.training_data(
    description = 'transactions fraud batch training dataset',
)

# (2) Files
version, job = feature_view.create_training_data(
    description = 'transactions_dataset_jan_feb',
    data_format = 'csv',
    write_options = {"wait_for_job": False}
)
```

Create training data as Pandas DataFrames

Create training data as files in 'csv' file format

Random or Time-Series Split into Train/Test sets?

- In the Iris lab, we performed a **random split** on the training data into *train* and *test sets*
- For time-series data, like credit-card data, it is better to do a **temporal split** on the training data
  - E.g., the *train set* is for the years 2015-2021, *test set* is for data from the year 2022

```
td_version, td_job = feature_view.create_train_test_split(
    train_start = "2015-01-01 00:00",
    train_end = "2021-12-31 23:59",
    test_start = "2022-01-01 00:00",
    test_end = "2022-10-11 00:00",
    data_format = 'csv',
    write_options = {'wait_for_job': True},
    coalesce = True
)
X_train, X_test, y_train, y_test =
feature_view.get_train_test_split(td_version)
```

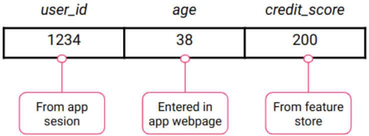
Start/end timestamps for the train/test sets

Not efficient! Write as a single CSV file

Read time-series splits of TD as DataFrames

Feature View Online API: Retrieve Feature Vectors for Online Models

- Retrieve a row containing features using the feature view and the primary key(s).
- Optionally specify `passed_features` that are features that come from the application, not from the feature store.



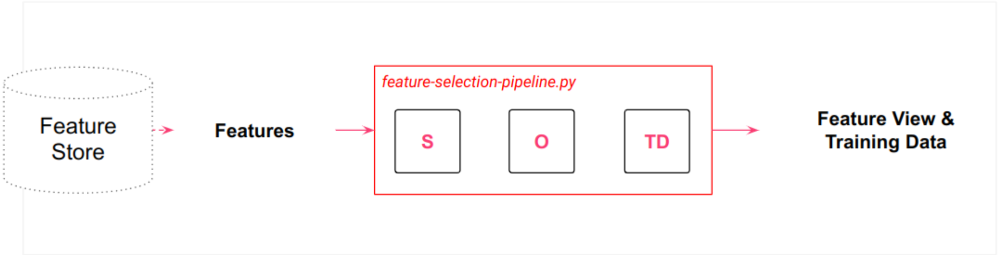

```
# training_data_version number is required if there are featurestore
# transformations - they are computed using stats from training data
feature_view.init(training_data_version=1)

keys = { “cc_num” : “1111 2222 3333 4444” }

array_features = feature_view.get_feature_vector(entry=keys,
    passed_features = {"feature_a": "value_a"}
)
model.predict(array_features)
```

Init the FV, so that transformation use correct TD version state

Retrieve feature vector for scoring from feature store, with ‘feature_a’ as a supplied value

Feature Selection Pipeline

If you want to automate feature selection, you should build a **feature selection pipeline** that takes as input a set of candidate features, a feature selection algorithm, an optional specification for **training data** (file format, splits) and writes as output a **Feature View** and Training Data.



SO = Select features, Optimize Features, Training Data.

Model-Specific Transformations can be applied by Feature Views (1/3)

- Transformation functions are applied to features to (1) make their data compatible with the model training algorithm or (2) to improve model performance
- Transformation functions typically use state computed on the train set (e.g., the arithmetic mean is used to normalize a numerical feature or the number of categories is used to one-hot encode acategorical variable)
- Model-specific transformations functions need **identical implementations** in the training and inference pipelines. If the implementations differ, you may introduce training-inference skew.
- Training-inference skew is difficult to diagnose and fix, and causes models to perform poorly.

Model-Specific Transformations can be applied by Feature Views (2/3)

```
min_max_scaler = fs.get_transformation_function(name="min_max_scaler")
label_encoder = fs.get_transformation_function(name="label_encoder")

transformation_functions = {
    "category": label_encoder,
    "amount": min_max_scaler,
    "trans_volume_mavg": min_max_scaler,
    "trans_volume_mstd": min_max_scaler,
    "loc_delta_mavg": min_max_scaler,
    "trans_freq": min_max_scaler,
    "loc_delta_t_minus_1": min_max_scaler,
    "time_delta_t_minus_1": min_max_scaler,
    "age_at_transaction": min_max_scaler,
    "days_until_card_expires": min_max_scaler,
}

feature_view = fs.create_feature_view(
    name='cc_trans_fraud_all',
    query=ds_query,
    labels=["fraud_label"],
    transformation_functions=transformation_functions
)
```

Built-in transformation functions in Hopsworks

Specify which transformation functions are applied to which features

Apply the transformations to the features in the Feature View

Model-Specific Transformations can be applied by Feature Views (3/3)

```
fv = fs.get_feature_view(name='cc_trans_fraud_all', version=1)


X_train, y_train, X_test, y_test = fv.train_test_split(test_ratio=0.2)


fv.batch_init(td_version)
df_to_score = fv.batch_data()


fv.init(td_version)
keys = {“cc_num” : “1111 2222 3333 4444”}
array_features = fv.get_feature_vector(keys=keys)

model.predict(array_features)
```

Transformations are applied before returning the DataFrames

td_version needed to identify state used by transformations. Transformations are applied before returning the DataFrame

Transformations are applied before returning the feature vector

Consistent Training/Inference Transformations with Scikit-Learn

- Save the transformation pipeline object in the model registry along with the model
- In the inference pipeline, deserialize the transformation pipeline object
  - Note: ensure the same version of scikit-learn that was used in training and is used in the

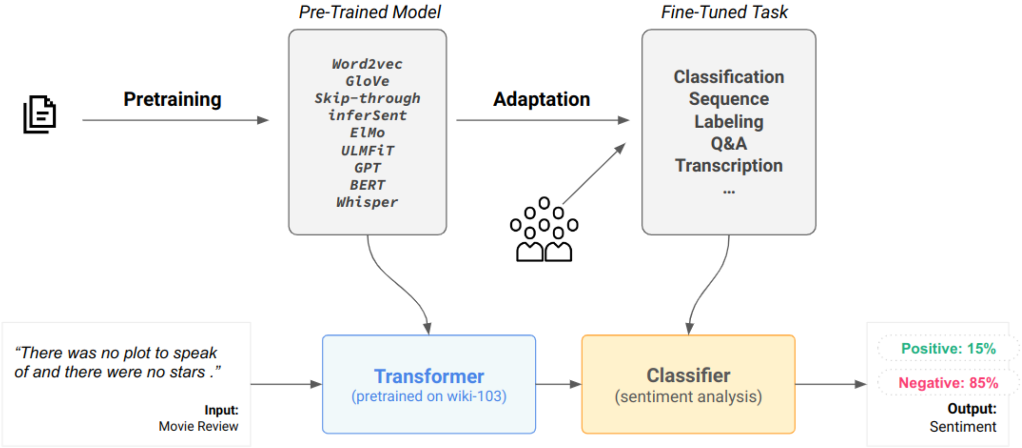
```
joblib.dump(model, model_dir + "cc_fraud/cc_fraud_model.pkl")
joblib.dump(transformer, model_dir + "cc_fraud/cc_fraud_trans.pkl")

iris_model = mr.python.create_model( … )
iris_model.save(model_dir)
```

```
the_model = mr.get_model("cc_fraud_model", version=1)
model_dir = the_model.download()
transformer = joblib.load(model_dir + "cc_fraud/cc_fraud_trans.pkl")
model = joblib.load(model_dir + "cc_fraud/cc_fraud_model.pkl")
```

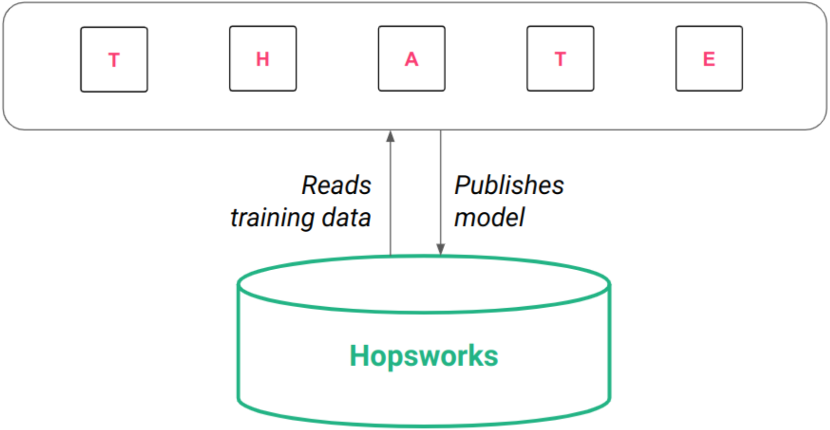
Same transformation pipeline object used in training and online inference

Example Notebook: https://github.com/logicalclocks/hopsworks-tutorials/blob/master/iris/iris_sklearn.ipynb

Use Pretrained Models and Transfer Learning, where appropriate



Typical steps in a training pipeline that uses a Feature Store



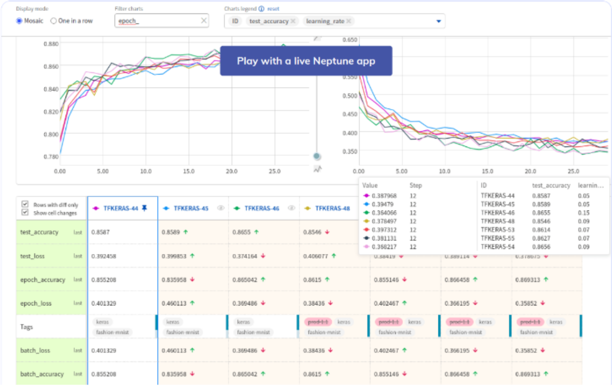
T-HATE = Model-Specific feature **T**ransformations, **H**yperparameter tuning, compile model **A**rchitecture, **T**rain model (fit to the data), **E**valuate your model.

Experiment tracking tools help manage your training pipelines

- Use Experiment Tracking Platforms to track and organize training pipeline outputs

- Free Serverless Experiment Tracking Systems
  - Weights and Biases
  - Comet ML
  - Neptune
  - MLFlow with Infinstor

- Open-source
  Experiment Tracking Tools
  - MLFlow
  - Tensorboard



[Image from Neptune]

Common training pipeline pattern when using a Feature Store

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import xgboost as xgb

X_train,X_test,y_train,y_test =
    train_test_split(features, labels, test_size=0.2)

model = xgb.XGBClassifier()

model.fit(X_train,y_train)

y_pred = model.predict(X_test)

report_dict = classification_report(
                 y_test, y_pred, output_dict=True)
```

Get train and test data sets as features (X) and labels (y)

Use XGBoost as modelling algorithm

Train supervised ML classifier with features and labels from train set

Generate predictions with model on test features (X_test)

Evaluate model performance by comparing predictions (y_pred) and labels (y_test) for the test set

Training Pipeline output - save your model to a Model Registry

```
project =  hopsworks.login()
mr = project.get_model_registry()

model_dir="iris_model"
os.mkdir(model_dir)
joblib.dump(model, model_dir + "/iris_model.pkl")

input_example = X_train.sample()
input_schema = Schema(X_train)
output_schema = Schema(y_train)
model_schema = ModelSchema(input_schema, output_schema)

iris_model = mr.python.create_model(
    version=1,
    name="iris",
    metrics={"accuracy" : metrics['accuracy']},
    model_schema=model_schema,
    input_example=input_example,
    description="Iris Flower Predictor")

iris_model.save(model_dir)
```

Save an input example to be used for testing a model deployment

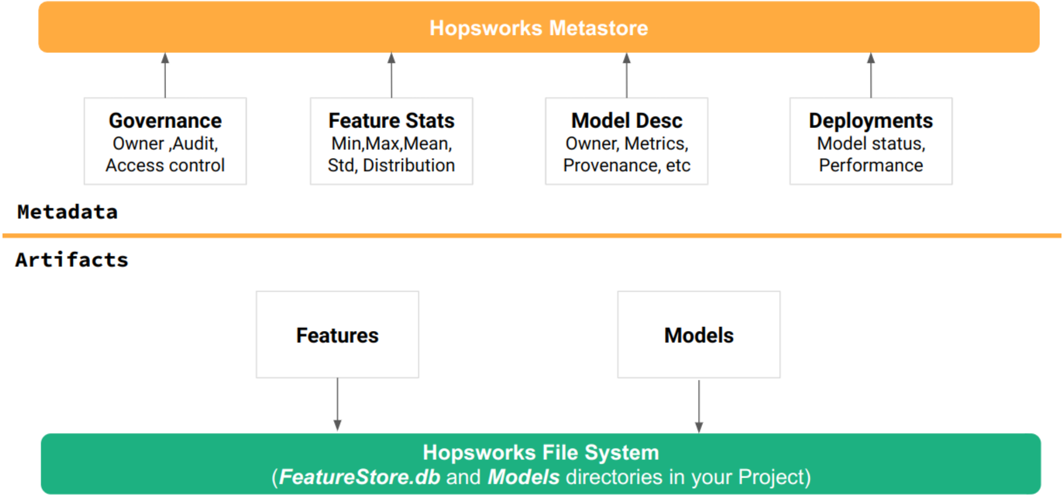
The Model API is defined as a Schema

mr.tensorflow.create_model(...)
mr.sklearn.create_model(...)

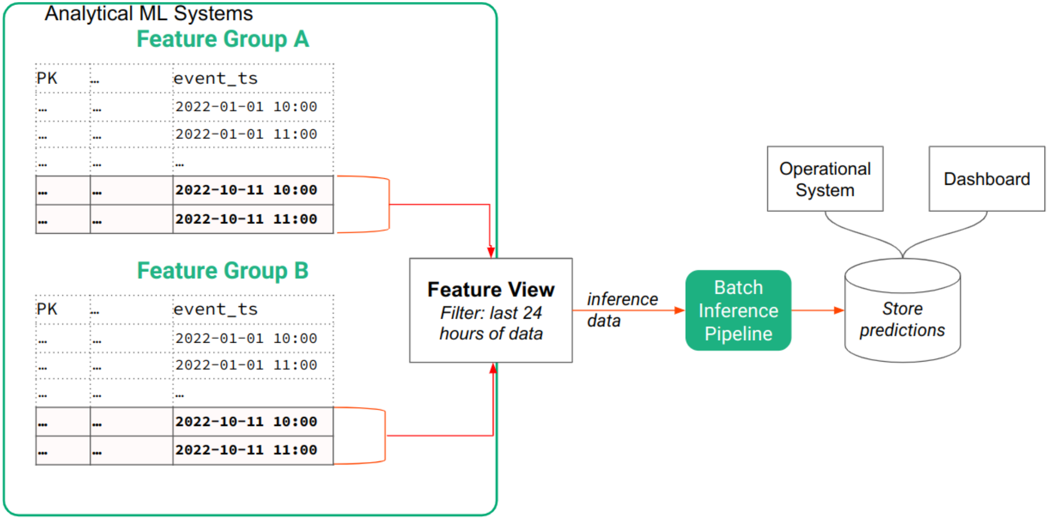
Pass any dict of metrics here

All files in *model_dir* are stored in the model registry, along with the model

Hopsworks is both a Metadata and Artifact Store



Batch Inference Pipeline uses features from the Feature Store



Batch Inference Pipeline Code for Scoring Data from Last 24 hours

```python
feature_view = fs.get_feature_view("cc_trans_fraud_all", 1)

feature_view.init_batch_scoring(training_dataset_version=1)

start_date = (datetime.datetime.now() -
              datetime.timedelta(hours=24))

end_time = datetime.datetime.now()

transactions_df = feature_view.get_batch_data(
          start_time = start_time, end_time = end_time)
features_df = transactions_df.iloc[: , 3:]

mr = project.get_model_registry()
the_model = mr.get_model("cc_fraud_model", version=1)
model_dir = the_model.download()
model = joblib.load(model_dir + "/cc_fraud_model.pkl")

predictions = model.predict(features_df)
```

Init with TD version used by model

Start timestamp for inference data

End timestamp for inference data

Get the inference data between the start and end times as a DF

Drop PK and helper columns

Download the model from the model registry

Return predictions for inference data

References

- Feature Group Concepts, Feature Group Guide, API Docs for Feature Groups- https://docs.hopsworks.ai
- Data models - star schema - https://www.databricks.com/glossary/star-schema
- Credit Card Fraud - https://www.kaggle.com/datasets/kartik2112/fraud-detection