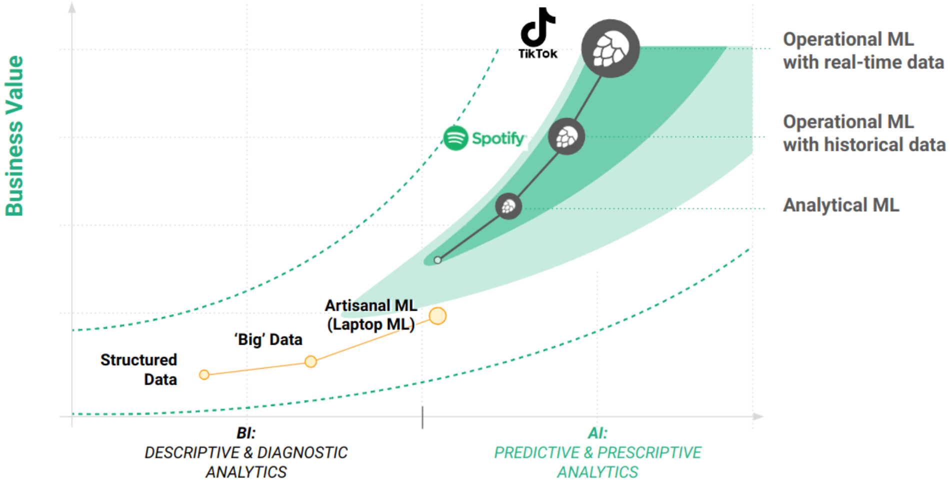
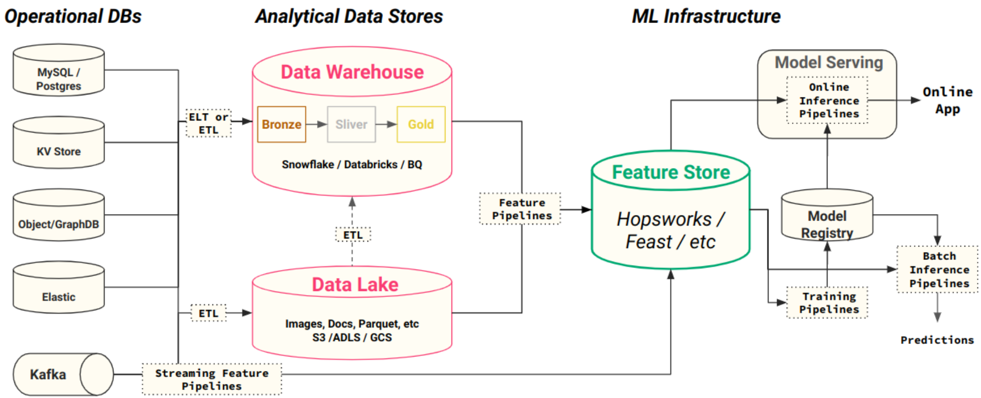
Serverless Machine Learning

Jim Dowling
jdowling@kth.se
2022-11-04

Enterprise AI Value Chain

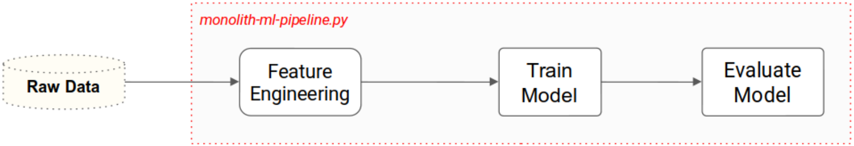


Modern Enterprise Data and ML Infrastructure



Monolithic ML Pipeline

- A pipeline is a program that takes and input and produces an output
- End-to-end ML Pipelines are a single pipeline that transforms raw data into features and trains and scores the model in one single program



Problems with Monolithic ML Pipelines

- They are often not modular - their components are not modular and cannot be independently scaled or deployed on different hardware (e.g., CPUs for feature engineering, GPUs for model training).
- They are difficult to test - production software needs automated tests to ensure features and models are of high quality.
- They tightly couple the execution of feature engineering, model training, and inference steps - running them in the same pipeline program at the same time.
- They do not promote reuse of features/models/code. The code for computing features (feature logic) cannot be easily disentangled from its pipeline jungle.

Modularity enables more Robust and Scalable Systems

Modular water pipes in a Google Datacenter. Instead of one giant water pipe (our monolithic notebook), separate water pipes reduce the blast radius if one fails. Color coding makes it easier to debug problems in a damaged water pipe.

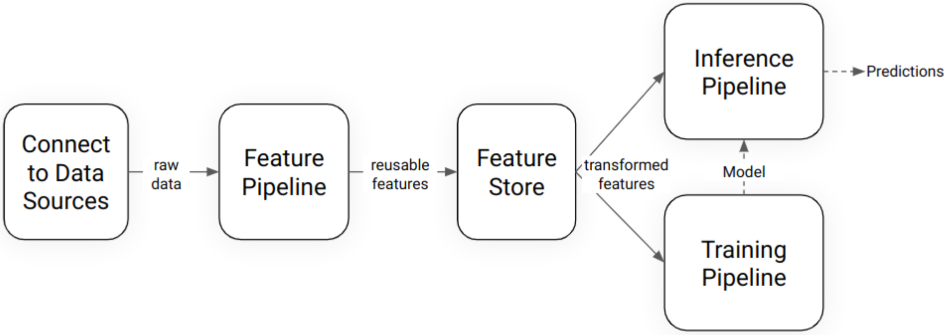
Pipelines as Modular Programs

- Modularity involves structuring your code such that its functionality is separated into independent classes and/or functions that can be more easily reused and tested.
- Modules should be placed in accessible classes or functions, keeping them small and easy to understand and document.
- Modules enable code to be more easily reused in different pipelines.
- Modules enable code to be more easily independently tested, enabling the easier and earlier discovery of bugs.

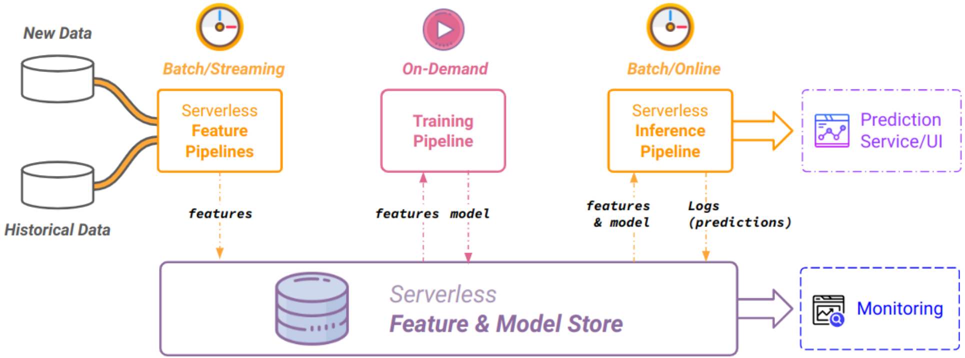
Supervised ML Pipeline Stages

$$\mathtt{train}(\mathtt{features}, \mathtt{labels}) -> \mathtt{model}$$

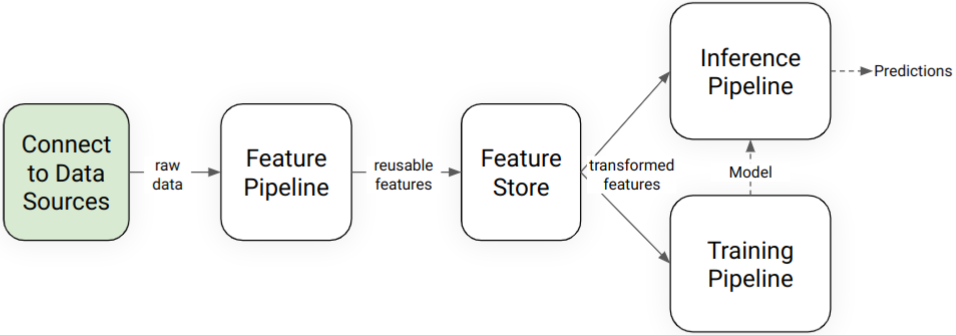
$$\mathtt{model}(\mathtt{features}) -> \mathtt{predictions}$$



ML Pipeline Stages in a Serverless Machine Learning System



ML Pipeline Stages - Data Sources



Connect to Data Sources and Read Raw Data

- Discover data sources, securely connect to heterogeneous data sources
- Manage dependencies such as connectors and drivers
- Manage connection information securely: network endpoint, database/table names, authentication credentials such as API keys or credentials (username/password)

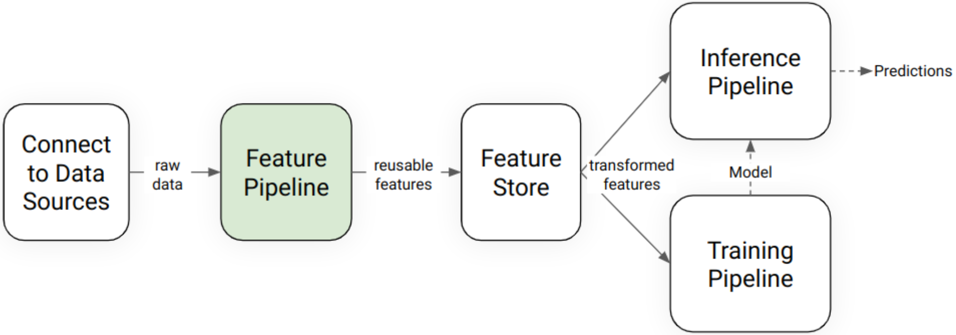
Heterogeneous Data Sources

Type of Data	Examples
Tabular data	Customer, transactions, marketing, sales, etc
Unstructured data	images, sound, video
Free-text search data	application/service logs
Documents / Objects	JSON
Graph data	Social network graphs
Time-series data	Performance metrics
Queued data	Messages, events
REST APIs	Salesforce, Hubspot, etc
Web scraped	Electricity prices, air quality

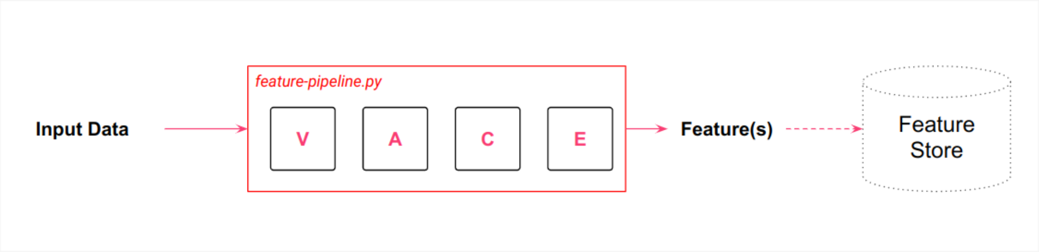
File Formats for different Data Sources

Type of Data	File Formats	Example Systems
Tabular data	.csv, .parquet, .tfrecord, .avro	Snowflake, Databricks, BQ, Redshift, S3, ADLS, GCS
Unstructured data	images, sound, video	S3, GCS, ADLS, HDFS
Free-text search	application/service logs	Elasticsearch, Solr
Documents	JSON	MongoDB
Graph data	Social networks	Neo4J
Time-series data	Performance metrics	InfluxDB, Prometheus
Queued data	Avro	Kafka, Kinesis
REST APIs	REST API with API key	Saas Platform
Web scraped	N/A	Websites publishing data

ML Pipeline Stages - Feature Pipelines



Feature Pipelines



VACE = **V**alidate, **A**ggregate, **C**ompress (dimensionality reduction), **E**xtract (Binning, Crosses, etc)

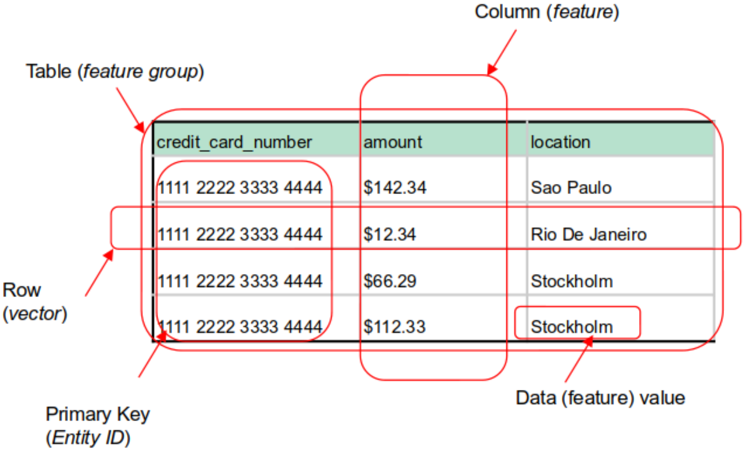
Feature Pipelines

- A feature pipeline is a program that orchestrates the execution of feature engineering steps on input data to create feature values.

Examples of feature engineering steps:

- Clean, validate, data
- Data de-duplication, pseudononymization, data wrangling
- Feature extraction, aggregations, dimensionality reduction, feature binning, feature crosses

Tabular Data



Tabular Data as Features, Labels, Entity (or Primary) Keys, Event Time

Entity Key	event_time	Feature	Feature	Label
credit_card_number	event_time	amount	location	Fraud
1111 2222 3333 4444	2022-01-01 08:44	$142.34	Sao Paulo	False
1111 2222 3333 4444	2022-01-01 19:44	$12.34	Rio De Janeiro	False
1111 2222 3333 4444	2022-01-01 20:44	$66.29	Stockholm	True
1111 2222 3333 4444	2022-01-01 20:55	$112.33	Stockholm	True

Row

Feature Vector

Tabular Data in Pandas

Object	Datetime	Float64	Object	Bool	
credit_card_number	event_time	amount	location	Fraud	
1111 2222 3333 4444	2022-01-01 08:44	$142.34	Sao Paulo	False	
1111 2222 3333 4444	2022-01-01 19:44	$12.34	Rio De Janeiro	False	
1111 2222 3333 4444	2022-01-01 20:44	$66.29	Stockholm	True	Row
1111 2222 3333 4444	2022-01-01 20:55	$112.33	Stockholm	True	

Exploratory Data Analysis in Pandas

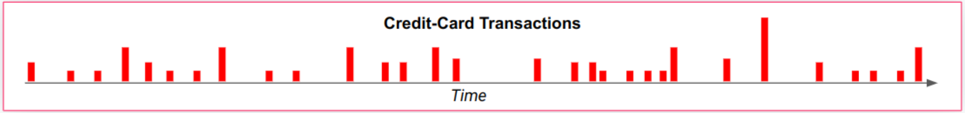
Useful EDA Commands	Description
df.head()	Returns the first few rows of *df*.
df.describe()	Returns descriptive statistics for *df*. Use with numerical features.
df[col].unique()	Returns all values unique for a column, *col*, in *df*.
df[col].nunique()	Returns the number of unique values for a column, *col*, in *df*.
df.isnull().sum()	Returns the number of null values in all columns in *df*.
df[col].value_counts()	Returns the number of values for with different values. Use with both numerical and categorical variables.
sns.histplot(...)	Plot a histogram for a DataFrame or selected columns using Seaborn.

Aggregations in Pandas

Aggregation	Description
df.count()	Count the number of rows
df.first(), df.last()	First and last rows
df.mean(), df.median()	Mean and median
df.min(), df.max()	Minimum and maximum
df.std(), df.var()	Standard deviation and variance
df.mad()	Mean absolute deviation
df.prod()	Product of all rows
df.sum()	Sum of all rows

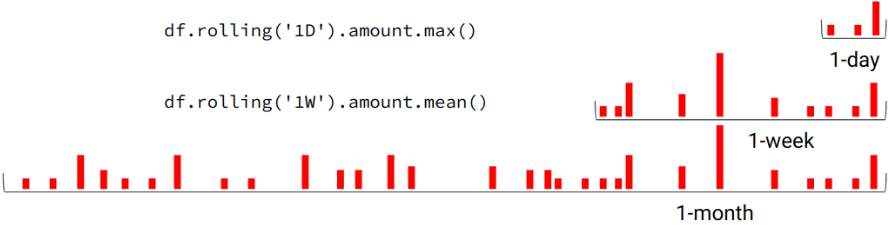
What is the 7 day rolling max/mean of the credit card transaction amounts?

```
# For rolling windows in Pandas, first set a DateTime column as index to the df
```



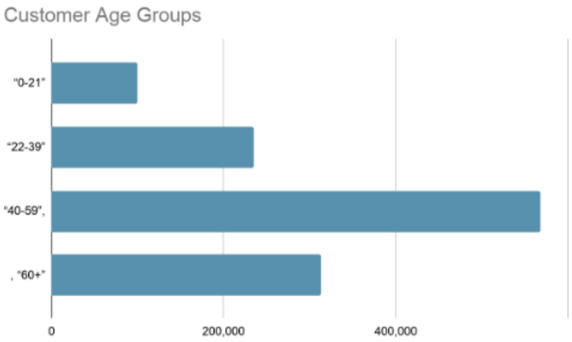

```
df.rolling('1D').amount.max()

df.rolling('1W').amount.mean()
```




```
df.rolling('30D').amount.min()
```

Feature binning



Feature Crosses

- A feature cross is a synthetic feature formed by multiplying (crossing) two or more features. By multiplying features together, you encode nonlinearity in the feature space.
- For example, imagine we are looking for credit card fraud activity within a geographic region (e.g., a city district), how would we capture that as a feature?
- We could cross to a geographic area (binned latitude and binned longitude - a grid identifying a city district) with the level of credit card activity within that geographic area.

Binned Latitude	Binned Longitude	cc_spend_1hr
xx	yy	8000

$\longrightarrow$

b-lat⊕b-long⊕cc_spend_1hr
xx⊕yy⊕8000

Embeddings as Features

- An embedding is a lower dimension representation of a sparse input that retains some of the semantics of the input.
- An embedding store (vector database) stores semantically similar inputs close together in the embedding space. You can implement “similarity search” by finding embeddings close in embedding space. You can even apply arithmetic on embeddings to discover semantic relationships.

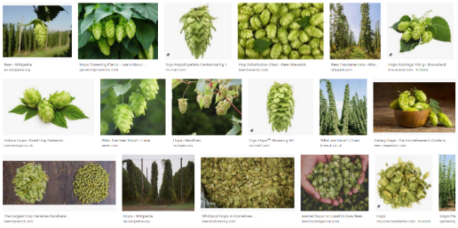
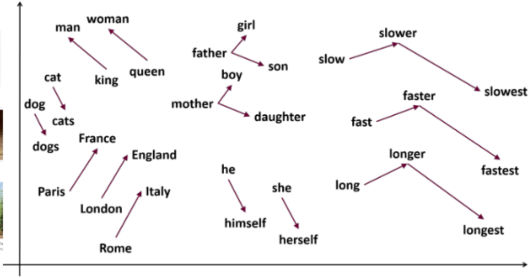
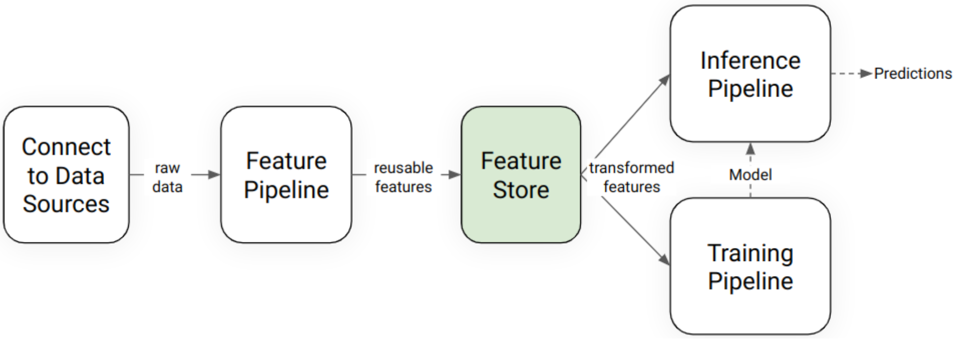


Image Embeddings enable Similarity Search



ML Pipeline Stages - Feature Store



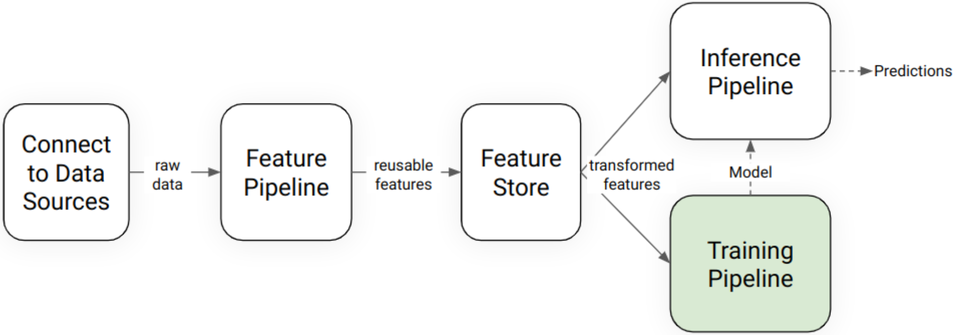
Store Features

There are two general ways people manage features and labels for both training and serving:

- (1) Compute features on-demand as part of the model training or batch inference pipeline.

- (2) Use a **feature store** to store the features so that they can be reused across different models for both training and inference. For **online models** that require features with either **historical or contextual information**, feature stores are typically used.

ML Pipeline Stages - Training Pipelines

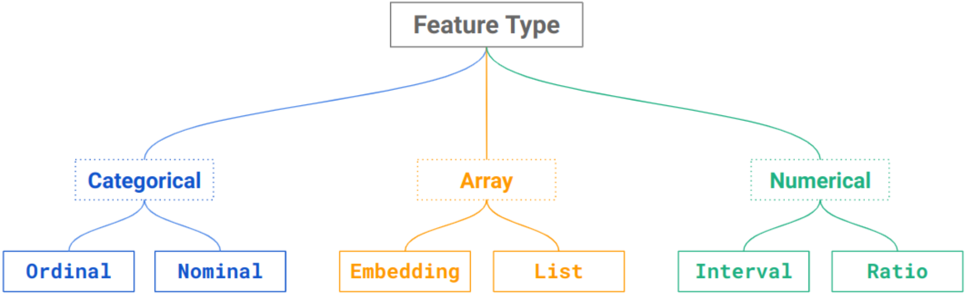


Feature Types

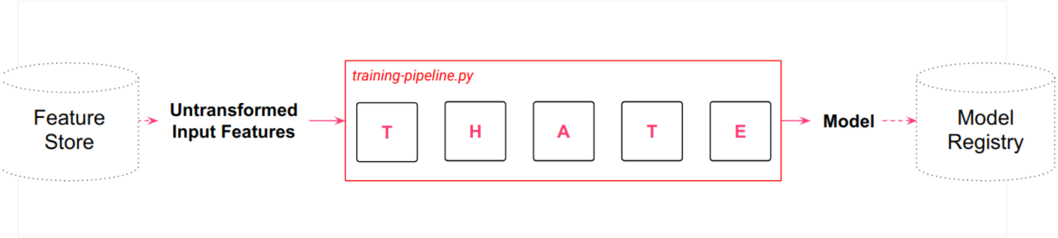
credit_card_number	event_time	amount	location	Fraud
<primary_key>	***<event_time>***	***<numerical feature>***	***<categorical feature>***	***<label>***
1111 2222 3333 4444	2022-01-01 08:44	$142.34	Sao Paulo	False
1111 2222 3333 4444	2022-01-01 19:44	$12.34	Rio De Janeiro	False
1111 2222 3333 4444	2022-01-01 20:44	$66.29	Stockholm	True
1111 2222 3333 4444	2022-01-01 20:55	$112.33	Stockholm	True

Reference: https://www.hopsworks.ai/post/feature-types-for-machine-learning

Feature Types Taxonomy



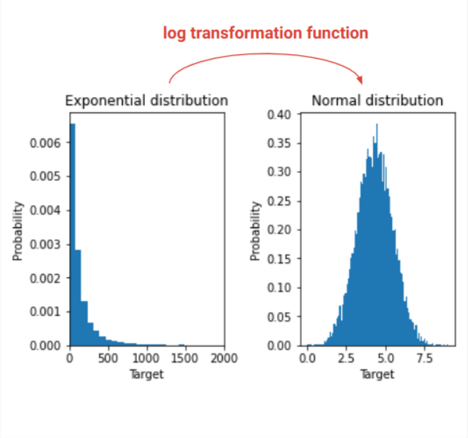
Model Training Pipelines




```
T-HATE =
Transform features, Hyperparameter tuning, model Architecture, Train model (fit to data), Evaluate your model.
```

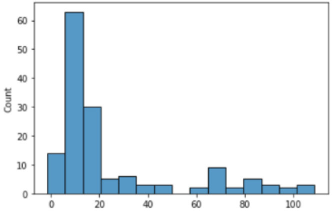
Model-Dependent Transformations

- **Transformations for data compatibility**
  - Convert non-numeric features into numeric
  - Resize inputs to a fixed size
- **Transformations to improve model performance**
  - Many models perform badly if numerical features do not follow a normal (Gaussian) distribution
  - Tokenization or lower-casing of text features
  - Allowing linear models to introduce non-linearities into the feature space



Reference: https://developers.google.com/machine-learning/data-prep/transform/introduction

Transformations in Pandas

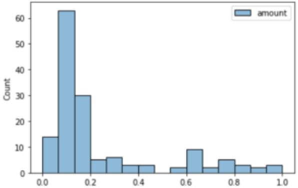



```
columns = ['amount']
df_exp = pd.DataFrame(data = array, columns = columns)
```

```
# Min-Max Normalization in Pandas
df_norm = (df_exp-df_exp.min())/(df_exp.max()-df_exp.min())
df_norm.head()
```

```
sns.histplot(df_norm)
```

```
<AxesSubplot:ylabel='Count'>
```



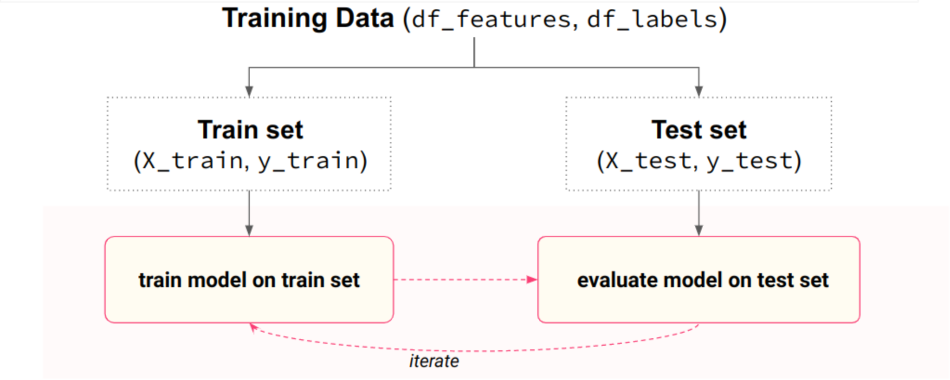
Different types of Transformations

Type of Transformation
Scaling to Minimum And Maximum values
Scaling To Median And Quantiles
Gaussian Transformation
Logarithmic Transformation
Reciprocal Transformation
Square Root Transformation
Exponential Transformation
Box Cox Transformation

ML Algorithms that may need Transformations
Linear regression
Logistic regression
K Nearest neighbours
Neural networks
Support vector machines with radial bias kernel functions
Principal components analysis
Linear discriminant analysis

Note: tree-based models do not need transformations

Model Training with Train and Test Sets




```
X_train = training set features
y_train = training set labels
```

```
X_test is test set features
y_test is test set labels
```

Model Training with Train and Test Sets in Scikti-Learn

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

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

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

```python
model = xgb.XGBClassifier()
```

Use XGBoost as modelling algorithm

```python
model.fit(X_train,y_train)
```

Train supervised ML classifier with features and labels from train set

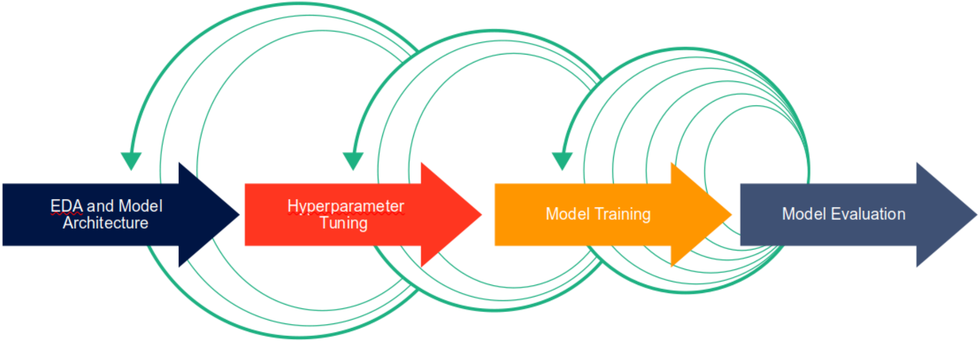
```python
y_pred = model.predict(X_test)
```

Generate predictions with model on test features (X_test)

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

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

Model Training is an Iterative Process



Model-Centric Iteration to Improve Model Performance

Possible steps to improve your model performance:

- Try out a different supervised ML learning algorithm (e.g., random forest, feedforward deep neural network, Gradient-boosted decision tree)
- Try out new combinations of hyperparameters (e.g., number of training epochs, the learning rate, number of layers in a deep neural network, adjust regularizations such as Dropout or BatchNorm)
- Evaluate your model on a validation set (keeping a separate holdout test set for final model performance evaluation)

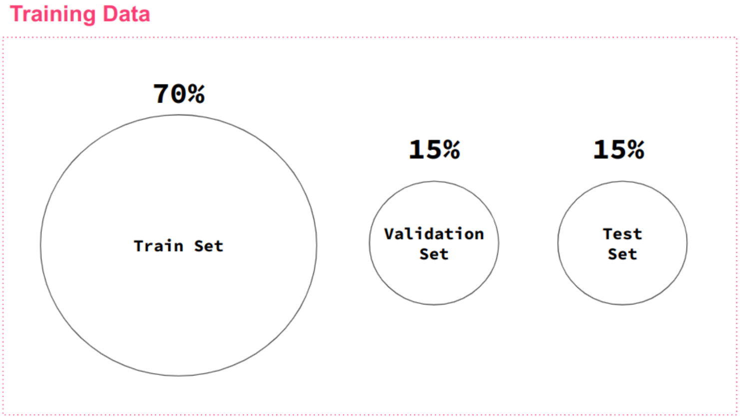
Data-Centric Iteration to Improve Model Performance

Steps to improve your model

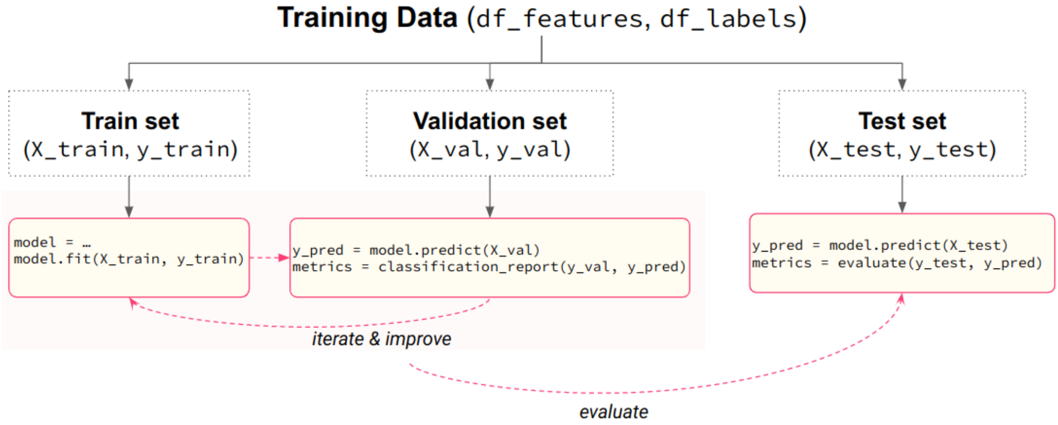
- Add or remove features to or from your model (feature selection)
- Add more training data
- Remove poor quality training samples
- Improve the quality of existing training samples (e.g., using Cleanlab or Snorkel)
- Rank the importance of the training samples (Active Learning)

Train, Validation, and Test Sets

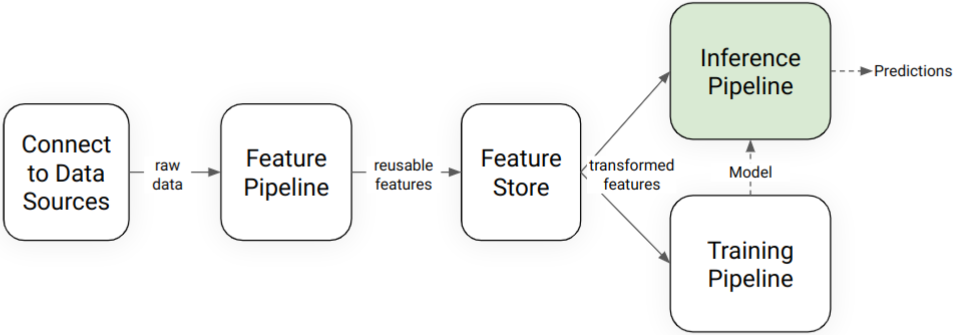
- Random splits of the training data when the data is not time-series data
- Time-series splits of the training data when the data is time-series data



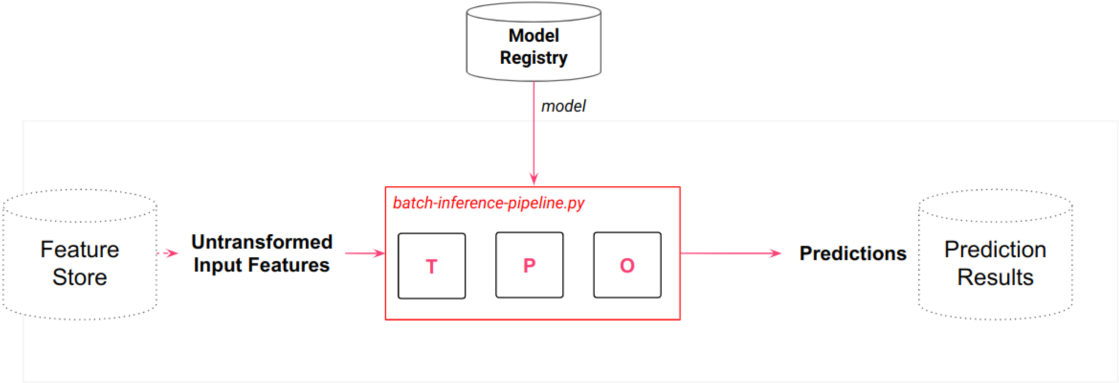
Model Training is an Iterative Process



ML Pipeline Stages - Inference Pipelines

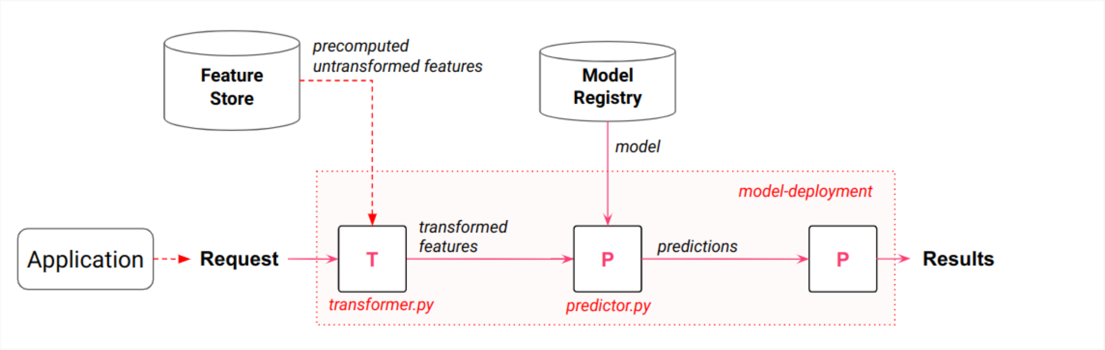


Batch Inference Pipeline



`TPO = Transform features, Predict, Output.`

Online Inference Pipeline



`TPP = Transform the input request into features, Predict using input features and the model, Post-process predictions, before output results.`

Serverless ML with Python

- Write Feature, Training, and Inference Pipelines in **Python**
- Orchestrate the execution of Pipelines using Serverless Compute Platforms
- Store features and models in a serverless feature/model store
- Run a User Interface (UI), written in Python, on serverless infrastructure

Serverless Compute Platforms

Serverless Python Functions

- **Modal**
- **GitHub Actions**
- render. com
- pythonanywhere.com
- replit.com
- deta.sh
- linode.com
- hetzner.com
- digitalocean.com
- AWS lambda functions
- Google Cloud Functions

Orchestration Platforms

- **Modal**
- **GitHub Actions**
- Astronomer (Airflow)
- Dagster
- Prefect
- Azure Data Factory
- Amazon Managed Workflows for Apache Airflow (MWAA)
- Google Cloud Composer
- Databricks Workflows

Serverless Feature Stores and Model Registry/Serving

Feature Stores

- **Hopsworks**

Model Registry and Serving

- **Hopsworks**
- AWS Sagemaker
- Databricks
- Google Vertex

Serverless User Interfaces

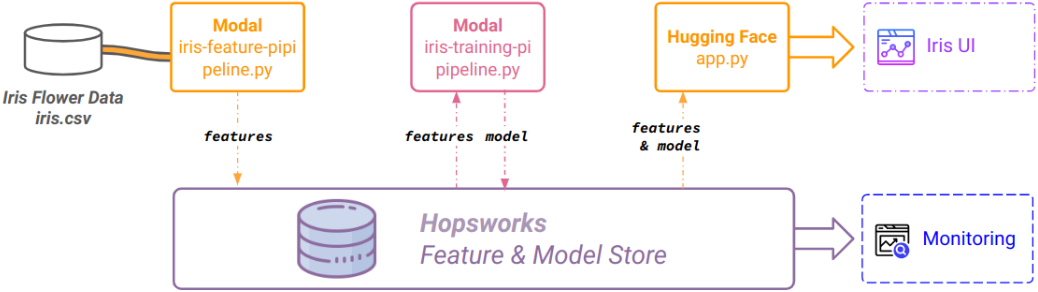
- **Hugging Faces Spaces**
- Streamlit Cloud

Iris Flower Dataset

https://github.com/ID2223KTH/id2223kth.github.io/tree/master/src/serverless-ml-intro

- 4 input features: sepal length, sepal width, petal length, petal width
- label (target): Iris Flower Type (one of Setosa, Versicolor, Virginica)
- Only 150 samples in the dataset

Serverless Iris with Modal, Hopsworks, and Hugging Face



Iris Flowers: Feature Pipeline with Modal and Hopsworks

```python
import os
import modal
stub = modal.Stub()
hopsworks_image = modal.Image.debian_slim().pip_install(["hopsworks"])

@stub.function(image=hopsworks_image, schedule=modal.Period(days=1), \
    secret=modal.Secret.from_name("jim-hopsworks-ai"))
def f():
    import hopsworks
    import pandas as pd
    project = hopsworks.login()
    fs = project.get_feature_store()
    iris_df = pd.read_csv("https://repo.hops.works/master/hopsworks-tutorials/data/iris.csv")
    iris_fg = fs.get_or_create_feature_group( name="iris_modal", version=1,
        primary_key=["sepal_length","sepal_width","petal_length","petal_width"],
        description="Iris flower dataset")
    iris_fg.insert(iris_df)

if __name__ == "__main__":
    with stub.run():
        f()
```

Training Pipeline with Modal and Hopsworks

```
@stub.function(image=hopsworks_image, schedule=modal.Period(days=1),\
    secret=modal.Secret.from_name("jim-hopsworks-ai"))
def f():
    # lots of imports
    project = hopsworks.login()
    fs = project.get_feature_store()
    try:
        feature_view = fs.get_feature_view(name="iris_modal", version=1)
    except:
        iris_fg = fs.get_feature_group(name="iris_modal", version=1)
        query = iris_fg.select_all()
        feature_view = fs.create_feature_view(name="iris_modal",
                                          version=1,
                                          description="Read from Iris flower dataset",
                                          labels=["variety"],
                                          query=query)
    X_train, X_test, y_train, y_test = feature_view.train_test_split(0.2)
    model = KNeighborsClassifier(n_neighbors=2)
    model.fit(X_train, y_train.values.ravel())
```

Training Pipeline (ctd)

```
y_pred = model.predict(X_test)
metrics = classification_report(y_test, y_pred, output_dict=True)
results = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(results, ['True Setosa', 'True Versicolor', 'True Virginica'],
                     ['Pred Setosa', 'Pred Versicolor', 'Pred Virginica'])
cm = sns.heatmap(df_cm, annot=True)
fig = cm.get_figure()
joblib.dump(model, "iris_model/iris_model.pkl")
fig.savefig("iris_model/confusion_matrix.png")
input_schema = Schema(X_train)
output_schema = Schema(y_train)
model_schema = ModelSchema(input_schema, output_schema)
mr = project.get_model_registry()
iris_model = mr.python.create_model(
    name="iris_modal",
    metrics={"accuracy" : metrics['accuracy']},
    model_schema=model_schema,
    description="Iris Flower Predictor")
iris_model.save("iris_model")
```

Interactive Inference Pipeline with Hugging Face/Hopsworks

```
model = mr.get_model("iris_modal", version=1)
model_dir = model.download()
model = joblib.load(model_dir + "/iris_model.pkl")

def iris(sepal_length, sepal_width, petal_length, petal_width):
    input_list = []
    input_list.append(sepal_length)
    input_list.append(sepal_width)
    input_list.append(petal_length)
    input_list.append(petal_width)
    res = model.predict(np.asarray(input_list).reshape(1, -1))
    flower_url = "https://raw.githubusercontent.com/.../assets/" + res[0] + ".png"
    return Image.open(requests.get(flower_url, stream=True).raw)

demo = gr.Interface(
    fn=iris, title="Iris Flower Predictive Analytics", allow_flagging="never",
    description="Experiment with sepal/petal lengths/widths to predict which flower it is.",
    inputs=[ gr.inputs.Number(default=1.0, label="sepal length (cm)"),
        gr.inputs.Number(default=1.0, label="sepal width (cm)"),
        gr.inputs.Number(default=1.0, label="petal length (cm)"),
        gr.inputs.Number(default=1.0, label="petal width (cm)"),],
    outputs=gr.Image(type="pil"))
demo.launch()
```

Questions?

Acknowledgements

Some of the images are used with permission from Hopsworks AB.