

Feature engineering and selection

END-TO-END MACHINE LEARNING



Joshua Stapleton
Machine Learning Engineer

Feature engineering

Creating features

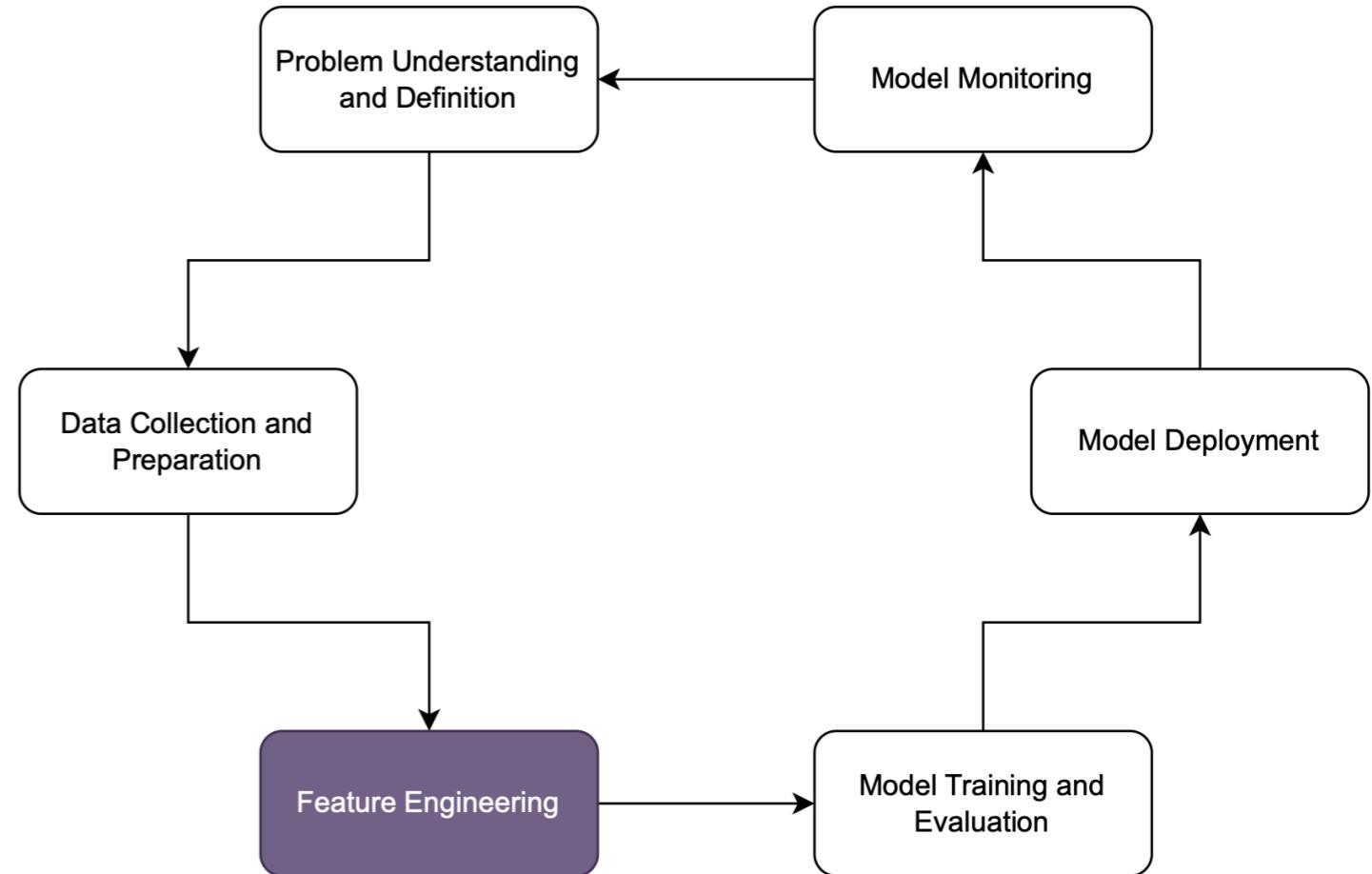
- Simplifies problem
- Improves model efficiency

Techniques

- Modify pre-existing features
- Design new features

Benefits

- Easier deployment, maintenance, training
- Interpretability gain



Normalization

- Scales numeric features to [0, 1]
- Helpful when features have different scales/ranges.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer

# Split the data
X_train, X_test = train_test_split(df, test_size=0.2, random_state=42)
# Create a normalizer object, fit on training data, normalize, and transform test set
norm = Normalizer()
X_train_norm = norm.fit_transform(X_train)
X_test_norm = norm.transform(X_test)
```

Standardization

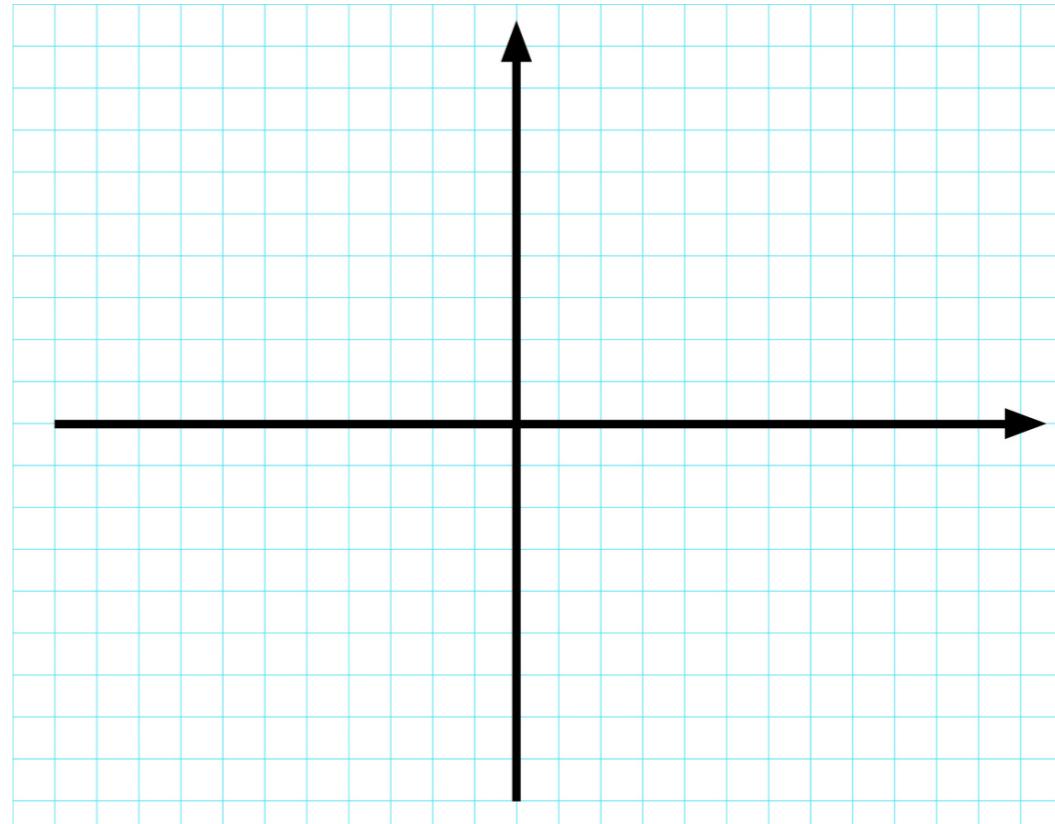
- Scales data to have mean = 0, variance = 1
- Beneficial for algorithms that assume similar mean and variance

```
from sklearn.preprocessing import StandardScaler

# Split the data
X_train, X_test = train_test_split(df, test_size=0.2, random_state=42)
# Create a scaler object and fit training data to standardize it
sc = StandardScaler()
X_train_stzd = sc.fit_transform(X_train)
# Only standardize the test data
X_test_stzd = sc.transform(X_test)
```

What constitutes a good feature?

- Use relevant features
- Weather on the day of patient appointment should have no bearing on diagnosis
- Use dissimilar (orthogonal) features
- Two features of age in months and age in years would not be helpful



sklearn.feature_selection

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split

# Splitting data into train and test subsets first to avoid data leakage
X_train, X_test, y_train, y_test = train_test_split(
    heart_disease_df_X, heart_disease_df_y, test_size=0.2, random_state=42)
```

sklearn.feature_selection (cont.)

```
# Define and fit the random forest model
rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
rf.fit(X_train, y_train)

# Define and run feature selection
model = SelectFromModel(rf, prefit=True)
features_bool = model.get_support()
features = heart_disease_df.columns[features_bool]
```

Let's practice!

END-TO-END MACHINE LEARNING

Model training

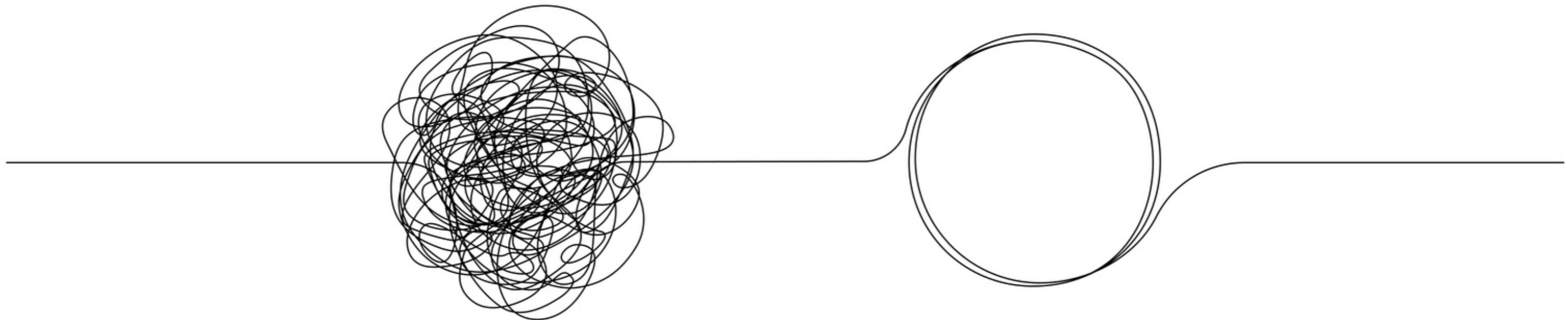
END-TO-END MACHINE LEARNING



Joshua Stapleton
Machine Learning Engineer

Occam's Razor

- Simplest satisfactory explanation is best
- Lean towards simple models when selecting



Modeling options

Logistic Regression

- Finds decision boundary between classes
- `sklearn.linear_model.LogisticRegression`

Support Vector Classifier

- Finds plane to separate classes
- `sklearn.svm.SVC`

Decision Tree

- Finds simple 'rules' to classify data
- `sklearn.tree.DecisionTreeClassifier`

Random Forest

- Combines multiple decision trees
- `sklearn.ensemble.RandomForestClassifier`

Other models

Deep learning models

- Neural Networks
- Convolutional Neural Networks
- Generative Pretrained Transformer (GPT)

K-Nearest Neighbors (KNN)

- Supervised learning algorithm

XGBoost

- Gradient boosted model
- <https://xgboost.readthedocs.io/en/stable/>

Training principles

Model:

- Uses cleaned and feature-handled dataset
- Learns patterns in training data
- Aims to predict target of heart disease diagnosis

Principles:

- Model must generalize to unseen data (outside of training set)
- 'Hold-out' some data to test model on after training completes.
- Split of training/testing is normally 70/30 or 80/20
- Can use `sklearn.model_selection.train_test_split`

Training a model

```
# Importing necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Split the data into training and testing sets (80:20)
X_train, X_test, y_train, y_test = train_test_split(features, heart_disease_y,
                                                    test_size=0.2, random_state=42)

# Define the models
logistic_model = LogisticRegression(max_iter=200)

# Train the model
logistic_model.fit(X_train, y_train)
```

Getting model predictions

```
# Jane Doe's health data, for example: [age, cholesterol level, blood pressure, etc.]  
jane_doe_data = [45, 230, 120, ...]  
  
# Reshape the data to 2D, because scikit-learn expects a 2D array-like input  
jane_doe_data = jane_doe_data.reshape(1, -1)  
  
# Use the model to predict Jane's heart disease diagnosis probabilities  
jane_doe_probabilities = logistic_model.predict_proba(jane_doe_data)  
jane_doe_prediction = logistic_model.predict(jane_doe_data)
```

Getting model predictions (cont.)

```
# Print the probabilities  
print(f"Jane Doe's predicted probabilities: {jane_doe_probabilities[0]}")  
print(f"Jane Doe's predicted health condition: {jane_doe_prediction[0]}")
```

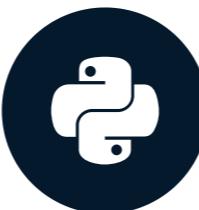
```
Jane Doe's predicted health condition probabilities: [0.2 0.8]  
Jane Doe's predicted health condition: 1
```

Let's practice!

END-TO-END MACHINE LEARNING

Logging experiments on MLFlow

END-TO-END MACHINE LEARNING



Joshua Stapleton
Machine Learning Engineer

MLFlow

Without MLflow...

- Many untracked, disorganized experiment runs
- Dissimilar, or incomparable runs
- Unreproducible, lost runs

With MLflow...

- Tracked, organized experiment runs
- Comparison between standardized runs
- Reproducible runs
- Share, deploy models

Creating experiments

`mlflow.set_experiment()`

- Sets experiment name
- Provides workspace for experiment runs

Usage:

```
import mlflow

# Set an experiment name, which is a workspace for your runs
mlflow.set_experiment("Heart Disease Classification")
```

Running experiments

```
# Start a new run in this experiment
with mlflow.start_run():

    # Train a model, get the prediction accuracy
    logistic_model = LogisticRegression()

    # Log parameters, eg:
    mlflow.log_param("n_estimators", logistic_model.n_estimators)

    # Log metrics (accuracy in this case)
    mlflow.log_metric("accuracy", logistic_model.accuracy)

    # Print out metrics
    print("Model accuracy: %.3f" % accuracy)
```

Model accuracy: 0.96

Retrieving experiments

```
mlflow.get_run(run_id)
```

- Metadata for specific run

```
mlflow.search_runs()
```

- Returns DataFrame of metrics for multiple runs

Usage:

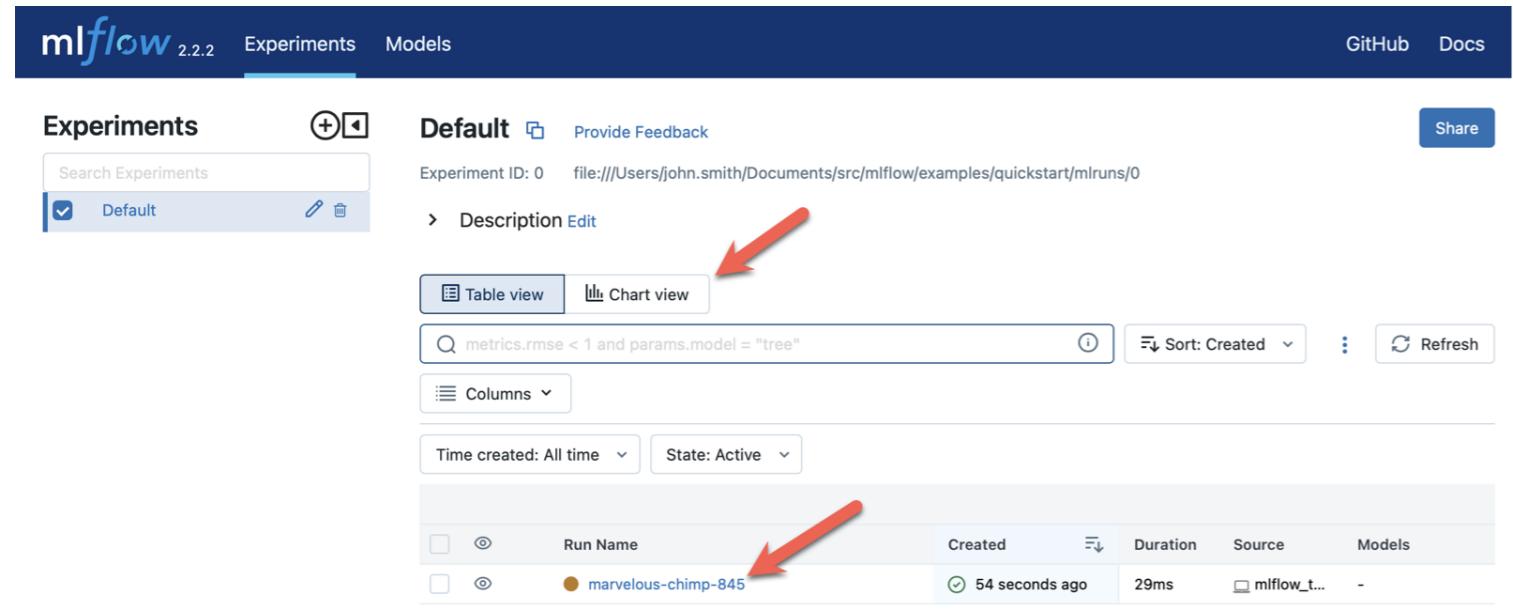
```
# Fetch the run data and print params  
run_data = mlflow.get_run(run_id)  
print(run_data.data.params)  
print(run_data.data.metrics)
```

```
# Search all runs in experiment
```

```
exp_id = run_data.info.experiment_id  
runs_df = mlflow.search_runs(exp_id)
```

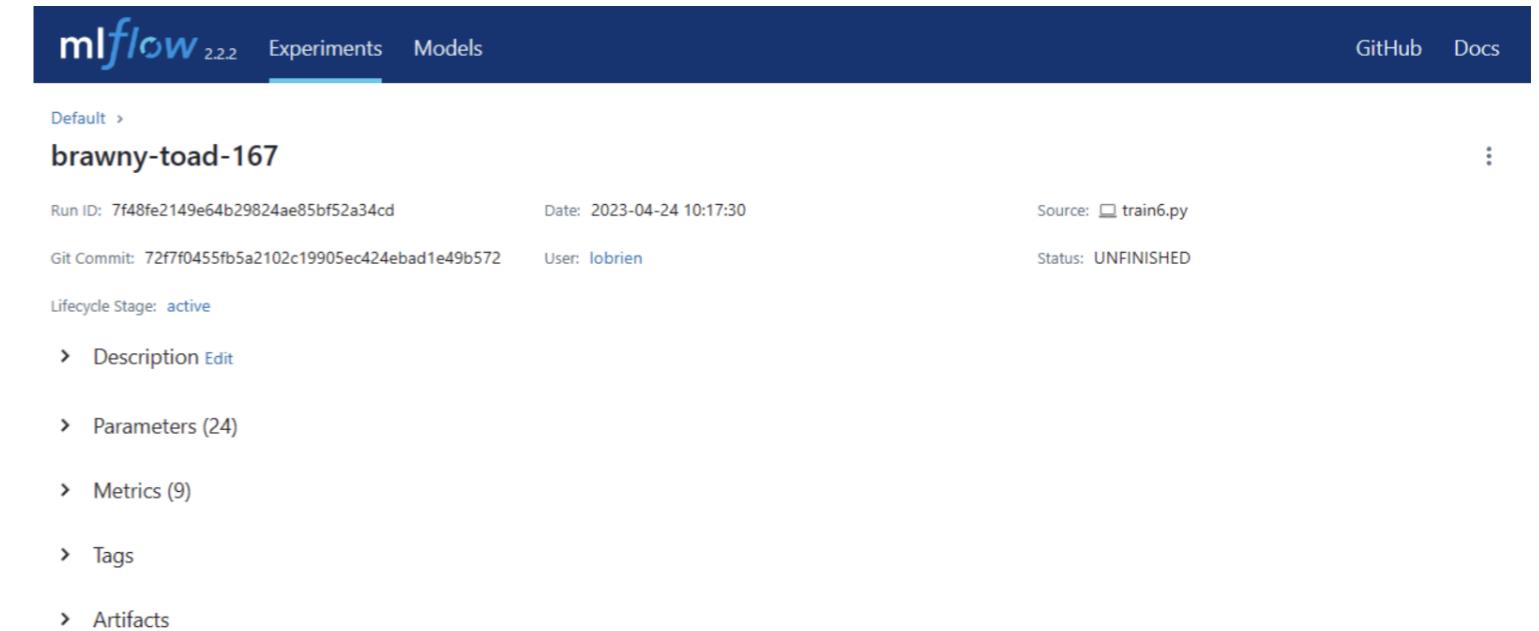
```
{'epochs': '20', 'accuracy': 0.95}
```

MLFlow UI



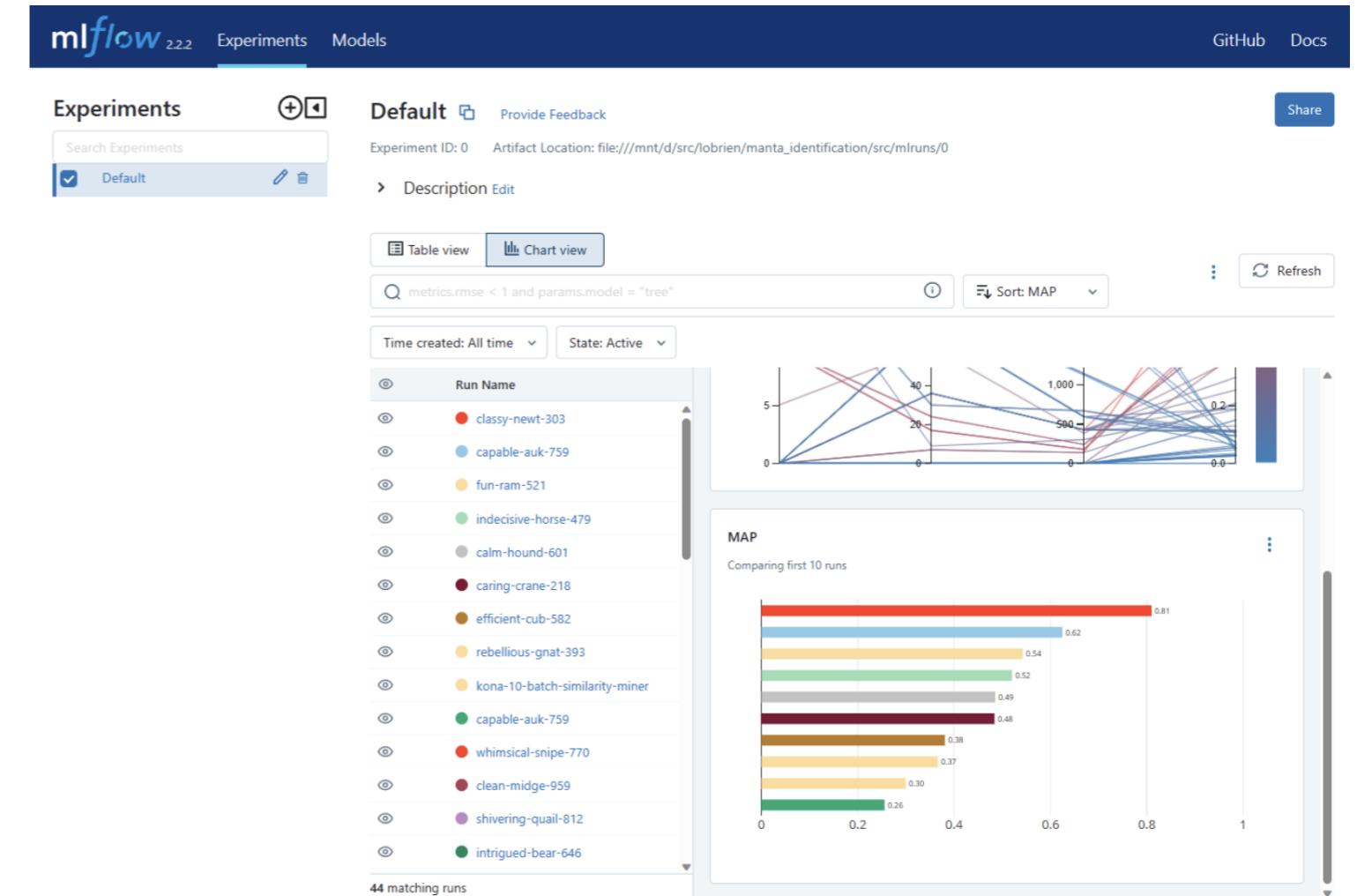
The screenshot shows the MLFlow UI Experiments page. At the top, there's a header with the MLFlow logo, version 2.2.2, and links for GitHub and Docs. Below the header, the title "Default" is shown with a "Provide Feedback" button. A red arrow points from the text "MLFlow UI" in the question to the "Default" title. Underneath, the experiment ID is listed as "Experiment ID: 0 file:///Users/john.smith/Documents/src/mlflow/examples/quickstart/mlruns/0". A "Description Edit" link is present. Below this, there are two view buttons: "Table view" (selected) and "Chart view". A search bar contains the query "metrics.rmse < 1 and params.model = "tree"" with a refresh icon. A "Sort: Created" dropdown and a "Refresh" button are also visible. Further down, there are filters for "Time created: All time" and "State: Active". The main area displays a table of runs:

Run Name	Created	Duration	Source	Models
marvelous-chimp-845	54 seconds ago	29ms	miflow_t...	-



The screenshot shows the MLFlow UI Experiment details page for run "brawny-toad-167". The header includes the MLFlow logo, version 2.2.2, and GitHub/Docs links. The experiment name "brawny-toad-167" is displayed with a "Default" link. A red arrow points from the text "MLFlow UI" in the question to the experiment name. Below the name, run details are shown: "Run ID: 7f48fe2149e64b29824ae85bf52a34cd", "Date: 2023-04-24 10:17:30", "Source: train6.py", "Git Commit: 72f7f0455fb5a2102c19905ec424ebad1e49b572", "User: lobrien", and "Status: UNFINISHED". A "Lifecycle Stage: active" link is also present. The page is organized into sections: "Description Edit", "Parameters (24)", "Metrics (9)", "Tags", and "Artifacts".

MLFlow UI (cont.)



MLflow resources

- Introduction to MLflow

The screenshot shows a DataCamp course page for 'Introduction to MLflow'. At the top, there's a green 'Start Course' button and a 'Bookmark' button. Below that, course statistics are listed: 4 hours, 16 Videos, 51 Exercises, 413 Participants, and 3,750 XP. A 'Course Description' section explains the complexity of managing ML lifecycles and how MLflow simplifies it. It includes a 'Read More' link. A chapter preview for 'Introduction to MLflow' is shown, featuring a photo of Weston Bassler, a Senior MLOps Engineer, and a brief description of the chapter's content. On the right, there's a sidebar with a 'DATASETS' section containing links to '50_Startups', 'Student_Study_Hour', 'Salary_predict', and 'Insurance', with a 'Explore datasets' button.

- MLflow's official website

The screenshot shows the official MLflow website. The header features a dark blue background with a blue digital wave graphic. Below the header, the text 'An open source platform for the machine learning lifecycle' is displayed. To the right, there's a 'Latest News' sidebar with links to recent releases: 'MLflow 2.5.0 released!' (17 Jul 2023), 'MLflow 2.4.0 released!' (06 Jun 2023), 'MLflow 2.3.2 released!' (12 May 2023), 'MLflow 2.3.1 released!' (28 Apr 2023), and 'MLflow 2.3.0 released!' (18 Apr 2023). A 'News Archive' link is also present. At the bottom, there are four icons with text: '</>' (WORKS WITH ANY ML LIBRARY, LANGUAGE & EXISTING CODE), a checkmark in a cloud (RUNS THE SAME WAY IN ANY CLOUD), a group of people (DESIGNED TO SCALE FROM 1 USER TO LARGE ORGS), and the Apache Spark logo (SCALES TO BIG DATA WITH APACHE SPARK™).

Let's practice!

END-TO-END MACHINE LEARNING

Model evaluation and visualization

END-TO-END MACHINE LEARNING



Joshua Stapleton
Machine Learning Engineer

Accuracy

- Correct accuracy metrics are vital to robust model evaluation
- Easy to misinterpret or obscure results

Standard accuracy:

- Standard accuracy = num correct answers / num answers
- Standard accuracy can be unhelpful

Example:

```
# achieves ~99% accuracy for imbalanced dataset of 99 positive and 1 negative
for patient_datapoint in heart_disease_dataset:
    model.prediction(patient_datapoint) = 'positive'
```

Confusion matrix

True positives (TP)

- Model prediction = actual classification = positive
- The model predicted heart disease, the patient had heart disease

False negatives (FN)

- Model prediction = negative, actual classification = positive
- The model predicted no heart disease, the patient had heart disease

False positives (FP)

- Model prediction = positive, actual classification = negative
- The model predicted heart disease, the patient did not have heart disease

True negatives (TN)

- Model prediction = actual classification = negative
- The model predicted no heart disease, the patient did not have heart disease

Balanced accuracy

- Better metric than plain accuracy for most binary classification models
- Provides weighted average across both classes
- $\text{Balanced accuracy} = (\text{TP} + \text{TN}) / 2$

```
from sklearn.metrics import balanced_accuracy_score

# Assume y_test is the true labels and y_pred are the predicted labels
y_pred = model.predict(X_test)
bal_accuracy = balanced_accuracy_score(y_test, y_pred)
print(f"Balanced Accuracy: {bal_accuracy:.2f}")
```

Balanced Accuracy: 0.85

Confusion matrix usage

	Actual: Heart disease	Actual: No heart disease
Predicted: Heart disease	TP: 20	FP: 5
Predicted: No heart disease	FN: 3	TN: 24

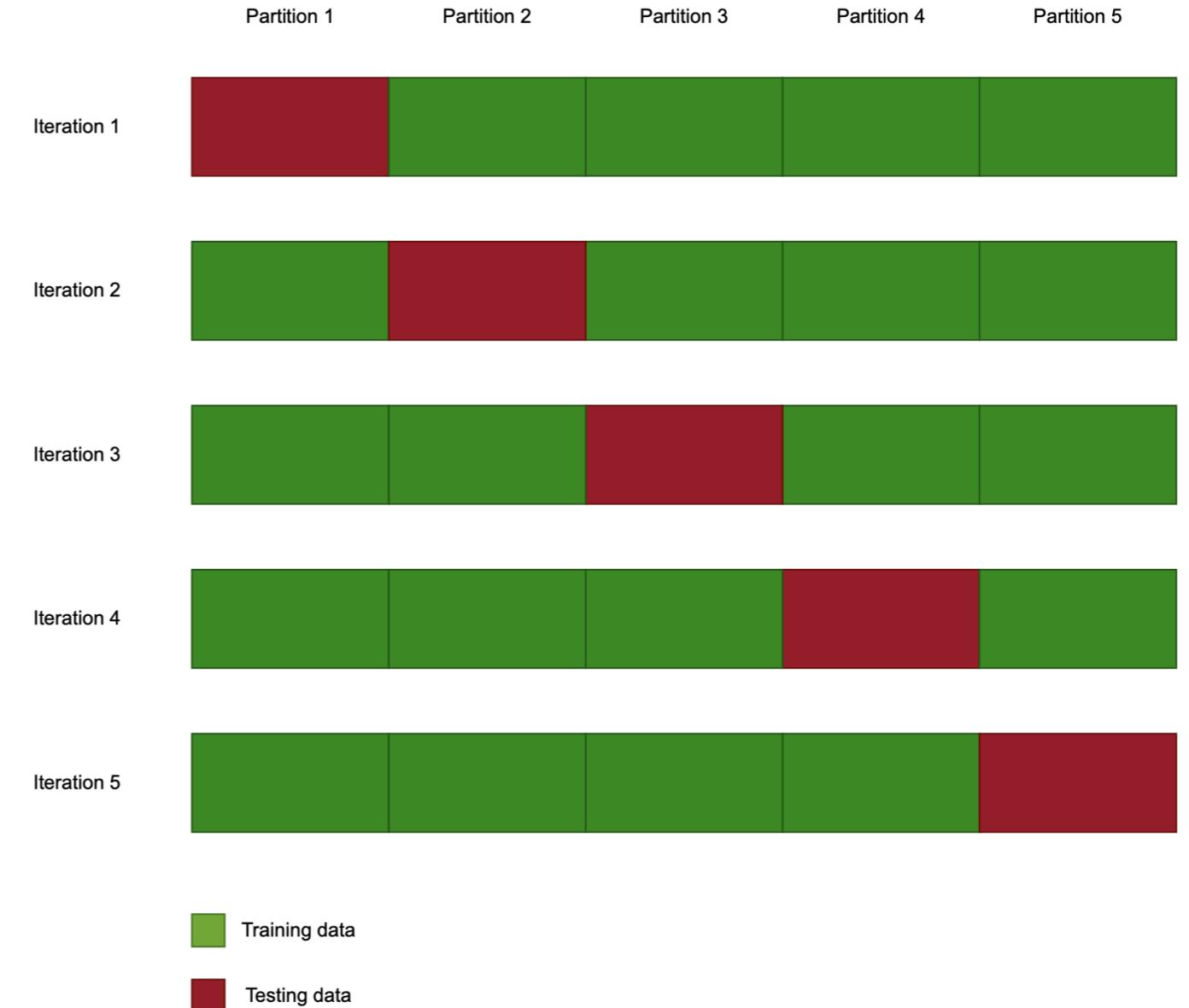
Cross validation

Cross-validation

- Resampling procedure
- Ensures robustness of results

k-fold cross-validation

- Param 'k' = number of splits for dataset
- Resample new train/test split for each modeling run



Cross validation usage

- Straightforward implementation of k-fold cross validation using sklearn
- Model-agnostic scoring

Usage:

```
from sklearn.model_selection import cross_val_score, KFold

# split the data into 10 equal parts
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# get the cross validation accuracy for a given model
cv_results = cross_val_score(model, heart_disease_X,
heart_disease_y, cv=kfold, scoring='balanced_accuracy')
```

Hyperparameter tuning

Hyperparameter:

- Global model parameter (doesn't change during training)
- Adjust to improve model performance

```
# Hyperparameters to test
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Manually iterate over the hyperparameters
for C in C_values:
    model = LogisticRegression(max_iter=200, C=C)
    model.fit(X_train, y_train)
    accuracy = cross_val_score(model, X, y, cv=kfold, scoring='balanced_accuracy')
    print(f"C = {C}: Bal Acc: {accuracy.mean():.4f} (+/- {accuracy.std():.4f})")
```

Hyperparameter tuning example

Example output for hyperparameter tuning:

```
C = 0.001: Bal Acc: 0.6200 (+/- 0.0215)
C = 0.01: Bal Acc: 0.7325 (+/- 0.0234)
C = 0.1: Bal Acc: 0.7923 (+/- 0.0202)
C = 1: Bal Acc: 0.8050 (+/- 0.0191)
C = 10: Bal Acc: 0.8034 (+/- 0.0185)
C = 100: Bal Acc: 0.8021 (+/- 0.0187)
C = 1000: Bal Acc: 0.8017 (+/- 0.0188)
```

Let's practice!

END-TO-END MACHINE LEARNING