Módulo 2: Ensuring Reproducibility

Designing reproducible experiments

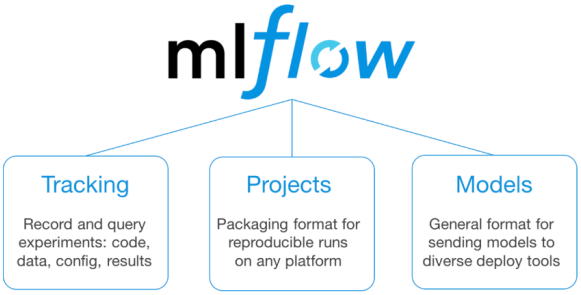
Reproducible experiments

La reproducibilidad en ML es esencial para construir resultados confiables y precisos en los modelos. Esta nos ayuda a replicar los resultados, además de mejorar la colaboración con otras personas. Por otro lado, ayuda a reducir el riesgo de sesgos en los modelos de ML y asegura la integridad del proceso y sus resultados.

MLflow

An open-source platform for tracking and managing machine learning experiments. MLflow can be used to:

- Create reproducible ML pipelines
- Track and manage:
  - package dependencies
  - code versions
  - experiment settings
- Allows multiple users to access experiments



Example of using Mlflow

```
# standard scikit-learn imports
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# new imports from MLflow
import mlflow
import mlflow.sklearn
```

```
with mlflow.start_run():  # Start an MLflow run assuming we have data prepared

    # Build and train model
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)

    # Log parameters and model information
    mlflow.log_param("n_estimators", rf.n_estimators)
    mlflow.sklearn.log_model(rf, "model")

    y_pred = rf.predict(X_test)  # Evaluate model
    accuracy = accuracy_score(y_test, y_pred)
    mlflow.log_metric("accuracy", accuracy)  # log the test accuracy metric
```

Tracking code

- Logging code versions and changes with MLflow
- Comparing different versions of code
- Identifying which version of the code was used to produce a given set of results
- Easily reproducing experiments
- Making it easy to debug and troubleshoot code

Model registries

Son repositorios centralizados de modelos y sus metadatos. MLFlow se puede usar para llevar este registro de los modelos y comparar diferentes versiones, reproduciendo todo el pipeline.

Experiment reproducibility

MLFlow también puede usarse para asegurar la reproducibilidad de los experimentos, llevando un seguimiento de los datos de entrada, el código y las configuraciones.

Revisiting documentation

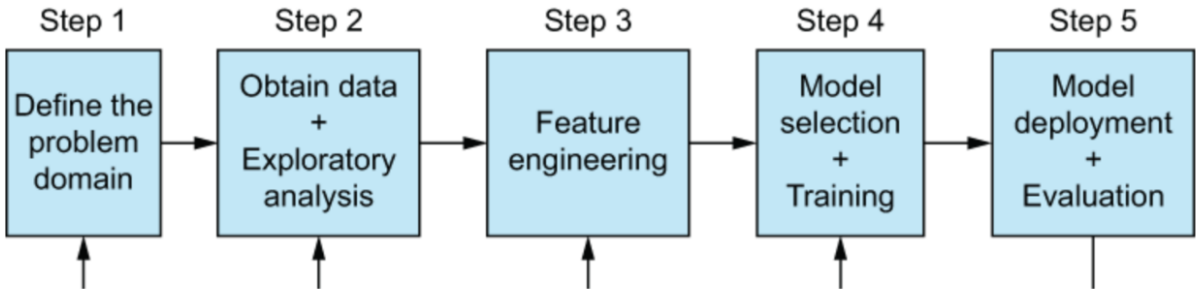
La documentación es esencial para la reproducibilidad de los procesos de ML. Una documentación adecuada debe incluir de forma clara y detallada información sobre los datos de entrada, el código, las configuraciones y los resultados de los experimentos. Además, esta debe ser accesible para el resto de personas.

Feature Engineering

Introduction to feature engineering

Feature engineering es el proceso de transformar los datos de entrenamiento para maximizar el desempeño del pipeline de ML y reducir la complejidad computacional.

- Agregar nuevos datos de múltiples fuentes
- Construir nuevas variables
- Aplicar transformaciones a las variables



Aggregating data from multiple sources

Agregar datos de múltiples fuentes es un aspecto importante del Feature Engineering. Podemos combinar datos de diferentes conjuntos e incluir múltiples

tipos de datos en nuestro conjunto de entrenamiento; esto puede ayudar a mejorar el rendimiento del modelo.

```
class DataAggregator:
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self  # nothing to fit

    def transform(self, X, y=None):
        # Load data from multiple sources
        data1 = pd.read_csv('data1.csv')
        data2 = pd.read_csv('data2.csv')
        data3 = pd.read_csv('data3.csv')

        # Combine data from all sources (including X) into a single data frame
        aggregated_data = pd.concat([X, data1, data2, data3], axis=0)

        return aggregated_data  # Return aggregated data
```

Feature construction

La construcción de características es el proceso de combinar variables existentes para crear nuevas o generar nuevas variables a partir de datos existentes. Puede mejorar el desempeño del modelo y su interpretabilidad.

```python
class FeatureConstructor:
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        # Calculate the mean of each column in the data
        mean_values = X.mean()

        # Create new features based on the mean values
        X['mean_col1'] = X['col1'] - mean_values['col1']
        X['mean_col2'] = X['col2'] - mean_values['col2']

        return X  # Return the augmented data set
```

Feature transformations

Es el proceso de transformar variables existentes en unas más adecuadas o usables por el modelo. Normalizar los datos, eliminar los outliers.

Transforming existing features in place

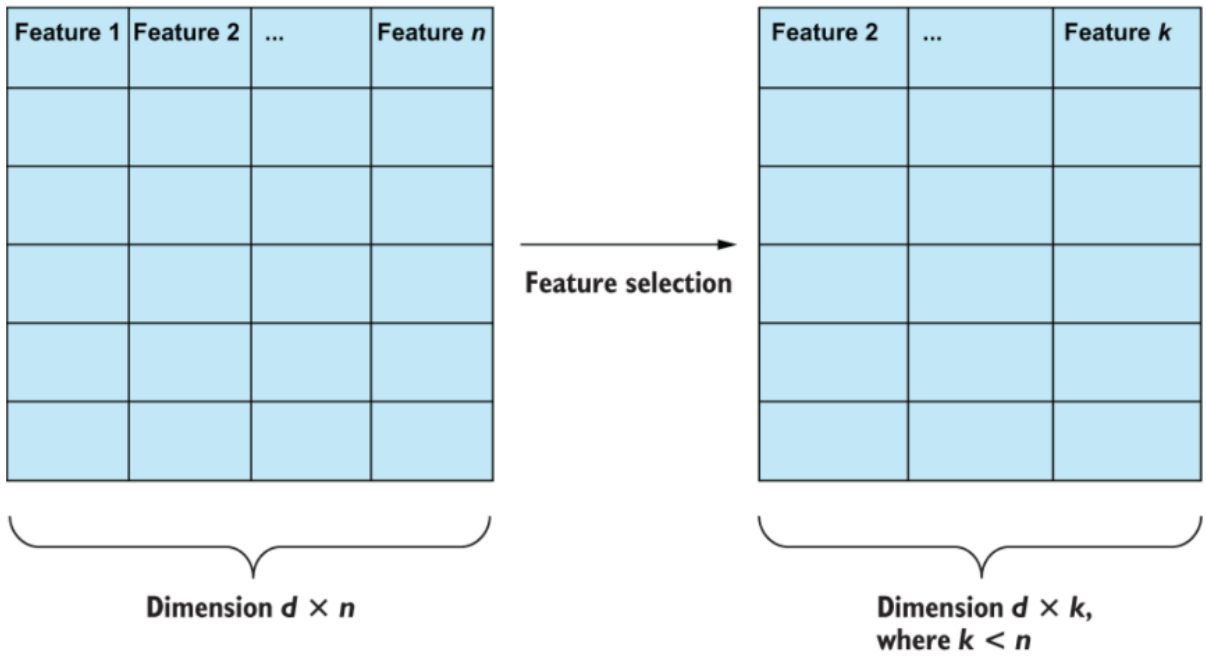
- Normalizing data distributions
- Removing outliers
- Improving model accuracy and performance

```python
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = load_breast_cancer()  # Load dataset
X, y = data.data, data.target
```

```python
model = LogisticRegression(random_state=42)  # instantiate a mode

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.3, random_state=42)

model.fit(X_train, y_train)  # Fit logistic regression model with
y_pred_no_scaling = model.predict(X_test)
acc_no_scaling = accuracy_score(y_test, y_pred_no_scaling)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model.fit(X_train_scaled, y_train)  # Fit logistic regression mod
y_pred_with_scaling = model.predict(X_test_scaled)
acc_with_scaling = accuracy_score(y_test, y_pred_with_scaling)

# Compare accuracies of models with and without scaling
print(f"Accuracy without scaling: {acc_no_scaling}")  # 0.971
print(f"Accuracy with scaling: {acc_with_scaling}")  # 0.982
```

Feature selection

La selección de variables se da cuando elegimos un subconjunto de características de uno más grande, removiendo variables redundantes o irrelevantes. Esto ayuda a reducir el overfitting, mejorar el desempeño del modelo y también la interpretabilidad.



```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.pipeline import Pipeline

pipeline = Pipeline([  # Define feature engineering pipeline

    ('aggregate', DataAggregator()),  # Aggregate data from multiple sources

    ('construction', FeatureConstructor()),  # Feature Construction

    ('scaler', StandardScaler()),  # Feature Transformation

    ('select', SelectKBest(chi2, k=10)),  # Feature Selection
])
X_transformed = pipeline.fit_transform(X)  # Fit and transform data using pipeline
```

Data and model versioning

En ML es importante llevar un seguimiento de las diferentes versiones de los datos y modelos usados en los experimentos. Esto es el versionado.

A través de versionar los datos y los modelos podemos asegurar la reproducibilidad y la rastreabilidad de los experimentos. Es decir, podemos volver

fácilmente a una versión anterior de los datos o el modelo y ver cuánto ha progresado el experimento a través del tiempo.

Major and minor versioning

Major versioning	Minor versioning
Indica cambios grandes en los datos o el modelo, como una nueva variable	Indica cambios pequeños, como la corrección de errores

Versioning training data

Para hacer un versionado de los datos de entrenamiento podemos usar etiquetas como major/minor o fechas únicas para identificar las diferentes versiones de los datos. De esta forma, aseguramos la reproducibilidad y rastreabilidad de los datos.

1. Data V 1.0 was our initial dataset
2. Data V 1.1 included additional feature transformations
3. Data V 1.2 added a feature selection method
4. Data V 2.0 includes a brand new source of data

Feature stores

Son repositorios centrales para almacenar y gestionar diferentes versiones de las variables; estos llevan un seguimiento de las versiones de las variables, lo que mejora la colaboración y reducen el trabajo duplicado.

Versioning ML models

Al igual que con los datos, también podemos versionar los modelos de ML para asegurar la reproducibilidad, llevando un seguimiento de las diferentes versiones de los modelos.

1. Model V 1.0 is our initial model (Random Forest)
2. Model V 1.1 is the same model fine-tuned on Data V 1.1
3. **Model V 2.0** is now a XGBoost Model fine-tuned **Data V 1.2**
4. Model V 2.1 is the XGBoost model fine-tuned on Data V 2.0

Model stores

Al igual que con las variables, existen repositorios centrales para almacenar y gestionar las diferentes versiones de los modelos, permitiendo llevar un seguimiento y asegurando la reproducibilidad de estos.

```
import mlflow

# Start a new mlflow run
with mlflow.start_run() as run:
    # Log model version as a parameter
    mlflow.log_param("model_version", "1.0")

    # Train and save the model
    model = train_model()
    mlflow.sklearn.log_model(model, "model")
```