Módulo 4: Testing ML Pipelines

Model Reliability

Aligning ML models with business impact metrics

Las métricas de impacto del negocio miden el impacto del modelo ML en el negocio, por ejemplo, utilidades, ahorros en costos, satisfacción del cliente.

Las métricas del negocio y el impacto del modelo de ML deben estar alineados.

Testing routines in ML pipelines

- **Unit test:** Prueban los componentes individuales. Ejemplo, probar que una instancia de PCA devuelva el número de variables esperado.
- **Integration test:** Considera todo el pipeline. Ejemplo, probar que los datos de entrada se procesaron correctamente, el modelo hace predicciones certeras y la salida es correctamente procesada.
- **Smoke test:** Son pruebas rápida para dar confianza de que el sistema está funcionando. Ejemplo, probar que el modelo puede clasificar correctamente un pequeño conjunto de imágenes.

Example unit test

```python
def test_pipeline():
    # Generate mock data for testing
    X_train = pd.DataFrame({'age': [25, 30, 35, 40], 'income': [50000, 60000, 70000, 80000]})
    y_train = pd.Series([0, 0, 1, 1])

    pipeline = Pipeline([('preprocessing', DataPreprocessor()),  # Set up pipeline
                         ('model', LogisticRegression())])
    pipeline.fit(X_train, y_train)  # Fit pipeline on training data

    # Generate mock data for testing
    X_test = pd.DataFrame({'age': [30, 35, 40, 45], 'income': [55000, 65000, 75000, 85000]})
    y_test = pd.Series([0, 0, 1, 1])
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)  # Evaluate pipeline on test data

    assert accuracy > 0.8, "Error: pipeline accuracy is too low."
```

Monitoring model staleness

EL desempeño del modelo disminuye con el tiempo por cambios en los datos o en el contexto, conocidos como Data o Concept drift.

Este estancamiento se puede identificar monitoreando el desempeño del modelo y los cambios en los datos y el entorno en el cual funciona el modelo.

Algunas formas de solucionar el estancamiento son: reentrenar el modelo con los nuevos datos o actualizar el pipeline para que considere los cambios en los datos o el contexto.

Testing data

Data validation and schema tests

- **Data validation tests:** Se usan para asegurar que los datos utilizados en el pipeline de ML son precisos y consistentes. Se buscan valores faltantes, inconsistencias y datos anormales.
- **Schema tests:** Se usan para asegurar que los datos están siendo usados en el formato correcto y responden a ciertos criterios. Se busca que el formato y los tipos de los datos sean los esperados.
- Herramientas como **Great Expectations** ayudan a automatizar este proceso.

Beyond simple testing

Las pruebas de datos y esquemas validan problemas básicos. Las pruebas de **Expectativas** buscan evaluar problemas más complejos, como que los valores estén cierto rango.

Expectation tests

Son un tipo de prueba de validación de los datos.

Buscan asegurar que los datos se comporten o presenten ciertos criterios o “expectativas” definidas por el usuario. Ejemplo, se espera que el tiempo que pasan los usuarios en el sitio web sea de medio 4 minutos con desviación estándar de 1 minuto.

Feature importance tests

Son pruebas para identificar cuáles variables son más importantes para el modelo de ML a la hora de hacer la predicción.

Un ejemplo sería una prueba de Permutación de Importancia, donde aleatoriamente se permutan variables y se mide cuánto disminuye el desempeño del modelo.

Example of permutation importance

Set up:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
# Train a random forest classifier (assuming we have some data)
model = RandomForestClassifier().fit(X_train, y_train)
```

Running our permutation importance test:

```
# Calculate feature importances using permutation importance
results = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42)
# Print the feature importances
feature_names = ['feature_1', 'feature_2', 'feature_3', ...]
importances = results.importances_mean
for i in range(len(feature_names)):
    print(f'{feature_names[i]}: {importances[i]}')
```

Looking for data drift

Data drift: También conocido como Feature Drift, se refiere a un cambio en la distribución de los datos de entrada del modelo.

Label drift: Se refiere a un cambio en la distribución de la variable objetivo.

Testing models

Invididual fairness

El modelo debe tratar de forma similar a individuos con características similares. Ejemplo: se deben dar las mimas oportunidades laborales a personas con experiencia similar.

Group fairness

Diferentes grupos deberían ser tratados de forma equitativa, es decir, el modelo debe arrojar predicciones similares a individuos en diferentes grupos como raza o género.

Holdout testing

Es el proceso de probar el modelo con un conjunto de datos separado que nunca se usó durante el entrenamiento. Es usado para probar la confiabilidad y la capacidad de generalizar del modelo. Con este proceso es posible identificar el overfitting o el underfitting.

Looking for model drift

Concept drift

- A shift in the relationship between the features and the response
  - E.g. a sentiment analysis algorithm
    - Terms can shift in meaning (saying something is "sick" is a good thing)

Prediction drift

- A shift in a model's prediction distribution
  - E.g. a temporary outage leads to an increase to the "outage" intent for your chatbot
  - Nothing is "wrong" per say but this drift could lead to slower response times for users

Example of looking for model drift

Setup:

```
# Import necessary libraries and split data
X_train, X_test, y_train, y_test = ...
```

Fitting our model:

```
# Train a classifier on the training data
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Calculate the accuracy on test data
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Testing for drift later:

```
X_test_drift = X_test + 1.0  # Simulate data drift
# Calculate the accuracy on drifted data
y_pred_drift = clf.predict(X_test_drift)
accuracy_drift = accuracy_score(
    y_test, y_pred_drift)
print(f"Accuracy with drift: {accuracy_drift}")

# Drift detection threshold based on the accuracy
drift_threshold = accuracy * 0.9

# Check for drop in accuracy on the drifted data
if accuracy_drift < drift_threshold:
    print("Concept drift detected!")
else:
    print("No concept drift detected.")
```

Cost of complex models vs baseline models

Modelos complejos son más caros y pueden afectar a la eficiencia del mismo.

La **latencia** mide el tiempo que toma procesar una única entrada petición y generar una predicción.

Throughput mide el número de predicciones que el modelo puede realizar por unidad de tiempo.

Probando la latencia y el throughput podemos identificar qué tanta complejidad debería tener un modelo para seguir manteniendo un rendimiento adecuado.

Es necesario alcanzar un equilibrio entre la precisión y la eficiencia.