Lesson: Introduction to XGBoost with Regression Metrics

Session Overview

In this lesson, we will learn about **XGBoost**, a powerful machine learning algorithm, and evaluate its performance using different **error metrics** for regression.

What is XGBoost?

XGBoost (**Extreme Gradient Boosting**) is an optimized machine learning algorithm based on **decision trees**. It is widely used in industry and **Kaggle competitions** because of its **speed and high accuracy**.

💡 **Key Features of XGBoost**:
✓ Fast and efficient training.
✓ Handles missing data automatically.
✓ Provides feature importance insights.

XGBoost Implementation (Step by Step)

We'll implement **XGBoost Regressor** on a dataset and evaluate the model using multiple regression metrics.

Step 1: Install & Import Libraries

```
# Install XGBoost if not installed
!pip install xgboost

# Import required libraries
import pandas as pd
import numpy as np
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,
mean_absolute_percentage_error
from sklearn.preprocessing import StandardScaler
```

Step 2: Load and Explore the Dataset

```
# Load dataset
file_path = "/mnt/data/health_workforce_indicators_kaz.csv"
df = pd.read_csv(file_path)

# Display first few rows
print(df.head())

# Check for missing values
print(df.isnull().sum())
```

Step 3: Data Preprocessing

```
# Drop missing values (if any)
df.dropna(inplace=True)

# Separate features (X) and target variable (y)
X = df.iloc[:, :-1]  # All columns except the last one (features)
y = df.iloc[:, -1]   # The last column (target)

# Normalize data (optional)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Print dataset shapes
print(f"Training set: {X_train.shape}, Testing set: {X_test.shape}")
```

Step 4: Train an XGBoost Model

```
# Create the XGBoost model
xgb_model = xgb.XGBRegressor(
    objective="reg:squarederror",  # Use squared error for regression
    n_estimators=100,  # Number of trees
    learning_rate=0.1,  # Step size for tree updates
    max_depth=5,  # Tree depth
    subsample=0.8,  # Use 80% of data per tree
    colsample_bytree=0.8,  # Use 80% of features per tree
    random_state=42
)

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

Step 5: Make Predictions

```
# Predict on test data
y_pred = xgb_model.predict(X_test)

# Display first 5 predictions
print("Predicted values:", y_pred[:5])
print("Actual values:", y_test[:5].values)
```

Model Evaluation: Understanding Regression Metrics

Now, let's evaluate our model using multiple **error metrics**.

(A) Mean Squared Error (MSE)

- MSE measures the **average squared difference** between actual and predicted values.
- A **lower MSE** means better performance.
- **Drawback:** Since it's squared, larger errors are weighted more.

```
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

(B) Root Mean Squared Error (RMSE)

- RMSE is just the **square root of MSE**.
- It is in the **same unit** as the target variable, making it more interpretable.

```
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

(C) Mean Absolute Error (MAE)

- Measures the **average absolute difference** between actual and predicted values.
- Unlike MSE, MAE is **not sensitive to outliers**.

```
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

(D) Mean Absolute Percentage Error (MAPE)

- Expresses error as a **percentage**, which makes it easier to interpret.
- **Lower MAPE = better model**.

```
mape = mean_absolute_percentage_error(y_test, y_pred)
print(f"Mean Absolute Percentage Error (MAPE): {mape * 100:.2f}%")
```

(E) R-Squared (R^2) Score

- Measures **how well** the model explains the variance in the data.
- **$R^2 = 1$** (Perfect fit), **$R^2 = 0$** (No relationship).
- **Higher R^2** means a better model.

```
r2 = r2_score(y_test, y_pred)
print(f"R-Squared (R²) Score: {r2:.4f}")
```

1. Why is RMSE preferred over MSE?

✅ **RMSE (Root Mean Squared Error) is preferred over MSE (Mean Squared Error)** because:

- **Interpretability**: RMSE is in the **same unit** as the target variable, making it easier to understand. MSE, on the other hand, squares the errors, making its unit different.
- **Sensitivity to Large Errors**: RMSE gives **more weight to large errors** due to the square root, making it useful when large deviations need to be minimized.
- **More Realistic Error Representation**: Since RMSE is in the same scale as the target, it directly tells us how much the predictions deviate from actual values on average.

Example:

- If **MSE = 400**, the RMSE would be **√400 = 20**, which means the model's predictions deviate **on average by 20 units** from actual values.

---

2. What does an R^2 score of 0.85 mean?

✅ **An R^2 score (coefficient of determination) of 0.85 means** that **85% of the variation in the target variable is explained by the model's features**.

🔹 **Interpretation:**

- **$R^2 = 1$** → Perfect model (all variance explained).
- **$R^2 = 0.85$** → Model explains **85%** of the variance, meaning it's a **good fit** but not perfect.
- **$R^2 = 0$** → Model does not explain any variance.
- **$R^2 < 0$** → Model is worse than a simple mean prediction.

Example:
Imagine we are predicting house prices. If **$R^2 = 0.85$**, then 85% of the price variations are explained by the features (income, number of rooms, etc.), while the remaining **15%** is due to factors not captured in the model (e.g., location, school ratings).

3. Which metric is best for evaluating models, and why?

The **best metric depends on the problem**:

Metric	Best Use Case	Why?
RMSE (Root Mean Squared Error)	General regression	Penalizes large errors, easy to interpret.
MAE (Mean Absolute Error)	When all errors should be treated equally	Not sensitive to outliers, simple.
R^2 Score	When explaining variance is important	Shows how well the model explains data.
MAPE (Mean Absolute Percentage Error)	When relative percentage error matters	Expresses error in % form (good for business/finance).

📌 **Final Choice:**

- **RMSE is the most commonly used** because it **balances accuracy and interpretability**.
- **R^2 is useful when comparing models** but should not be used alone.
- **MAE is useful when dealing with datasets that contain outliers** (as it is not affected by squaring errors like MSE).

Hyperparameter Tuning for Better Performance

We can use **Grid Search** to find the best hyperparameters.

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}

# Perform grid search
grid_search = GridSearchCV(xgb.XGBRegressor(objective="reg:squarederror"),
param_grid, cv=3)
grid_search.fit(X_train, y_train)

# Print best parameters
print("Best Parameters:", grid_search.best_params_)
```

Feature Importance (Understanding the Model)

```
# Plot feature importance
xgb.plot_importance(xgb_model)
plt.show()
```
