KAGGLE COMPETITION

Thea Boge

Introduction

Dataset

From 56 raw columns → 10 relevant, interpretable features

Textual

- title
- descriptions

Contains seller language that directly expresses product condition (e.g. “nuevo”, “sin uso”, “usado”)

Numerical

- price
- base_price
- sold_quantity
- available quantity

Capture pricing patterns and product demand, which differ between new and used items.

Categorical

- listing_type_id
- buying_mode
- category_id

Describe how items are sold and categorized, adding contextual variation.

Text Cleaning and Preprocessing

Implemented two functions:

- clean_text()
  - lowercased text
  - removed punctuation and special characters
  - normalized accented characters

- preprocess()
  - combine titles and descriptions
  - apply the clean_text function
  - feature engineering for the numeric and categorical variables

```
df["price_ratio"] = df["price"] / (df["base_price"] + 1)
```

```
df["log_price"] = np.log1p(df["price"].fillna(0))
df["log_sold"] = np.log1p(df["sold_quantity"].fillna(0))
```

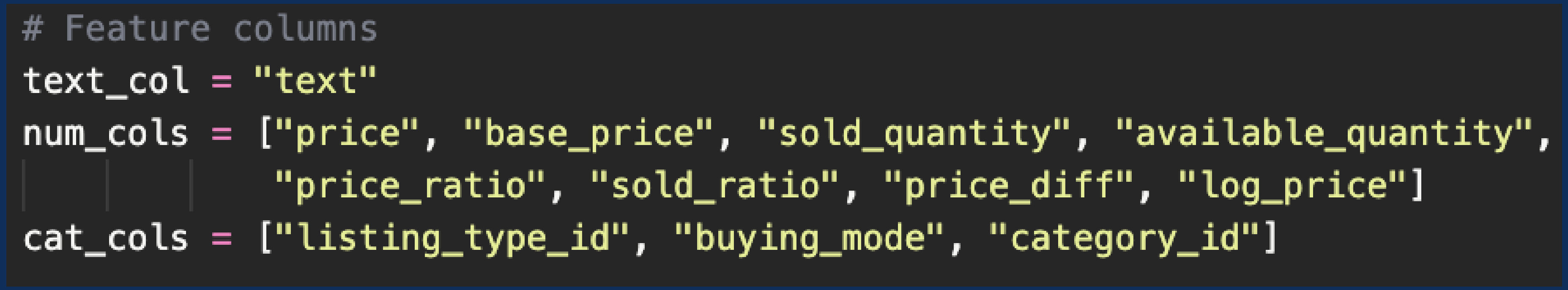
```
df["sold_ratio"] = df["sold_quantity"] / (df["available_quantity"] + 1)
```

Feature groups

Three feature groups for later preprocessing tequniques.

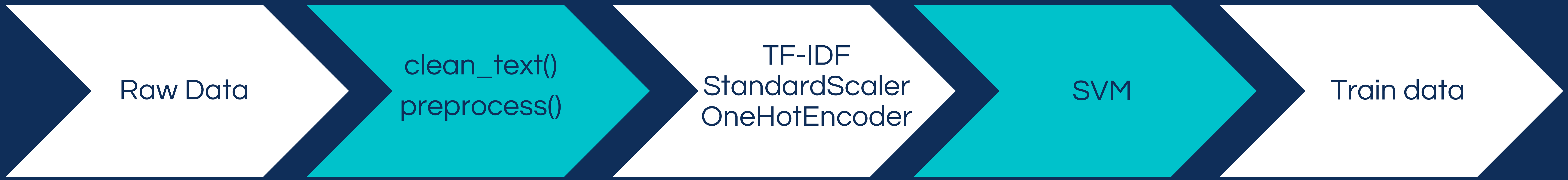
Preprocessing techniques:

- TF-IDF Vectorizer
- StandardScaler
- OneHotEncoder



```python
# Feature columns
text_col = "text"
num_cols = ["price", "base_price", "sold_quantity", "available_quantity",
            "price_ratio", "sold_ratio", "price_diff", "log_price"]
cat_cols = ["listing_type_id", "buying_mode", "category_id"]
```

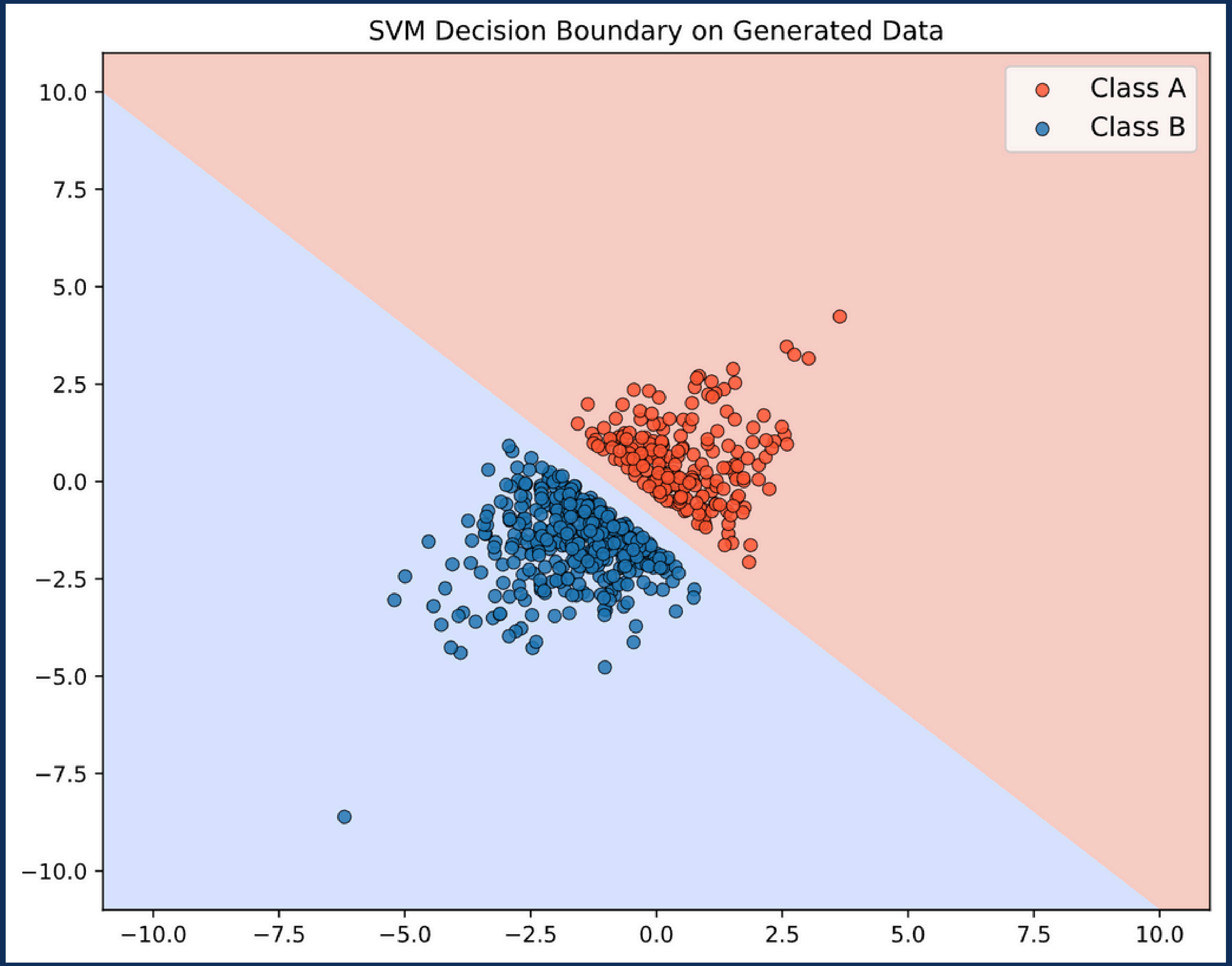
Baseline Model



Support Vector Machines

- Baseline model for comparison
- Uses TF-IDF text features
- Linear SVM: fast, interpretable, effective for high-dimensional sparse data
- Accuracy: 0.8692

- Non-linear kernel (RBF) could model complex patterns, but too heavy for this dataset.

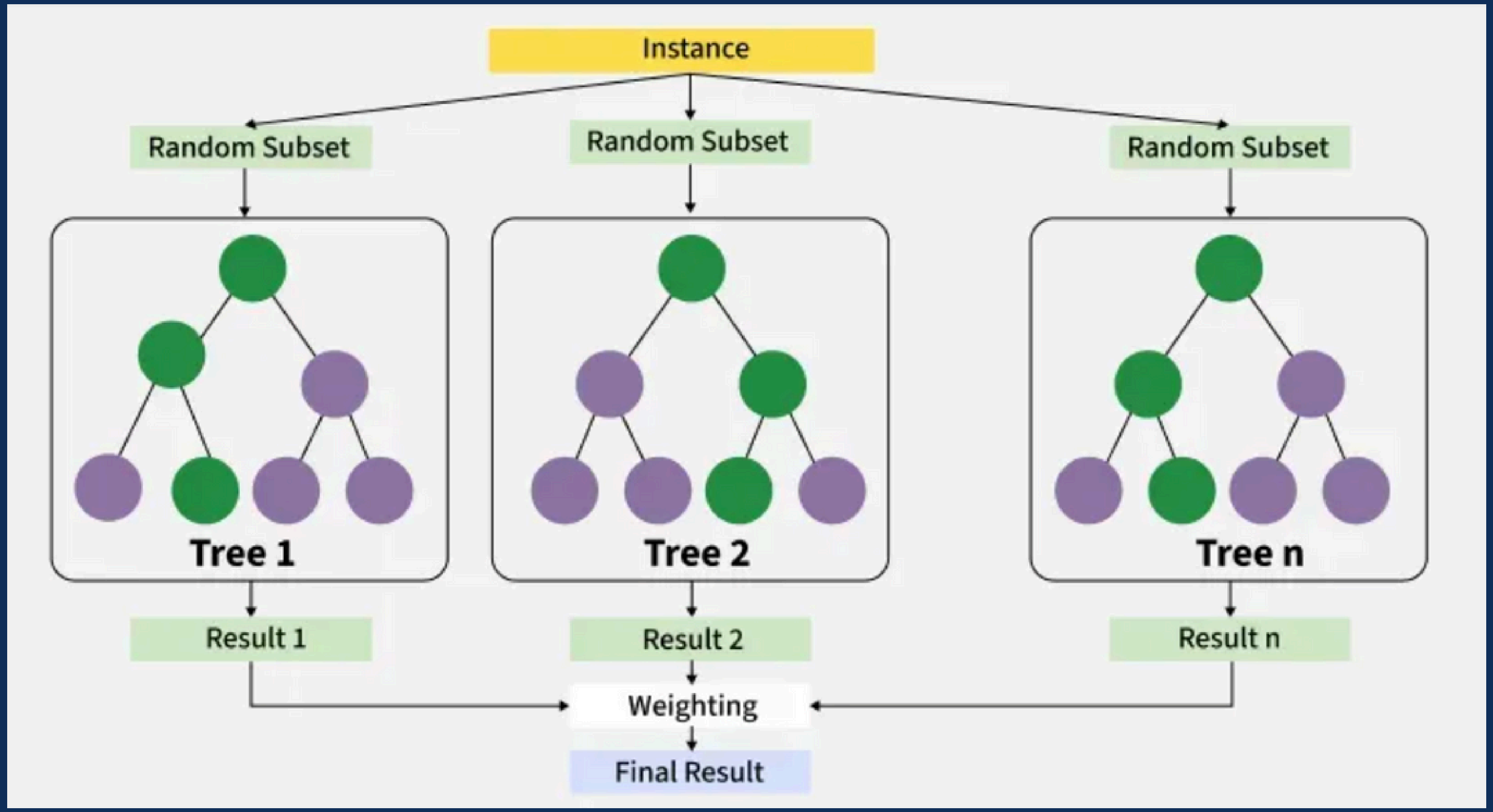


XGBoost

- Gradient boosting algorithm
  - builds trees sequentially
- Corrects previous errors
  - strong performance on mixed features
- Hist tree method: groups continuous features into bins
  - faster and memory-efficient

- Accuracy: 0.8828 (↑ from SVM)

Moving forward → improving the XGBoost

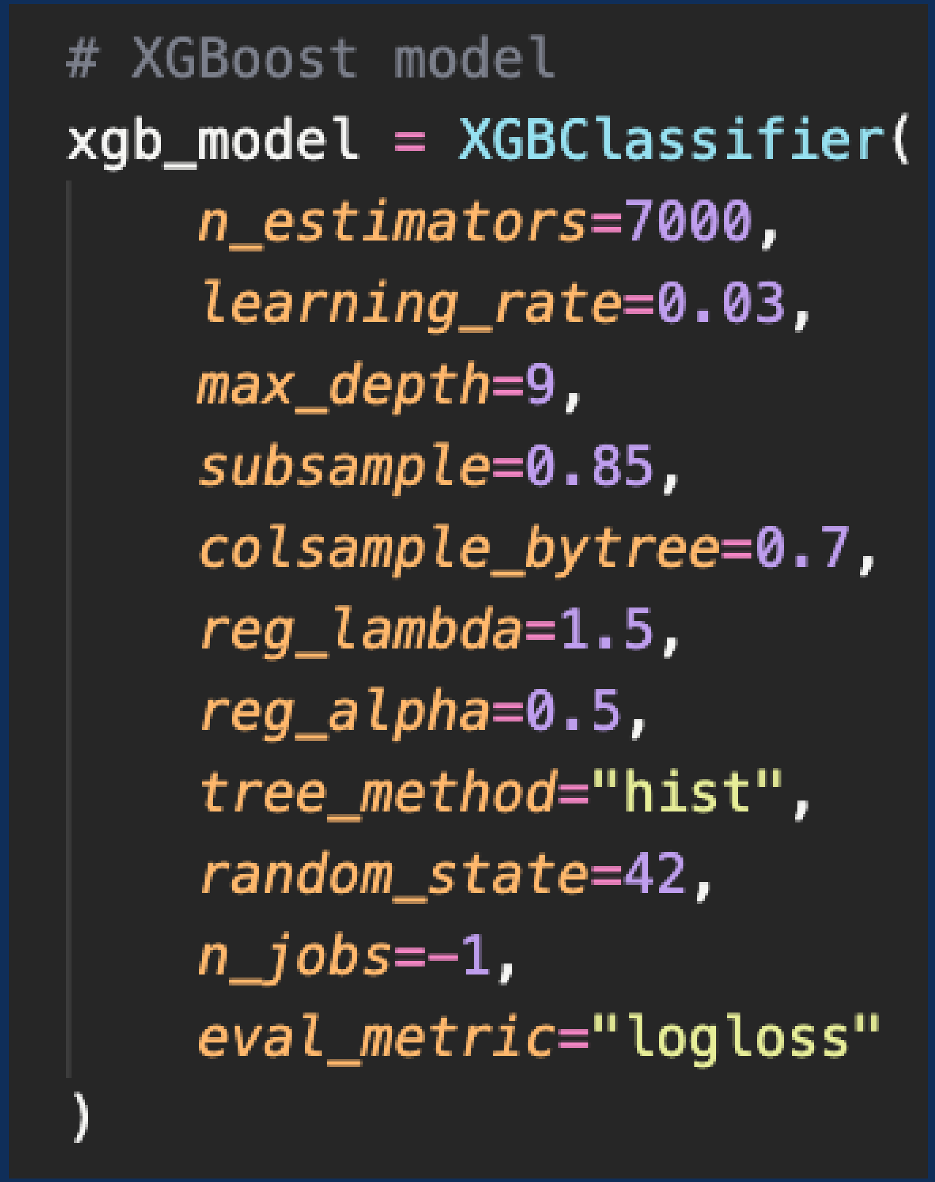


https://www.geeksforgeeks.org/machine-learning/xgboost/

Improvement

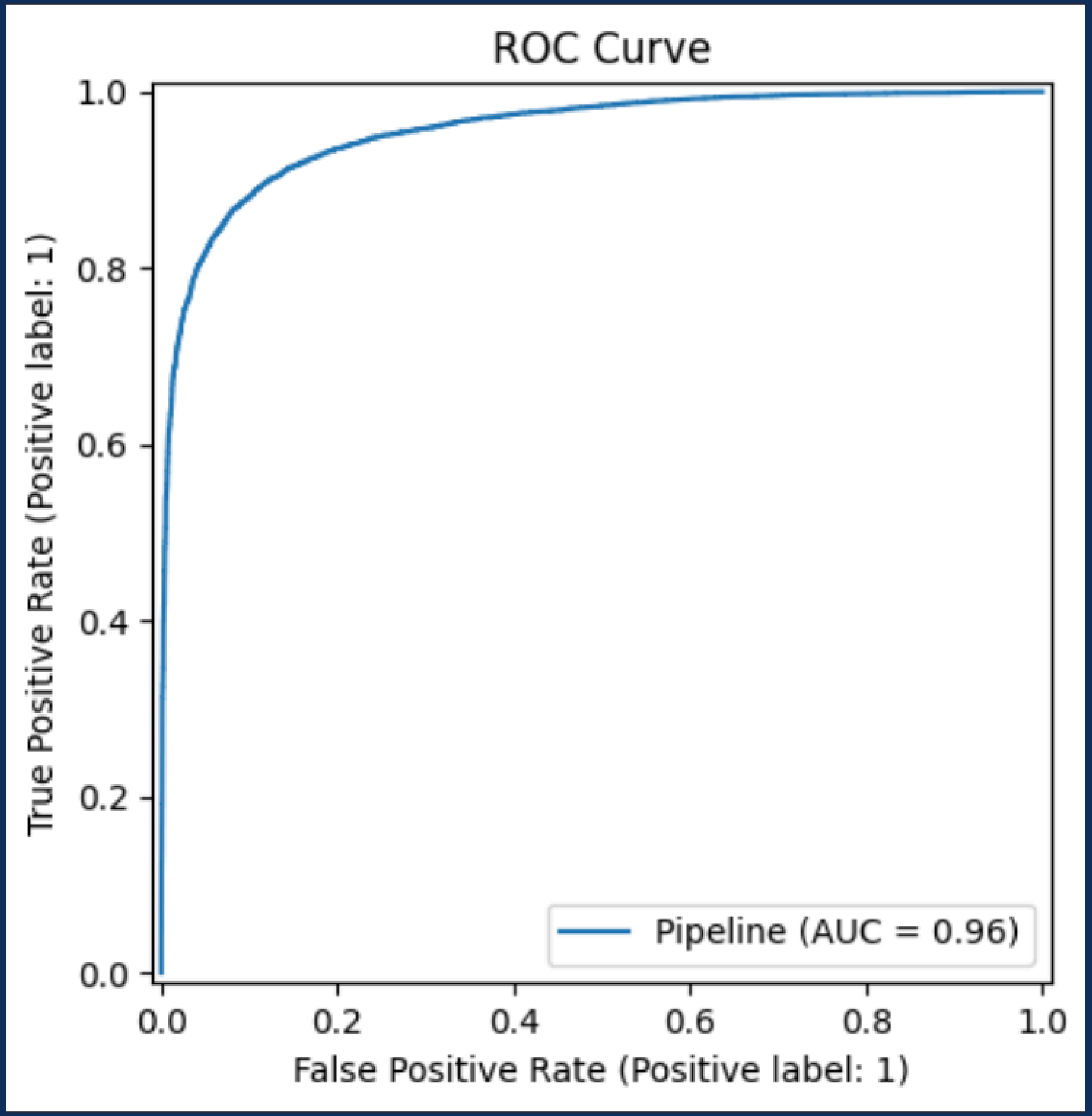
- ColumnTransformer → clean structure
- Added pipeline → end-to-end training
- Tuned hyperparameters
    - learning rate
    - max_depth
    - n_estimators

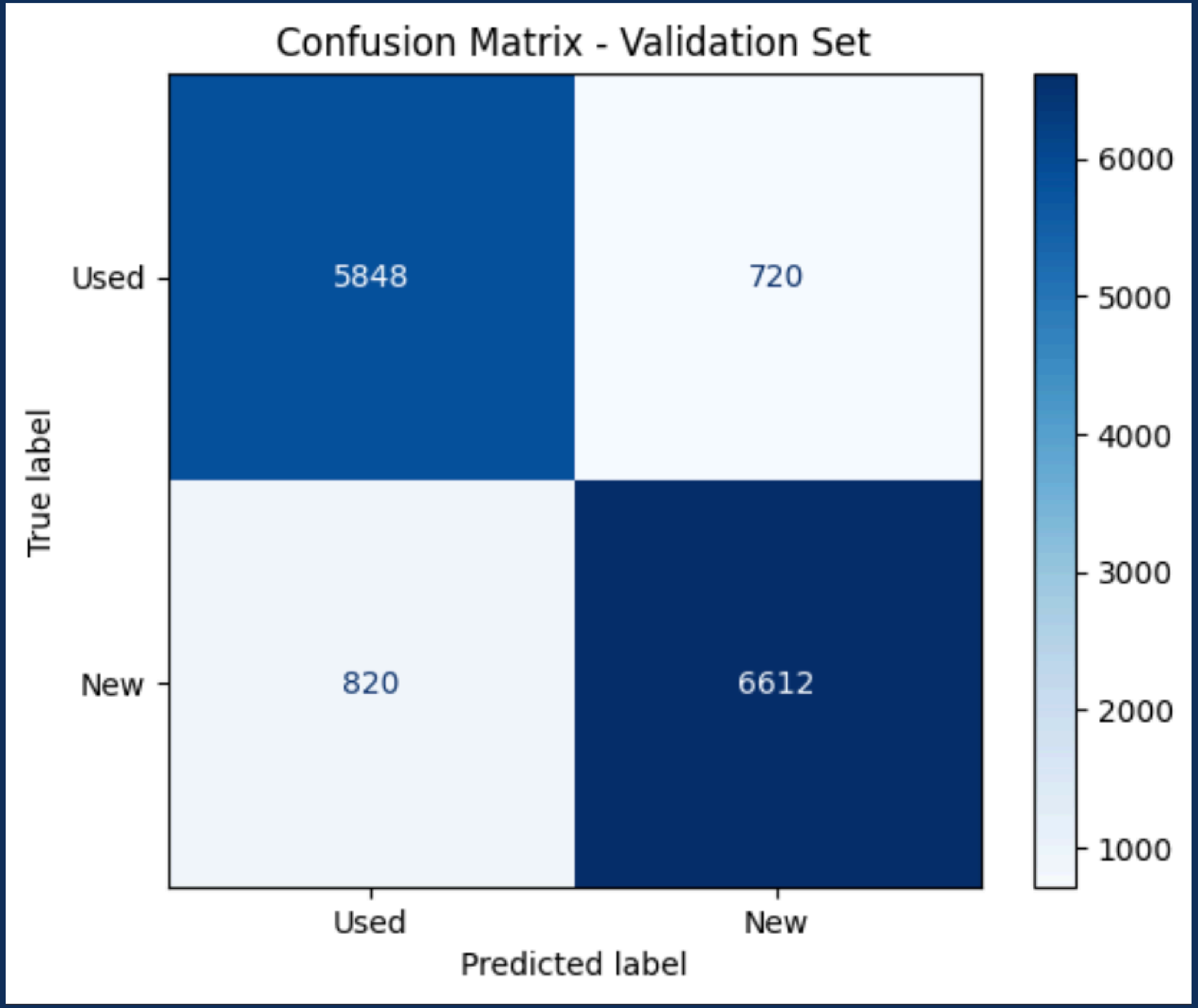
Improved model → 0.89



```
# XGBoost model
xgb_model = XGBClassifier(
    n_estimators=7000,
    learning_rate=0.03,
    max_depth=9,
    subsample=0.85,
    colsample_bytree=0.7,
    reg_lambda=1.5,
    reg_alpha=0.5,
    tree_method="hist",
    random_state=42,
    n_jobs=-1,
    eval_metric="logloss"
)
```

Model Validation





Limitations and further work

- GridSearchCV() or Optuna for model tuning
  - Automate the search for optimal hyperparameters instead of manual tuning.
- Test additional boosting frameworks (e.g., LightGBM, CatBoost)
  - Compare performance and training efficiency on mixed feature data.
- Use richer text embeddings
  - Replace TF-IDF with word embeddings such as FastText or BERT to capture semantic meaning.
- Feature importance and interpretability
  - Analyze the most influential features to better understand what drives the model’s predictions.

Gracias por su atención!