04_ModelComparison

January 29, 2026

1 Model Comparison and Selection

This notebook compares multiple models to select the best-performing one for customer churn prediction.

The question guiding this comparison is:

Does a more flexible model provide meaningful improvement over an interpretable baseline, and is that improvement worth the added complexity? Key evaluation metrics include: - F1-Score - balance between precision and recall - ROC-AUC - overall model quality - Precision at high recall - important for catching most churners - Recall at high precision - important for avoiding wasted interventions

Models compared: - Baseline: Logistic Regression - Random Forest

A Random Forest classifier is used as a non-linear benchmark to evaluate whether interactions and non-linear effects improve churn prediction beyond the linear baseline.

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import RocCurveDisplay, classification_report,␣
 ↪roc_auc_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
import warnings

warnings.filterwarnings("ignore")
```

[ ]:
```
plt.style.use('ggplot')
palette = [
'#E24A33',  #(orange)
'#348ABD',  #(blue)
'#8EBA42',   #(green)
```

```
'#988ED5',  #(cyan)
'#777777',  #(grey)
'#FBC15E',  #(yellow)
'#FFB5B8'    #(pink)
]
```

[ ]:
```
# Load preprocessed data
df = pd.read_csv('../data/ecommerce_churn_data_eda.csv')
```

[ ]:
```
# Same preproccessing steps are applied as in the baseline model
# Define Target & Drop identifier + leakage-prone features
TARGET = 'Churn'
df.drop(columns=['CustomerID', 'DaySinceLastOrder'], inplace=True)

X = df.drop(columns=[TARGET])
y = df[TARGET]

# Identify numeric vs categorical columns
numeric_features = X.select_dtypes(include=["number"]).columns.tolist()
categorical_features = X.select_dtypes(exclude=["number"]).columns.tolist()

if "CityTier" in numeric_features:
    numeric_features.remove("CityTier")
    categorical_features.append("CityTier")

# Preprocessing pipelines
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Unknown')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

[ ]:
```
# Train/validation split
test_split = 0.2
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=test_split,
```

```
    random_state=42,
    stratify=y
)
```

[ ]:
```
# Function to evaluate model

def evaluate_model(model, X_test, y_test, model_name, y_pred_threshold=None):
    results = {}
    model = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("classifier", model)
    ])

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    if y_pred_threshold is not None:
        y_pred = (y_proba >= y_pred_threshold).astype(int)

    report = classification_report(y_test, y_pred, output_dict=True)
    roc_auc = roc_auc_score(y_test, y_proba)

    results[model_name] = {
        "Precision": report["1"]["precision"],
        "Recall": report["1"]["recall"],
        "F1-Score": report["1"]["f1-score"],
        "ROC-AUC": roc_auc
    }

    return results, y_proba
```

[ ]:
```
# Define the models
log_reg_clf = LogisticRegression(
    max_iter=1000,
    class_weight="balanced",
    random_state=42
)

random_forest_clf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    min_samples_leaf=10,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
```

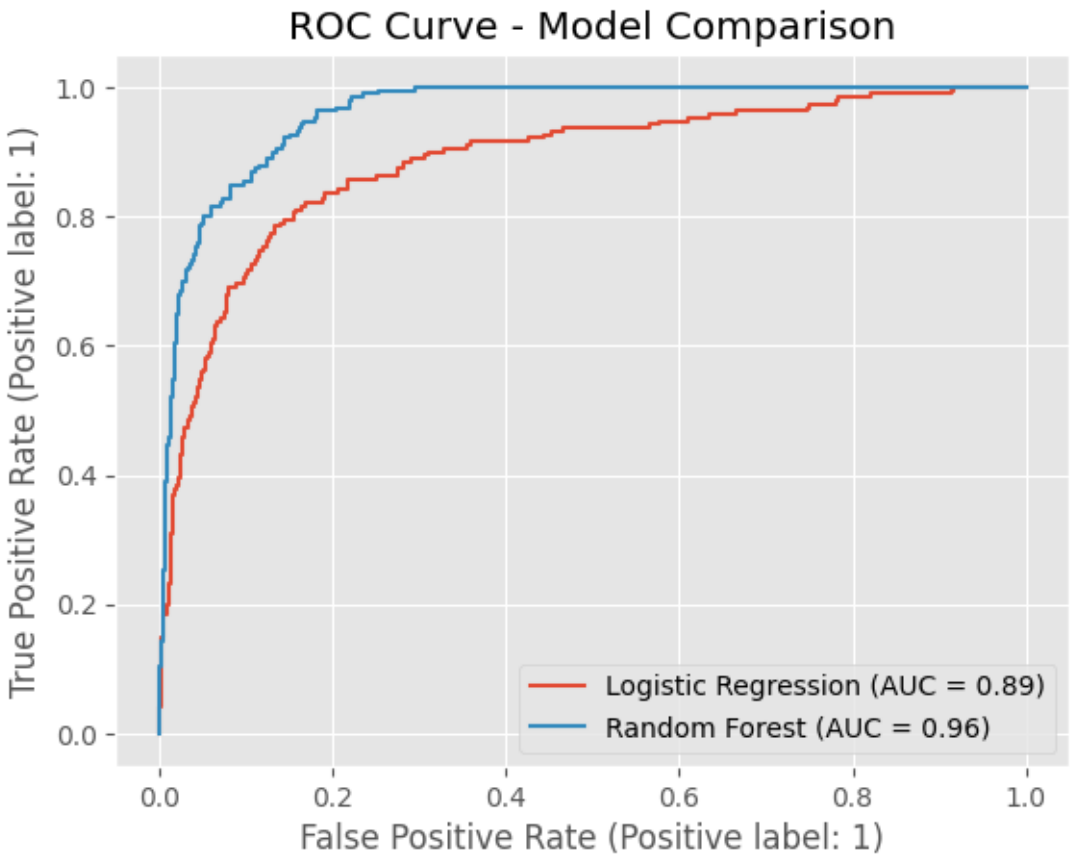
```
)

models = [
    ("Logistic Regression", log_reg_clf, None, True),
    ("Random Forest", random_forest_clf, None, True),
    ("Logistic Regression t>0.75", log_reg_clf, 0.75, False)
]
```

1.1 Initial Model Evaluation

[ ]:
```
results = {}
roc_auc_scores = {}
for model_name, model, threshold, roc_graph in models:
    model_results, y_proba = evaluate_model(model, X_test, y_test, model_name,␣
 ↪y_pred_threshold=threshold)
    results.update(model_results)
    roc_auc_scores[model_name] = y_proba

fig, ax = plt.subplots()
model_names = list(roc_auc_scores.keys())
roc_aucs = list(roc_auc_scores.values())
for model_name, _, _, roc_graph in models:
    if roc_graph:
        RocCurveDisplay.from_predictions(y_test, roc_auc_scores[model_name],␣
 ↪ax=ax, name=model_name, color = palette[model_names.index(model_name)])
plt.title("ROC Curve - Model Comparison")
plt.show()
```



[ ]:
```
# ROC-AUC scores
roc_auc_df = pd.DataFrame.from_dict(results, orient='index')[['ROC-AUC']]
roc_auc_df = roc_auc_df.sort_values(by="ROC-AUC", ascending=False).loc[['Random␣
↪Forest', 'Logistic Regression']]
roc_auc_df
```

[ ]:

	ROC-AUC
Random Forest	0.960824
Logistic Regression	0.885256

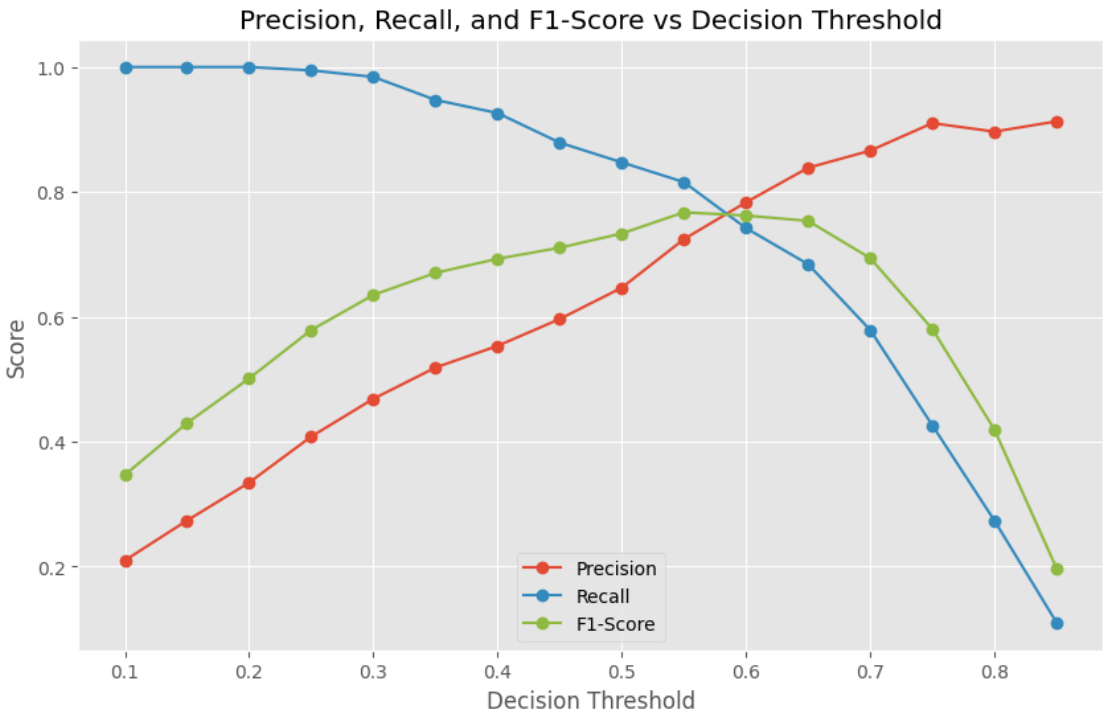
Comparing ROC-AUC scores shows that the Random Forest model outperforms the Logistic Regression baseline, indicating better overall discrimination between churners and non-churners. - ROC-AUC: 0.96 vs 0.88

1.2 Threshold Optimization for Business-aligned Metrics

[ ]:
```
opt_results = []
model_results, y_proba = evaluate_model(random_forest_clf, X_test, y_test,␣
↪"Random Forest", None)
thresholds = np.arange(0.1, 0.9, 0.05)
```

```
for threshold in thresholds:
    y_pred_threshold = (y_proba >= threshold).astype(int)
    report = classification_report(y_test, y_pred_threshold, output_dict=True)
    opt_results.append({
        "threshold": threshold,
        "precision": report["1"]["precision"],
        "recall": report["1"]["recall"],
        "f1-score": report["1"]["f1-score"]
    })
opt_results_df = pd.DataFrame(opt_results)
```

[ ]:
```
# Plot Precision-Recall vs Threshold
plt.figure(figsize=(10, 6))
plt.plot(opt_results_df["threshold"], opt_results_df["precision"],␣
 ↪label="Precision", marker='o', color=palette[0])
plt.plot(opt_results_df["threshold"], opt_results_df["recall"], label="Recall",␣
 ↪marker='o', color=palette[1])
plt.plot(opt_results_df["threshold"], opt_results_df["f1-score"],␣
 ↪label="F1-Score", marker='o', color=palette[2])
plt.xlabel("Decision Threshold")
plt.ylabel("Score")
plt.title("Precision, Recall, and F1-Score vs Decision Threshold")
plt.legend()
plt.show()
```



Key observations from the graph: - Recall decreases monotonically as threshold increases (expected) - Precision increases monotonically (expected) - F1-score: - rises steadily from low thresholds - peaks around 0.55–0.65 - then drops sharply after ~0.7 due to recall collapsing

Based on this, a threshold of 0.55 is selected for the Random Forest model to balance precision and recall.

[ ]:
```
# Evaluate all models with default and selected thresholds
models.append(
    ("Random Forest t>0.55", random_forest_clf, 0.55, False)
)

results = {}
for model_name, model, threshold, roc_graph in models:
    model_results, y_proba = evaluate_model(model, X_test, y_test, model_name,␣
 ↪y_pred_threshold=threshold)
    results.update(model_results)
```

[ ]:
```
# Business-aligned decision performance
results_df = pd.DataFrame(results).T
results_df = results_df.sort_values(by="F1-Score", ascending=False).
 ↪drop(columns=["ROC-AUC"])
results_df
```

[ ]:

	Precision	Recall	F1-Score
Random Forest t>0.55	0.724299	0.815789	0.767327
Random Forest	0.646586	0.847368	0.733485
Logistic Regression t>0.75	0.638743	0.642105	0.640420
Logistic Regression	0.463557	0.836842	0.596623

1.3 Conclusion on Model Comparison

Evaluating business-aligned metrics at selected decision thresholds further highlights the Random Forest's superiority: The Random Forest model with a decision threshold of 0.55 outperforms the baseline Logistic Regression model (threshold 0.75) across all key metrics: - F1-Score: 0.77 vs 0.64 - Precision: 0.72 vs 0.64 - Recall: 0.82 vs 0.64

This performance gain comes at the cost of reduced interpretability compared to Logistic Regression.