MachineLearning

December 1, 2024

1 Projecting High-Profit Orders with the Superstore Dataset

[2]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import numpy as np
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
import joblib
from datetime import datetime
```

[15]:
```python
# Data preprocessing function
def preprocess_data(data):
    # Handle missing values
    data = data.copy()
    numeric_columns = data.select_dtypes(include=[np.number]).columns
    data[numeric_columns] = data[numeric_columns].fillna(data[numeric_columns].
 ↪mean())

    # Feature engineering
    # Convert date columns to datetime
    data['Order Date'] = pd.to_datetime(data['Order Date'])
    data['Ship Date'] = pd.to_datetime(data['Ship Date'])

    # Create new features
    data['Processing Time'] = (data['Ship Date'] - data['Order Date']).dt.days
    data['Order Month'] = data['Order Date'].dt.month
    data['Order Day'] = data['Order Date'].dt.day
    data['Order Year'] = data['Order Date'].dt.year

    # Create profit category (target variable)
```

```
    data['Profit_Category'] = (data['Profit'] > data['Profit'].median()).
 ↪astype(int)

    # Select features for modeling
    features = ['Sales', 'Quantity', 'Discount', 'Processing Time',
                'Order Month', 'Order Day', 'Order Year']

    return data, features
```

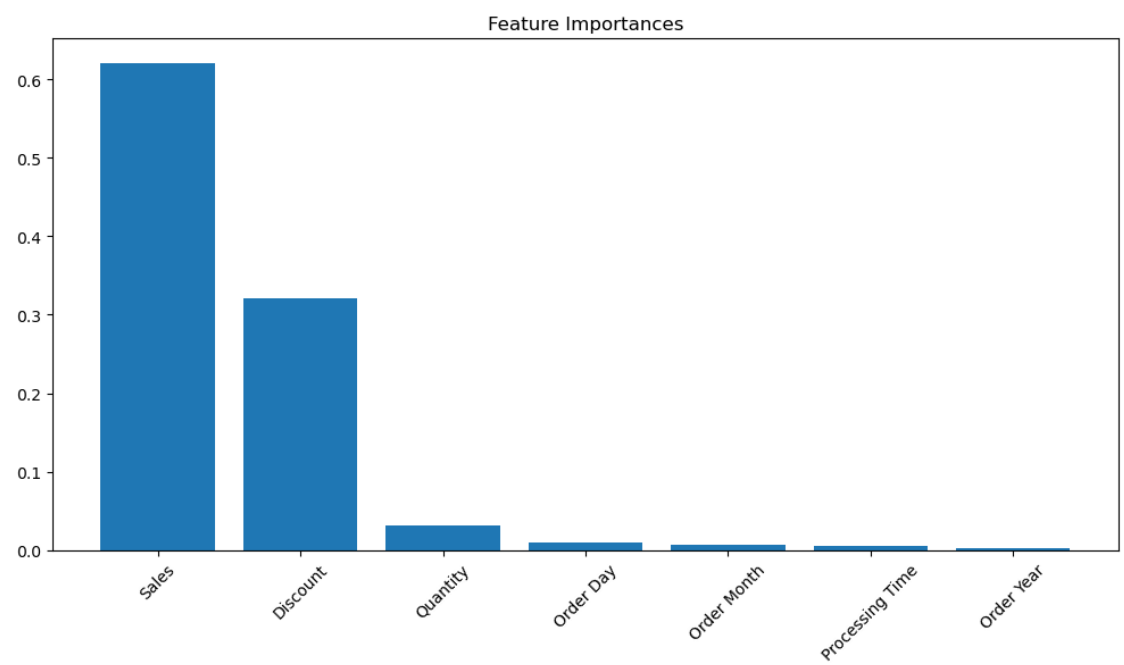
[16]:
```
# Load and preprocess data
data = pd.read_csv(r"C:\Users\lamarwells\CascadeProjects\personal-website\store.
 ↪csv", encoding="ISO-8859-1")
processed_data, features = preprocess_data(data)

# Split features and target
X = processed_data[features]
y = processed_data['Profit_Category']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=features)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)
```

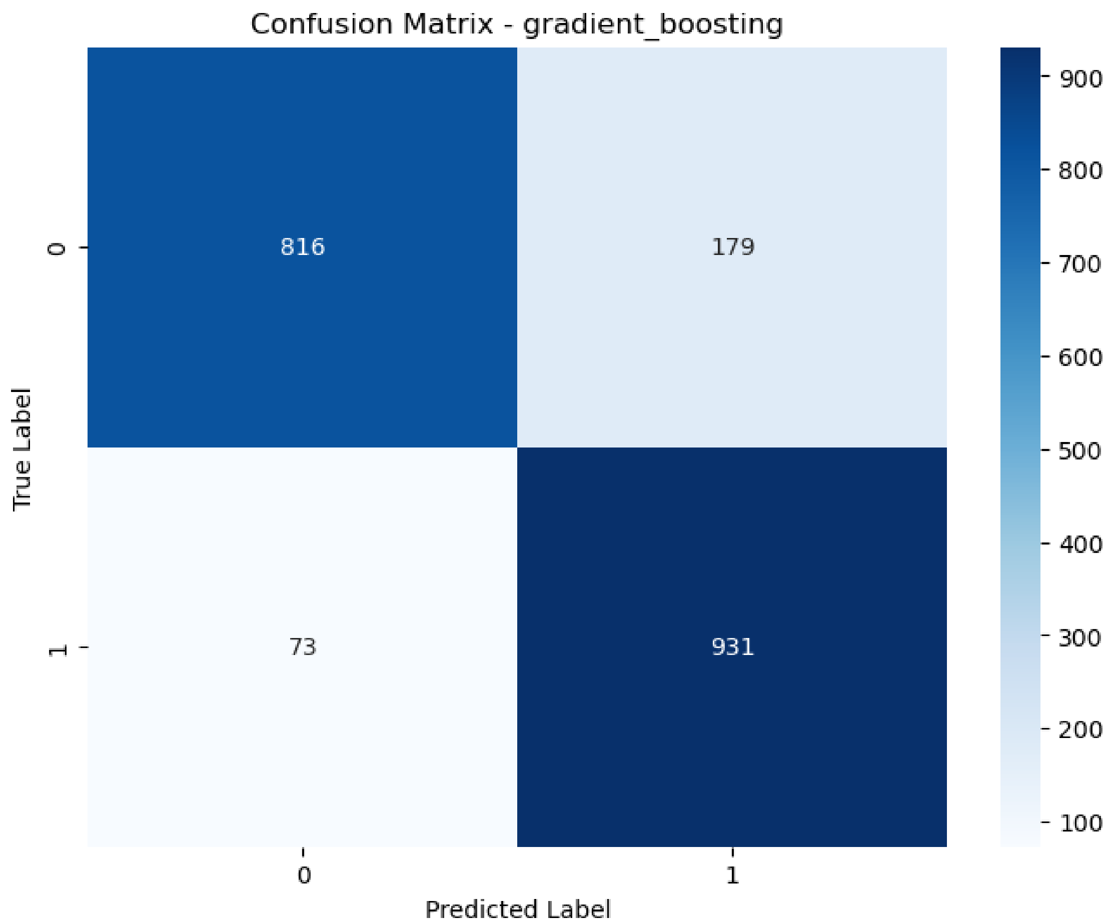
[17]:
```
# Define models to try
models = {
    'random_forest': RandomForestClassifier(random_state=42),
    'gradient_boosting': GradientBoostingClassifier(random_state=42)
}

# Define parameter grids for each model
param_grids = {
    'random_forest': {
        'n_estimators': [100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5]
    },
    'gradient_boosting': {
        'n_estimators': [100, 200],
        'learning_rate': [0.01, 0.1],
        'max_depth': [3, 5]
    }
}
```

```
# Function to train and evaluate models
def train_evaluate_model(model, param_grid, X_train, X_test, y_train, y_test):
    # Create GridSearchCV object
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='roc_auc',␣
↪n_jobs=-1)

    # Fit the model
    grid_search.fit(X_train, y_train)

    # Get best model
    best_model = grid_search.best_estimator_

    # Make predictions
    y_pred = best_model.predict(X_test)
    y_pred_proba = best_model.predict_proba(X_test)[:, 1]

    # Calculate metrics
    roc_auc = roc_auc_score(y_test, y_pred_proba)

    return best_model, y_pred, roc_auc, grid_search.best_params_

# Train and evaluate all models
results = {}
for model_name, model in models.items():
    best_model, y_pred, roc_auc, best_params = train_evaluate_model(
        model, param_grids[model_name], X_train, X_test, y_train, y_test
    )
    results[model_name] = {
        'model': best_model,
        'predictions': y_pred,
        'roc_auc': roc_auc,
        'best_params': best_params
    }
```

[19]:
```
# Function to plot feature importance
def plot_feature_importance(model, features):
    importance = model.feature_importances_
    indices = np.argsort(importance)[::-1]

    plt.figure(figsize=(10, 6))
    plt.title("Feature Importances")
    plt.bar(range(X_train.shape[1]), importance[indices])
    plt.xticks(range(X_train.shape[1]), [features[i] for i in indices],␣
↪rotation=45)
    plt.tight_layout()
    plt.show()
```

```python
# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Plot results for best model
best_model_name = max(results, key=lambda k: results[k]['roc_auc'])
best_model = results[best_model_name]['model']
y_pred = results[best_model_name]['predictions']

# Plot feature importance
plot_feature_importance(best_model, features)

# Plot confusion matrix
plot_confusion_matrix(y_test, y_pred, f'Confusion Matrix - {best_model_name}')

# Print classification report
print(f"\nClassification Report - {best_model_name}:")
print(classification_report(y_test, y_pred))
```






```
Classification Report - gradient_boosting:
              precision    recall  f1-score   support

           0       0.92      0.82      0.87       995
           1       0.84      0.93      0.88      1004

    accuracy                           0.87      1999
   macro avg       0.88      0.87      0.87      1999
weighted avg       0.88      0.87      0.87      1999
```

2 Model Evaluation: Gradient Boosting Classifier

2.1 Classification Report

The classification report provides several key metrics: - **Precision**: Ratio of correct positive predictions to total positive predictions - **Recall**: Ratio of correct positive predictions to total actual positives - **F1-score**: Harmonic mean of precision and recall - **Support**: Number of samples for each class

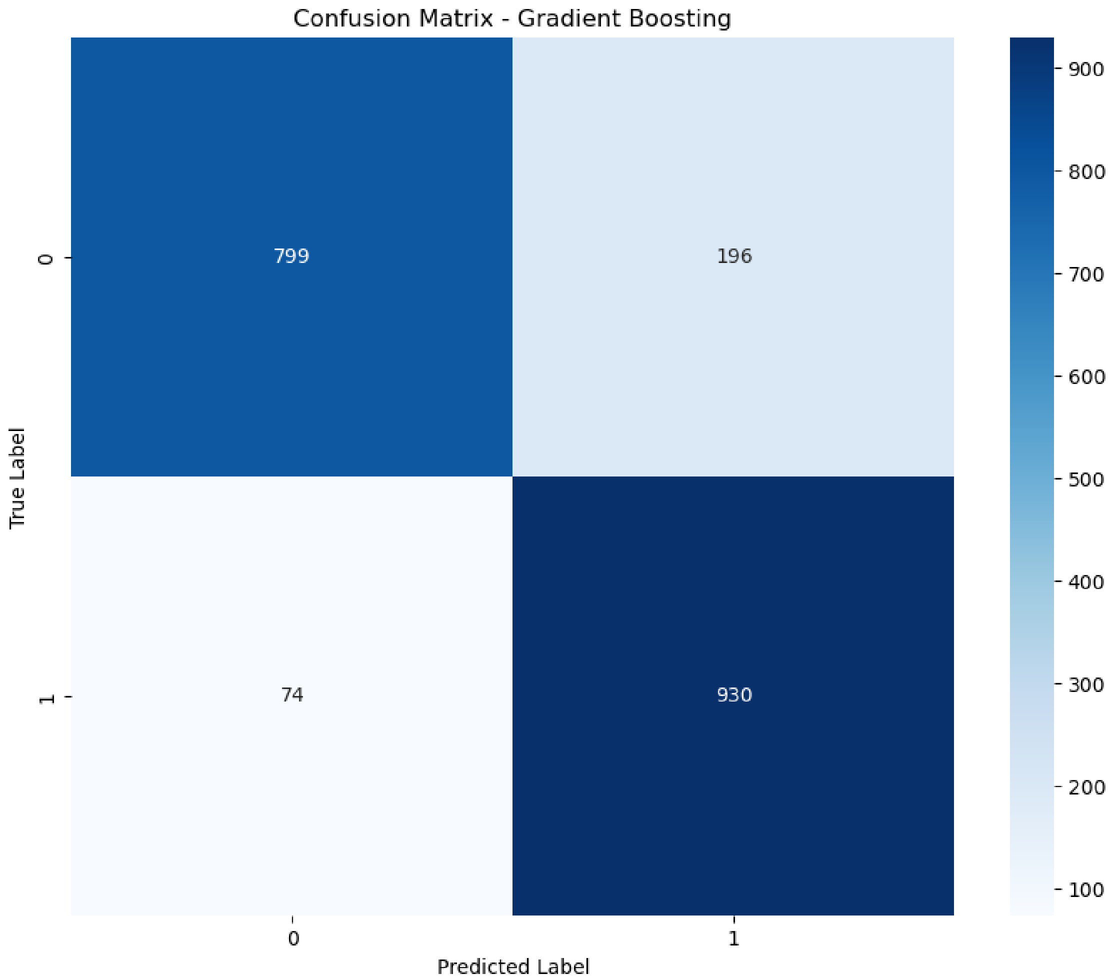
2.2 Confusion Matrix

The confusion matrix shows: - True Negatives (top-left) - False Positives (top-right) - False Negatives (bottom-left) - True Positives (bottom-right)

This visualization helps us understand: - How well the model identifies each class - Where misclassifications occur - Any class imbalance patterns

[26]:
```python
# Create and train the Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)

# Get predictions
y_pred_gb = gb_model.predict(X_test)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_gb))

# Confusion Matrix
plt.figure(figsize=(10, 8))
cm = confusion_matrix(y_test, y_pred_gb)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Gradient Boosting')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# Calculate and display accuracy
accuracy = np.sum(np.diag(cm)) / np.sum(cm)
print(f"\nAccuracy: {accuracy:.2%}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.80      0.86       995
           1       0.83      0.93      0.87      1004

    accuracy                           0.86      1999
   macro avg       0.87      0.86      0.86      1999
weighted avg       0.87      0.86      0.86      1999
```

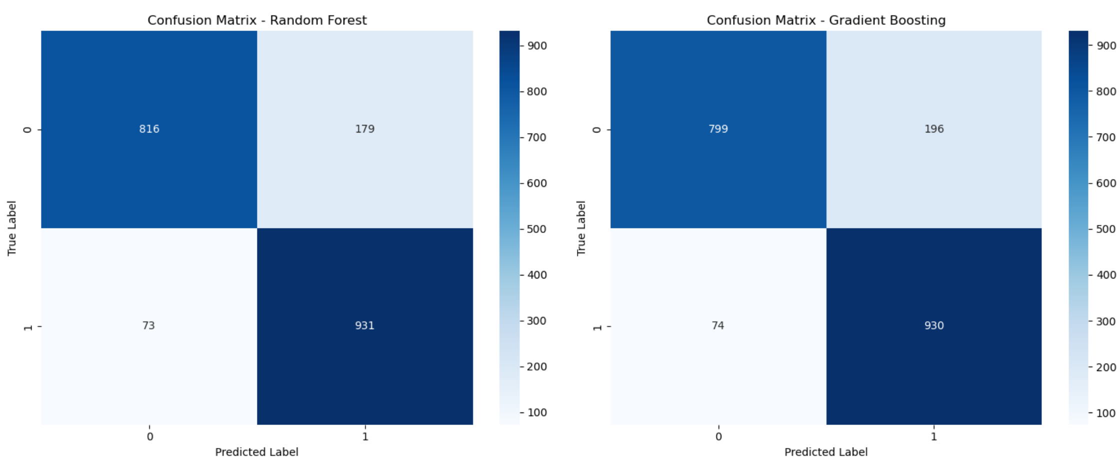



```
Accuracy: 86.49%
```

2.3 We will plot the differences between the Random Forest and Gradient Boosting confustion matrix models

[25]:
```
# Side by side confusion matrices
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Random Forest Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred)  # assuming y_pred is from Random␣
↪Forest
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', ax=ax1)
ax1.set_title('Confusion Matrix - Random Forest')
ax1.set_ylabel('True Label')
ax1.set_xlabel('Predicted Label')
```

```python
# Gradient Boosting Confusion Matrix
cm_gb = confusion_matrix(y_test, y_pred_gb)
sns.heatmap(cm_gb, annot=True, fmt='d', cmap='Blues', ax=ax2)
ax2.set_title('Confusion Matrix - Gradient Boosting')
ax2.set_ylabel('True Label')
ax2.set_xlabel('Predicted Label')

plt.tight_layout()
plt.show()

# Print comparison metrics
print("\nComparison of Models:")
print(f"Random Forest - Total Correct Predictions: {cm_rf[0,0] + cm_rf[1,1]}")
print(f"Random Forest - Total Incorrect Predictions: {cm_rf[0,1] + cm_rf[1,0]}")
print(f"Random Forest Accuracy: {(cm_rf[0,0] + cm_rf[1,1])/np.sum(cm_rf):.4f}")

print(f"\nGradient Boosting - Total Correct Predictions: {cm_gb[0,0] + cm_gb[1,1]}")
print(f"Gradient Boosting - Total Incorrect Predictions: {cm_gb[0,1] + cm_gb[1,0]}")
print(f"Gradient Boosting Accuracy: {(cm_gb[0,0] + cm_gb[1,1])/np.sum(cm_gb):.4f}")

# Calculate differences
print("\nDifferences in predictions:")
print(f"True Negatives difference (RF - GB): {cm_rf[0,0] - cm_gb[0,0]}")
print(f"False Positives difference (RF - GB): {cm_rf[0,1] - cm_gb[0,1]}")
print(f"False Negatives difference (RF - GB): {cm_rf[1,0] - cm_gb[1,0]}")
print(f"True Positives difference (RF - GB): {cm_rf[1,1] - cm_gb[1,1]}")
```



```
Comparison of Models:
Random Forest - Total Correct Predictions: 1747
Random Forest - Total Incorrect Predictions: 252
Random Forest Accuracy: 0.8739

Gradient Boosting - Total Correct Predictions: 1729
Gradient Boosting - Total Incorrect Predictions: 270
Gradient Boosting Accuracy: 0.8649

Differences in predictions:
True Negatives difference (RF - GB): 17
False Positives difference (RF - GB): -17
False Negatives difference (RF - GB): -1
True Positives difference (RF - GB): 1
```

[21]:

```python
import os

# Get the current working directory
current_dir = os.getcwd()

# Save the best model with full path
model_filename = os.path.join(current_dir, f'best_{best_model_name}_model.
 ↪joblib')
joblib.dump(best_model, model_filename)

# Print the location where the model was saved
print(f"Model saved to: {model_filename}")
```

```
Model saved to: c:\Users\lamarwells\OneDrive - Rasmussen,
Inc\Programming\AnalyticsHome\best_gradient_boosting_model.joblib
```

[20]:

```python
# Save the best model
# This model can be imported and used in other notebooks
model_filename = f'best_{best_model_name}_model.joblib'
joblib.dump(best_model, model_filename)

# Create a simple logging function
def log_results(model_name, params, metrics):
    with open('model_training_log.txt', 'a') as f:
        f.write(f"\n{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write(f"Model: {model_name}\n")
        f.write(f"Best Parameters: {params}\n")
        f.write(f"ROC-AUC Score: {metrics['roc_auc']:.4f}\n")
        f.write("-" * 50 + "\n")

# Log results for each model
for model_name, result in results.items():
```

```
    log_results(model_name, result['best_params'], {'roc_auc':␣
↪result['roc_auc']})
```