Galaxy Classification using Machine Learning

A Comparative Analysis of Classification Models

Somiddhya Biswas

February 5, 2026

Abstract

The objective of this project was to build a binary classification model to distinguish between two specific galaxy subclasses: **STARBURST** and **STARFORMING**, using spectral and photometric data from the Sloan Digital Sky Survey (SDSS). We evaluated five distinct machine learning algorithms. The **Random Forest Classifier** emerged as the superior model, achieving an accuracy of **89.5%** and demonstrating the best balance between precision and recall.

Contents

1 Introduction

Galaxy classification is a fundamental task in astronomy that helps in understanding the evolution and formation of the universe. This project utilizes dataset containing 100,000 observations from the SDSS to classify galaxies based on their spectral characteristics. The primary goal is to distinguish between 'Starforming' galaxies and 'Starburst' galaxies.

2 Data Preprocessing and EDA

Before feeding data into the models, rigorous preprocessing was applied to ensure fair comparisons.

2.1 Data Cleaning

- **Handling Missing Values:** Rows containing placeholder values (-9999.0) were identified and removed to prevent skewing results.
- **Feature Selection:** Non-predictive identifiers such as `objid` and `specobjid` were dropped.
- **Scaling:** We applied `StandardScaler` to normalize feature values. This was critical for distance-based models (KNN) and linear models (Logistic Regression) as features like `flux` and `redshift` exist on vastly different scales.

2.2 Exploratory Data Analysis

The dataset is imbalanced, with a higher prevalence of Starforming galaxies compared to Starburst galaxies.

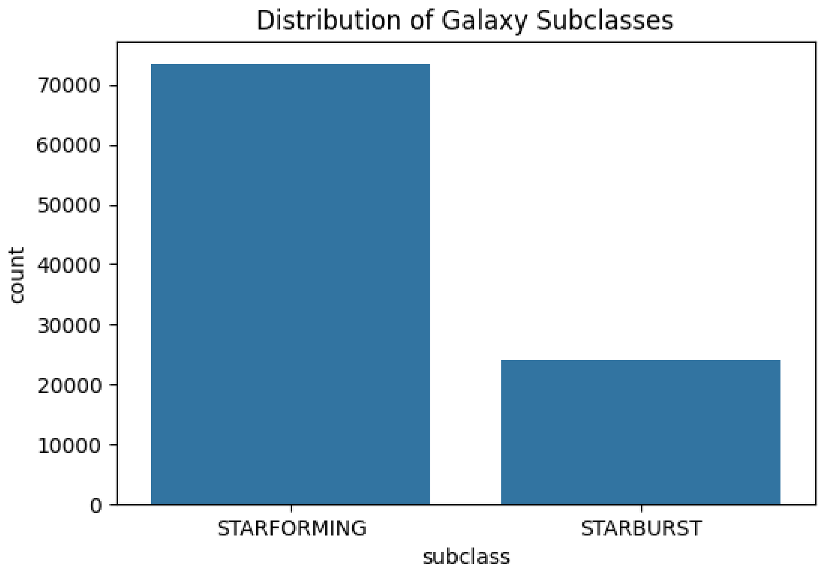

Figure 1: Distribution of Galaxy Subclasses showing class imbalance.

3 Model Analysis

We selected five diverse algorithms to test different learning paradigms.

3.1 Logistic Regression (The Baseline)

What it is: A statistical model that assumes a linear relationship between input features and the log-odds of the target class.
Performance: It performed surprisingly well (87.6%), indicating that the boundary between these galaxy types is somewhat linear, though it lacks the complexity to capture subtle non-linear spectral patterns.

3.2 Decision Tree (The Rule-Based Learner)

What it is: A non-parametric model that learns explicit cutoff points for features.
Performance: This was the lowest performing model (84.9%). Single decision trees are prone to overfitting, memorizing noise in the training data rather than learning general patterns.

3.3 K-Nearest Neighbors (KNN)

What it is: An instance-based algorithm that classifies a new galaxy based on the majority class of its 'K' closest neighbors.
Performance: Moderate performance (85.5%). KNN struggled with the high-dimensionality of the spectral data.

3.4 Random Forest (The Ensemble) – Best Model

What it is: An ensemble method that builds hundreds of decision trees and merges them to improve stability and accuracy.
Performance: The winner (89.5%). Its ability to handle non-linear relationships and resistance to noise made it ideal for this dataset.

3.5 Gradient Boosting

What it is: An iterative ensemble technique that builds trees sequentially to correct errors made by previous trees.
Performance: A close second (88.5%). While powerful, it was slightly more sensitive to noise than the Random Forest in this specific experiment.

4 Results and Discussion

4.1 Performance Metrics

The table below summarizes the weighted metrics for all models on the test set.

Table 1: Model Comparison Results

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Random Forest	**0.895**	**0.893**	**0.895**	**0.894**
Gradient Boosting	0.885	0.882	0.885	0.882
Logistic Regression	0.876	0.872	0.876	0.871
KNN	0.855	0.849	0.855	0.849
Decision Tree	0.849	0.849	0.849	0.849

4.2 Visual Comparison

The chart below highlights that while all models achieved above 84% accuracy, Random Forest consistently outperformed others in F1-score, which is crucial for our imbalanced dataset.

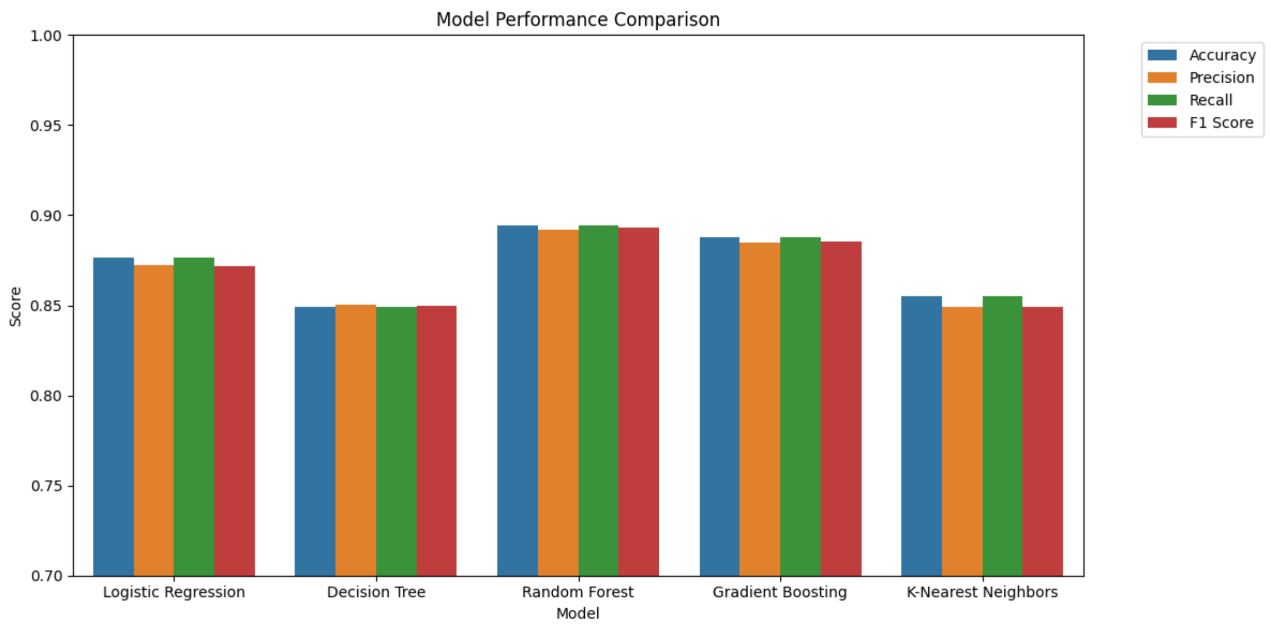

Figure 2: Performance metrics comparison across five models.

4.3 Redshift Distribution

The distribution of redshift values differs slightly between subclasses, providing a key feature for classification. The plot below illustrates these differences.

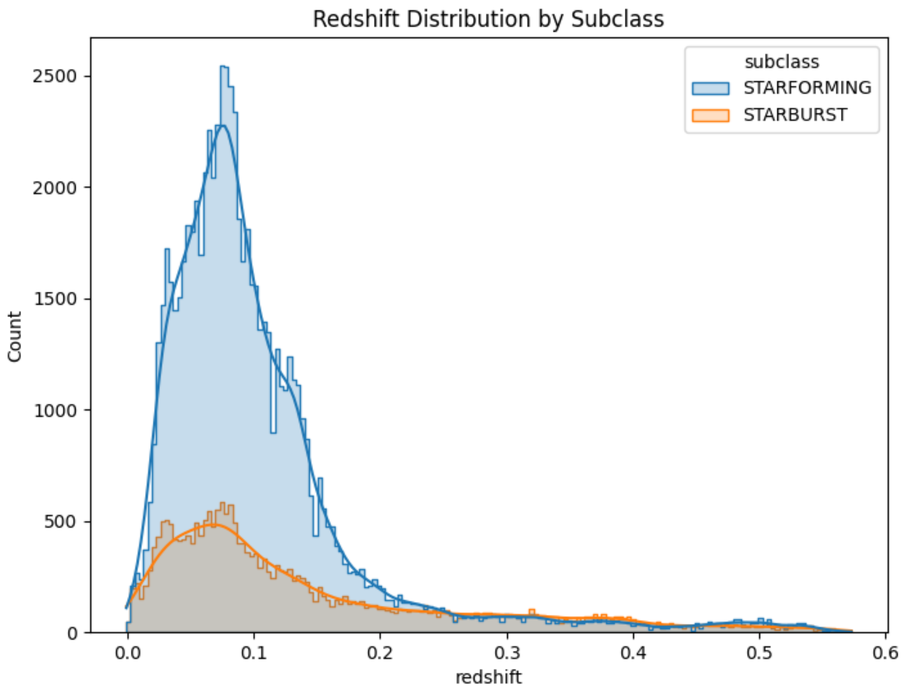

Figure 3: Redshift distribution comparison between Starburst and Starforming galaxies.

5 Conclusion

The comparative analysis highlights that **Ensemble Methods** are necessary to achieve high accuracy in astronomical classification tasks. The **Random Forest** model is recommended for deployment due to its high accuracy (89.5%) and robust handling of class imbalance.

6 Appendix: Python Code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, classification_report

warnings.filterwarnings('ignore')

# 1. Load Data
# Note: Ensure the path points to your actual CSV file
df = pd.read_csv('sdss_100k_galaxy_form_burst.csv', skiprows=1)

# 2. Data Cleaning
# Replace -9999 with NaN and drop missing values
df_cleaned = df.replace(-9999.0, np.nan)
df_cleaned = df_cleaned.dropna()

# 3. Visualization (Exports images for report)
plt.figure(figsize=(6, 4))
sns.countplot(data=df_cleaned, x='subclass')
plt.title('Distribution of Galaxy Subclasses')
plt.savefig('subclass_distribution.png')

plt.figure(figsize=(8, 6))
sns.histplot(data=df_cleaned, x='redshift', hue='subclass', kde=True,
    element="step")
plt.title('Redshift Distribution by Subclass')
plt.savefig('redshift_distribution.png')

# 4. Preprocessing
drop_cols = ['objid', 'specobjid', 'class', 'subclass']
X = df_cleaned.drop(columns=drop_cols)
y = df_cleaned['subclass']

# Encode Target
le = LabelEncoder()
y_encoded = le.fit_transform(y)
class_names = le.classes_

# Split and Scale
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size
    =0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Model Training & Evaluation
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state
    =42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=50, random_state
    =42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=50,
    random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5)
}

results = []
print("Training models...")

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    # Calculate Metrics
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        "Model": name,
        "Accuracy": acc,
        "F1 Score": f1
    })

# 6. Results
results_df = pd.DataFrame(results)
print(results_df)

# Generate Comparison Chart
plt.figure(figsize=(12, 6))
results_melted = results_df.melt(id_vars="Model", var_name="Metric",
    value_name="Score")
sns.barplot(data=results_melted, x="Model", y="Score", hue="Metric")
plt.title("Model Performance Comparison")
plt.savefig('model_comparison.png')
```

Listing 1: Python Implementation