Spotify Data Analysis and Model Performance Report

Introduction

This project's main goal is to categorize Spotify tracks as "popular" or "not popular" by using a popularity threshold value (higher than 50). Numerous characteristics, including danceability, energy, loudness, tempo, and others, are included in the dataset and are used to forecast a song's likelihood of becoming famous. To assess their performance and determine which classification model was best for this job, a variety of models were used, including Random Forest, KNN, SVM, and Logistic Regression.

Dataset and Data Exploration

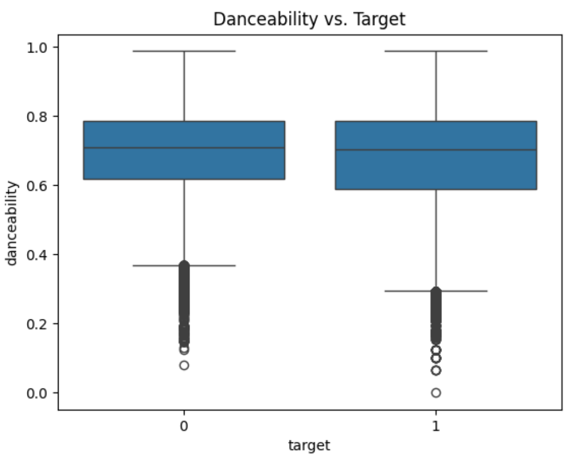
Data : https://www.kaggle.com/datasets/asaniczka/top-spotify-songs-in-73-countries-daily-updated

Target Variable : Popularity

Features: Danceability, energy, loudness, tempo, speechines, acousticness, instrumentalness, valence, duration_ms, is_explicit.

Target vs. Danceability:

Both popular and non-popular songs have comparable median danceability values, according to the



boxplot. There is no obvious difference between the two classes based on danceability alone. The classes were balanced using SMOTE which made sure the models could learn from both well-known and obscure tunes.

Code

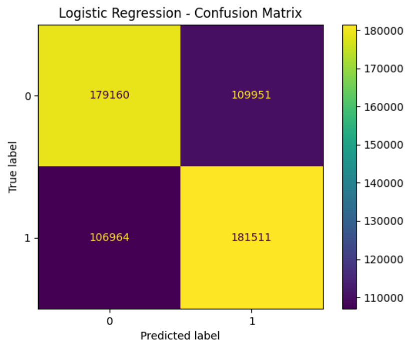
```
print("Dataset Information:")
print(data.info())
print("\nClass Distribution:")
print(data['target'].value_counts())
print("\nStatistical Summary:")
print(data.describe())

sns.boxplot(x='target', y='danceability', data=data)
plt.title("Danceability vs. Target")
plt.show()
```

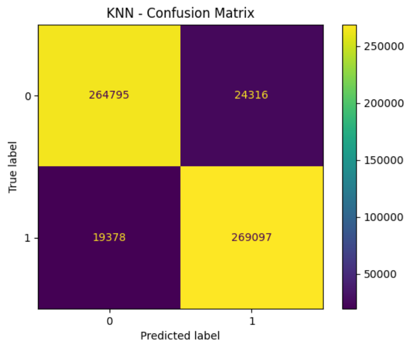
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
smote = SMOTE(random_state=42)
x_resampled, y_resampled = smote.fit_resample(x_scaled, y)
```

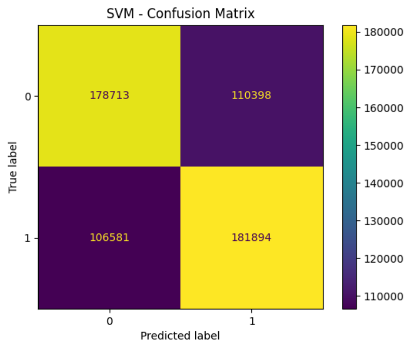
Model Training and Evaluation

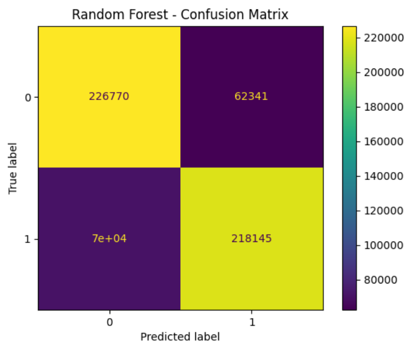
Multiple categorization algorithms were evaluated to assess their effectiveness in forecasting the popularity of Spotify songs. Logistic Regression, KNN, SVM, and Random Forest were assessed for their accuracy, computational efficiency, and capacity to manage class imbalance. These models were

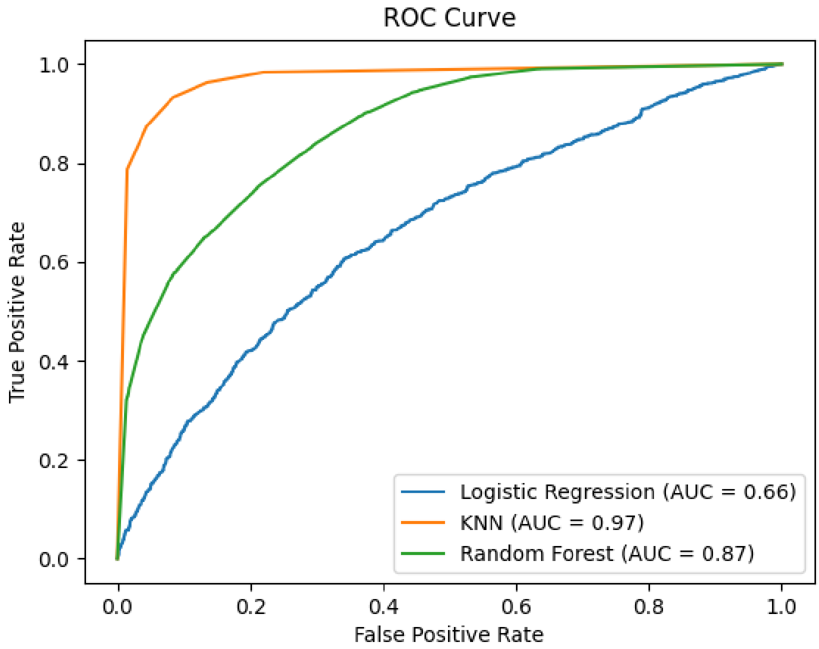
```
Model: Logistic Regression
Training Time: 1.07 seconds
Accuracy: 0.6244
              precision    recall  f1-score   support

           0       0.63      0.62      0.62    289111
           1       0.62      0.63      0.63    288475

    accuracy                           0.62    577586
   macro avg       0.62      0.62      0.62    577586
weighted avg       0.62      0.62      0.62    577586


Model: KNN
Training Time: 77.25 seconds
Accuracy: 0.9244
              precision    recall  f1-score   support

           0       0.93      0.92      0.92    289111
           1       0.92      0.93      0.92    288475

    accuracy                           0.92    577586
   macro avg       0.92      0.92      0.92    577586
weighted avg       0.92      0.92      0.92    577586


Model: SVM
Training Time: 23.81 seconds
Accuracy: 0.6243
              precision    recall  f1-score   support

           0       0.63      0.62      0.62    289111
           1       0.62      0.63      0.63    288475

    accuracy                           0.62    577586
   macro avg       0.62      0.62      0.62    577586
weighted avg       0.62      0.62      0.62    577586


Model: Random Forest
Training Time: 0.12 seconds
Accuracy: 0.7703
              precision    recall  f1-score   support

           0       0.76      0.78      0.77    289111
           1       0.78      0.76      0.77    288475

    accuracy                           0.77    577586
   macro avg       0.77      0.77      0.77    577586
weighted avg       0.77      0.77      0.77    577586
```











selected for their diverse complexity and capabilities in tackling various facets of the dataset. The performance of each model is outlined below.

The models' discriminating power was assessed using the ROC curve analysis. The AUC value of 0.66 in logistic regression suggests that the model has a limited capacity to differentiate between popular and non-popular compositions. The dataset's non-linear interactions challenge the linear model. KNN has the highest discriminating power of the models with an AUC of 0.97. This high number shows the model's ability to capture complicated feature interactions and separate the classes. Random Forest showed good performance, balancing true positive and false positive rates with an AUC of 0.87. Although slower than KNN, Random Forest is a good alternative because to its computational efficiency and versatility. Overall, the ROC analysis confirms that KNN is the most effective model for this classification task.

Conclusions and Recommendations

With the greatest accuracy and AUC, the research shows that KNN is the best model for forecasting Spotify song popularity. Scalability for real-time applications, however, can be constrained by its computational expense. A good substitute with quicker training times and respectable accuracy is Random Forest.

These findings have the following implications:

- Music platforms now have improved prediction skills to find possible hits.
- Producers and artists may use these strategic insights to customize their music according to the significance of each aspect.
- Possibilities to enhance model performance by adding other features like release year or artist popularity.

All Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, roc_curve, auc,
ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
import time
import seaborn as sns
import matplotlib.pyplot as plt

file_path = "YOURDATAPATH"
data = pd.read_csv(file_path

data['target'] = (data['popularity'] > 50).astype(int)

features = [
    'danceability', 'energy', 'loudness', 'tempo', 'speechiness',
    'acousticness', 'instrumentalness', 'valence', 'duration_ms', 'is_explicit'
]

x = data[features]
y = data['target']

print("Dataset Information:")
print(data.info())
print("\nClass Distribution:")
print(data['target'].value_counts())
print("\nStatistical Summary:")
print(data.describe())

sns.boxplot(x='target', y='danceability', data=data)
plt.title("Danceability vs. Target")
plt.show()

scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

smote = SMOTE(random_state=42)
x_resampled, y_resampled = smote.fit_resample(x_scaled, y)

x_train, x_test, y_train, y_test = train_test_split(x_resampled, y_resampled, test_size=0.2,
random_state=42)

x_train_small = x_train[:5000]
y_train_small = y_train[:5000]

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, class_weight='balanced'),
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(kernel='linear'),  # Linear kernel for faster training
    'Random Forest': RandomForestClassifier(n_estimators=10, n_jobs=-1)
}

# Train and evaluate models
results = {}
for name, model in models.items():
    start_time = time.time()


    if name in ['SVM', 'Random Forest']:
        model.fit(x_train_small, y_train_small)
    else:
        model.fit(x_train, y_train)


    y_pred = model.predict(x_test)
```

```python
    elapsed_time = time.time() - start_time
    results[name] = accuracy_score(y_test, y_pred)


    print(f"\nModel: {name}")
    print(f"Training Time: {elapsed_time:.2f} seconds")
    print(f"Accuracy: {results[name]:.4f}")
    print(classification_report(y_test, y_pred))


results_df = pd.DataFrame(results.items(), columns=['Model', 'Accuracy'])
print("\nModel Performances:")
print(results_df)



for name, model in models.items():
    y_pred = model.predict(x_test)
    disp = ConfusionMatrixDisplay.from_estimator(model, x_test, y_test)
    disp.ax_.set_title(f"{name} - Confusion Matrix")
    plt.show()



for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(x_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_score)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=3, scoring='accuracy')
grid_search.fit(x_train_small, y_train_small)
print("\nBest Parameters from Grid Search:")
print(grid_search.best_params_)
```