Evaluation of models

In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import preprocessing
import numpy as np
from sklearn import *
import matplotlib.pyplot as plt
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

In [ ]:
```
file_path = '/content/gdrive/MyDrive/Capstone/final_dataset.csv'
df=pd.read_csv(file_path,)
df2=df.drop(['index', 'Title',  'Artist','track_id'], axis=1)
df2
X = df2.drop(['Popular'], axis=1)
y = df2['Popular']
```

In [ ]:
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(sampling_strategy="not majority")
X_r, y_res = ros.fit_resample(X, y)
stsc=preprocessing.StandardScaler()
X_res = stsc.fit_transform(X_r)
#splitting data into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_res, y_res, test_size=0.30, random_state=42)
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
models=[]
```

In [ ]:
```
df2
```

Models

We are using 3 models: Logistic regression, Random forest and XGBoost. We are still in the process of developing a Convolutional Neural Network.

In [ ]:
```
#making predictions
```

Evaluation

Logistic regression

In [ ]:
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
models.append(lr)
score = lr.score(X_test, y_test)
# make predictions
predictions = lr.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

Precision and Recall provide a deeper look into model performance than overall accuracy. We focused on precision and recall specifically for the minority positive class in order to directly measure how well the model predicts popular songs, which is the main objective.

Precision of 87% indicates that of all the songs predicted as popular by the model, about 87% actually were popular according to the ground truth Billboard data. This shows the model is avoiding a high rate of false positives. We want to maximize precision to ensure our predicted popular songs are indeed likely to be hits based on the features.

Recall of 89.5% means the model is correctly predicting almost 90% of the songs that were actually popular as per Billboard. This shows we are not missing many real hits. High recall aligns with our goal of trying to identify as many of the true popular songs as possible, minimizing false negatives.

Balancing precision and recall is important - optimizing one at the expense of the other is not helpful. The F1 score combines both metrics by taking their harmonic mean. The high F1 score of 88.2% demonstrates we have achieved both solid precision and recall

together.

When tuning the model, adjusting the classification threshold impacts this balance. A higher threshold improves precision but lowers recall. We found 0.5 threshold optimal, but further optimization may improve the F1 score.

The confusion matrix revealed that the majority of errors were false positives rather than false negatives. This aligns with optimizing for recall. We prefer to mistakenly predict some non-hits as popular vs. miss many actual hits.

By focusing on precision and recall for the key positive class, we directly measured model performance on the main task. Metrics guided developing a model tuned to minimize errors in identifying and predicting popular Billboard songs based on audio features.

Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

# start a model with 500 decision trees
rf = RandomForestClassifier(n_estimators = 500, max_depth = 4, max_features = 4, bootstrap = True, random_state

# train the model
rf.fit(X_train, y_train)
models.append(rf)


predictions = rf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

Let's start by analysing the precision, recall and F-1 scores, since it allows us to make an overall evaluation of the model's performance. Precision of 78,07% means the proportion of correct positive predictions made by the model. In the following case, it means that high percentage of positive predictions made by the model were accurate.

Recall also reflects the model's ability to make positive predictions, yet it evaluates the proportion of correct positive predictions made out of all positive predictions (considers missed positive predictions). 92,52% recall indicates good ability of the model to capture all the actual positive cases. The F-1 score of 84,68% illustrates a balanced performance of both precision and recall. This shows that the model has reached a good trade-off between identifying positive instances and minimizing false positives.

In [ ]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(rf, X_res, y_res, cv=5, scoring='accuracy')

for i, score in enumerate(cv_scores):
    print(f"Fold {i + 1}: {score}")
print(f"\nMean Accuracy: {cv_scores.mean()}")
```

To achieve a better results, we performed a cross-validation, and we can observe that across the folds the model performs well (the mean accuracy equals to 85,99%). Scores are ranging from 81,62% to 87,77%. These results illustrate that the model has perform well and stable while working on different subsets of data.

To optimize our model further, we could adjust the classification threshold. This can affect the trade-off between precision and recall. In other words, it might increase precision, but decrease recall. After analysing different parameters of the model, we have concluded that changing any of them would negatively affcet the model's performance (n_estimators, max_depth, max_features, bootstrap, and random_state). Hence, the current values of each of the parameter is an optiaml setting for our dataset and model.

Overall, the evaluation of the model was mainly focused on two main metrics (recall and precision), as those demonstrate how well the model handles teh main task (identifying if the song is going to be a hit or not).

XGboost

In [ ]:
```
#installing
!pip install xgboost
#imports
from numpy import loadtxt
from xgboost import XGBClassifier
#setting parameters for data splitting
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=test_size, random_state=seed)
# training model
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
predictions = [round(value) for value in y_pred]
```

In [ ]:
```
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

The model is performing well and has quite similar high values in all four performance metrics. The accuracy of the model is at 87,61%, which means that the model correctly predicts the class lables for about 87,61% of all instances. This is a high value of accuracy, and it shows that the model performs well overall.

Precision of 86,32% shows that the model has a good ability of predicting correct positive predictions. This is crucial, since it illustrates the model's ability to avoid incorrect classification of instances to the target class.

Recall of the model is at 89,81%, which means that a high percentage of real positive instances were classified correctly by the model. This is especially beneficial, when it is better to avoid false negatives.

The F-1 score reaches 88,03% illustrates that the model has a good trade off between precision and recall.

In [ ]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(xgb, X_res, y_res, cv=10, scoring='accuracy')

for i, score in enumerate(cv_scores):
    print(f"Fold {i + 1}: {score}")

print(f"\nMean Accuracy: {cv_scores.mean()}")
```

The cross-validation results across diferent folds range from 82,64% to 92,33%, which demonstrates consistency of a model performance while working with different subsets of data. The mean accuracy is also high at 88,29%.

In summary, XG Boost model performance seems good and stable with all performance metrics being high and well-balanced between each other.

In [ ]:
```
import matplotlib.pyplot as plt

models = ['Logistic Regression', 'Random Forest', 'XGBoost', 'RNN']
accuracy = [0.760631443298969, 0.7777061855670103, 0.9296069587628866, 0.8145940721649485]
precision = [0.7254682694995807, 0.7389491242702252, 0.8822860392379869, 0.7762336925694838]
recall = [0.8373668925459826, 0.8576960309777347, 0.9926423544465771, 0.883188125201678]
f1 = [0.7774116237267825, 0.7939068100358422, 0.9342164684630438, 0.8262641509433962]

x = np.arange(len(models))
width = 0.2

fig, ax = plt.subplots()
rects1 = ax.bar(x - width, accuracy, width, label='Accuracy')
rects2 = ax.bar(x, precision, width, label='Precision')
rects3 = ax.bar(x + width, recall, width, label='Recall')
rects4 = ax.bar(x + 2*width, f1, width, label='F1 Score')

ax.set_ylabel('Scores')
ax.set_title('Model Evaluation Metrics')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

fig.tight_layout()
plt.show()
```

Accuracy: 0.8145940721649485 Precision: 0.7762336925694838 Recall: 0.883188125201678 F1 Score: 0.8262641509433962

For Accuracy:

- XGBoost had the highest accuracy score at 0.8734. This means overall, XGBoost made the correct classification 87.34% of the time.
- Logistic Regression had the lowest accuracy at 0.8186 or 81.86% correct predictions.
- There was a ~6% difference between the highest (XGBoost) and lowest (Logistic Regression) accuracy scores.

For Recall:

- Random Forest had the highest recall at 0.921. This means Random Forest correctly identified 92.1% of all the positive examples.
- Logistic Regression had the lowest recall at 0.7879. It only correctly identified 78.79% of the actual positives.
- There was a 13.31% difference between the recall of Random Forest and Logistic Regression. This indicates Random Forest was

much better at avoiding false negatives.

For Precision:

- XGBoost had the best precision at 0.8576. When XGBoost predicted positive, it was correct 85.76% of the time.
- Random Forest had the lowest precision at 0.7731. Its positive predictions were only accurate 77.31% of the time.
- There was an 8.45% difference in precision between XGBoost and Random Forest. XGBoost positive predictions were more reliable.

For F1 scores:

- XGBoost achieved the highest F1 score, meaning it balanced precision and recall the best.
- Logistic Regression had the lowest F1 score, indicating it did not balance precision and recall as well.

In summary, XGBoost performed the best overall, with the top accuracy, precision, and F1 scores. Random Forest had the highest recall, meaning it was best at finding positive examples. Logistic Regression lagged behind in all metrics.