Project “Pop or flop”

Do you have what it takes to be the next superstar?

Team: Indecisive trees

Muhammad Ali, Syed Aziz, Omer Rosen

1. Question:

- Can we predict with a certain amount of confidence whether a song will be a hit or a flop?
- You were hired by a production company to help them decide whether to enlist a new artist to their company. The production company has provided you with a song list for an artist and you need to predict if the artist has a song that can be a hit.

2. Context:

- This dataset, created by Farooq Ansari, consists of features for tracks from Spotify's Web API. The data consists of six csv files each file represents a decade worth of datapoints, where each data point is an audio file. The decades range from the 60s to the 2010s inclusive of both end points. There are several features in our dataset.
- Our response variable is called ‘target’ and our independent variables are 'track', ‘artist', 'uri', 'danceability', 'energy', 'key', 'loudness','mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature', 'chorus_hit','sections', 'target', and 'decade'.
- We will not use all these features to predict but these are the features that we are starting out with.

2.1 Attribute Exploration

2.1.1 Target

target: Is our response variable '1' for hit '0' for flop

Conditions for a flop (as provided by Farooq Ansari):

- The track must not appear in the 'hit' list of that decade.
- The track's artist must not appear in the 'hit' list of that decade.
- The track must belong to a genre that could be considered non-mainstream and / or avant-garde.
- The track's genre must not have a song in the 'hit' list.
- The track must have 'US' as one of its markets.

2.1.2 Predictors

- **danceability**: Danceability describes how suitable a track is for dancing. The value of will between 0 and 1.
- **energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
- **key**: The estimated overall key of the track. These range between 0 and 11.
- **loudness**: The overall loudness of a track in decibels (dB). Values typically range between -60 and 0 db.
- **mode**: Mode indicates the modality (major or minor) of a track. Major is represented by 1 and minor is 0.
- **speechiness**: Speechiness detects the presence of spoken words in a track. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values below 0.33 most likely represent music and other non-speech-like tracks.
- **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
- **instrumentalness**: Predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
- **liveness**: Detects the presence of an audience in the recording. A value above 0.8 provides strong a likelihood that the track is live.

- **valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
- **tempo**: The overall estimated tempo of a track in beats per minute (BPM).
- **duration_ms**: The duration of the track in milliseconds.
- **time_signature**: An estimated overall time signature of a track.
- **chorus_hit**: This is the best estimate of when the chorus would start for the track. It's the timestamp of the start of the third section of the track (in milliseconds).
- **sections**: The number of sections the track has.

2.2 Additional Attributes

2.2.1 Decade

The data consist of six csv files each file represents a decade worth of datapoints.
The decades range from the 60s to the 2010s inclusive of both end points. When loading the data, we will use the decade as an additional feature

2.2.1 Gender

To further explore the data, we will try to seek additional information about the artists. To do so, we will use the trackId of each song and build a scraper.

2.2.2 Spotify Scraping

- Step 1: From our Dataset we loop through the list of uri in the "uri" column and convert them to URLs and save these into a list called ("song_url_list")
- Step 2: Next, we take our "song_url_list" and loop through each song URL page one by one and create a second list of Artist URLs where there is information about the artist's songs, biographies etc. We called this "artist_url_list".
- Step 3: Next, we create our third loop and go over each link one by one. From the links we scrape the artist page using the "BeautifulSoup" Library and scan the text for some of our keywords to predict some gender information for each artist.
- Step 4: For our keywords we created a dictionary with three keywords: "M" for male, "F" for female and "Band". Each keyword had some of the common words associated with these and we picked the keyword with the highest count. We saved it under "gender_info_list". (e.g. – A high occurrence of "his" and "him" keywords will likely indicate that the artist is a Male).
  For cases where the bio was empty or did not contain gender related keywords we marked: "unknown"
- For simplicity's sake we did not further explore these artists for their preferred pronouns but there is more room here. For that we will have to get detailed biographies from some other external sources as well.

Note: We divided our dataset into 6 different parts and simultaneously worked on these datasets for computational purposes and to save time because overall we had over 41,000 tracks and overall, this part took a few days to complete.

3 Exploring the Data

3.1 Observing the data types

We observed the column types, the scales of attributes, plotted the feature relations with the target and made mental notes of which features appear correlated.

3.2 Splitting into feature types

3.2.1 Binary

The binary_columns was 'mode'

3.2.2 Categorical

The categorical columns were 'key', 'time_signature', 'decade', 'artist_gender' - Less than 20 unique values for column.

3.2.3 Numerical

The numerical_columns were 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'chorus_hit','sections'

3.2.4 Y

Our target variable y is 'target'

3.2.5 Columns to drop

We decided drop 'track','artist','uri'

- 'track' : Because the name of the track does not help us in our prediction as every name is practically unique
- 'artist': Because there are 11,903 unique artists
- 'uri' : Because uri is a unique identifier for each song

3.3 Looking at data balance

Looking at the distribution per our binary and categorical columns.

3.3.1 Target class balance

An interesting find is that our target classes are perfectly balanced
This isn't necessarily a positive thing since it can indicate a potential bias in the data collection (e.g. - 20,553 are all the hits from the past six decades, while 20,553 flops are just a small sample from an endless population of flops)

3.2.3.2 Categorical column balance

There appears to be an imbalance with most of our categorical data, the most notable being time_signature with a high frequency of value '5'. This will need to be re-examined once we build our model and estimate the feature importance.

3.4 Looking at each predictor relations to the target

We observe the relationship between each predictor individually and attempt to see if any notable correlation is spotted between the feature and whether the song is a hit. We noticed some interesting relationships:

Dancability - If your song has less than ~0.2 dancability rate you will not become a hit.
Speechness - If your speech-to-music ration is higher than 0.45 it decreases your odds significantly at having a hit.
Duration - The longer your song lasts the fewer people will listen to it.

3.5 NaN values

While the original dataset doesn't contain NaN values, scraping Spotify we were occasionally unable to retrieve data regarding gender, either due to missing bio or lack of pronouns in it.

We have several options to tackle this:

- We could assume that the data was **Missing completely at random** (MCAR).
  Meaning - missing artist description on Spotify is NOT related to the artist's gender.
  In that case we could randomly fill the missing data or delete the missing rows from our dataset.
- Assume our data is **Missing at random** (MAR) or **Missing not at random** (MNAR).
  Meaning - There is a reason why certain artists did not have a description or a pronoun in their bio, and this can correlate to their gender. e.g. - Male artists put less emphasis on maintaining their Spotify account and are less likely to fill a detailed bio.

In our case, we will assume MNAR and tackle this by using a KNNImputer. (Additional details can be found in the notebook).
Since our missing data is textual, we will need to first convert it to integers, impute, and then rename it back.
This also means that we can only use n_neighbors=1 since we want round numbers and not decimals.

3.6 Removing non-music data - Speechiness level

Per Kaggle, any datapoint above 0.66 speechness ratio is not considered a song, but rather a podcast or lecture. As such, we would like to filter these out.

This removed 141 records (less than 0.3% of our data)

3.7 One Hot Encoding Categorical Columns

We Identified our categorical columns as ‘key’, ‘time_signature’, ‘atrist_gender’ and ‘decade’. We then used a ‘For Loop’ to iterate over them to one-hot-encode them. The key column was converted into 12 hot-encoded columns, The time_signature column was converted into five hot-encoded columns, the decade columns were converted into six hot-encoded columns and imputed gender was converted into a three hot-encoded

3.8 Looking at numerical predictor’s correlation with each other

Feature correlation is a statistical measure that indicates the extent to which two or more variables are related. For example, if two variables are perfectly correlated, then a change in one variable can be precisely predicted from a change in the other variable. We drop correlated features because they can result in bias in the feature importance and decrease the overall performance of the model.

We used the seaborn heat map and found two pairs of features that were correlated to each other:

1. 'duration_ms' and 'section' - With a correlation score of 0.89.
   We decided to drop section as duration_ms was much easier to understand and interpret than section.
2. 'energy' and 'loudness' - With a correlation score of 0.77.
   We decided to drop energy after looking at the following graph loudness was not only easier to understand but also appears to give us more insight into our data.

3.9 Split Train-Test data

Using our one-hot-encoded data, as we had previously decided we split our data into train with 85% of the data, and test with 15% of the data. The train dataset has 34,820 data points with 34 features, and our test dataset ended up with 6,145 data points with 34 features.

3.10 Scale Data

Scaling the data will generalize the data points. Without scaling our data the model, we train will most probably not be able to give correct weight to the correct feature and we risk over fitting. We used sklearn’s StandardScaler() method to standardize our data, this means that the mean of our data is zero and the standard deviation is 1. We will fit the scale on the training dataset then use it to transform both train and test datasets.

4 Model Exploring - Single Classifiers

In this section we will train and fine-tune various models to achieve the best prediction on our test dataset.

4.1 Choosing Base classifiers

We will use a brute force method to train and fine tune 3 different Classifier types:

1. **LogisticRegression**
   Logistic regression uses an iterative optimization algorithm, such as gradient descent, to find the values of the coefficients that maximize the likelihood of the observed data
2. **K-Neighboors**
   KNN is a non-parametric method, which means that it does not make any assumptions about the underlying data distribution. Instead, it uses the training data to find close-by datapoints and perform prediction based on them.
3. **DecisionTree**
   Decision trees use a tree-like structure to make predictions, where each internal node represents a decision based on the value of one of the input features, and each leaf node represents a prediction.

4.2 Choose Hyper-Parameters to explore

For each of these models, we will try various hyper-parameter tunning and report the results. The exact hyper-parameter attempts can be found in the notebook.

4.3 Brute force all model-parameter combinations:

Using a method we built, the function will iterate over any possible model-hyper-parameter combination and perform the following:

1. Update the model hyper-parameters based on the row instructions.
2. cross_validate the model and save the train/validation scores to an output df.
3. Fit the model on the entire train dataset.
4. Test the model using test data and save the test score to the output df.
5. Use predict_proba method on the test and use the y probabilities to get our auc_result (** We will expand in it later in this notebook)
6. Save the AUC score to the data frame.

Additional outputs we will write to the data frame: *Train/Test size*, *Feature number*, *Run time* (in seconds).

4.4 Compare model performances:

After running our cursor and saving the outputs into a data frame, we can observe it and sort it by the different outcomes.

4.4.1. The best "**Show off**" award:

The award for the model with the best **training** score (aka- mean accuracy):

Top 3 performing models based on Validation Score

	model_type	C	penalty	solver	max_depth	n_neighbors	train_score	val_score	test_score	auc_result	run_time
DecisionTreeClassifier_3	DecisionTreeClassifier	NaN	NaN	NaN	10	NaN	0.833099	0.769271	0.772986	0.826306	1.2278
KNeighborsClassifier_3	KNeighborsClassifier	NaN	NaN	NaN	NaN	15	0.791427	0.765968	0.768592	0.854944	7.6993
KNeighborsClassifier_2	KNeighborsClassifier	NaN	NaN	NaN	NaN	8	0.821080	0.765020	0.773963	0.842545	7.5521

The DecisionTree classifier excelled in this part. This is due to its high depth value which made it prone to over-fitting.

4.4.2. The best Accuracy award:

The award for best overall **test score** model (Based on proportion of correct predictions made by the classifier)

Top 3 performing models based on Test Score

	model_type	C	penalty	solver	max_depth	n_neighbors	train_score	val_score	test_score	auc_result	run_time
KNeighborsClassifier_2	KNeighborsClassifier	NaN	NaN	NaN	NaN	8	0.821080	0.765020	0.773963	0.842545	7.5521
DecisionTreeClassifier_3	DecisionTreeClassifier	NaN	NaN	NaN	10	NaN	0.833099	0.769271	0.772986	0.826306	1.2278
KNeighborsClassifier_4	KNeighborsClassifier	NaN	NaN	NaN	NaN	50	0.769802	0.761114	0.769731	0.863065	8.3572

For the actual test score, we can tell that our DecisionTree is starting to fall behind, making room for KNeighborsClassifier.

4.4.2. The smoothest AUC award:

The award for the model best able to distinguish between the two classes by plotting the true positive rate (TPR) against the false positive rate (FPR) across thresholds and calculating the area under the curve.

Top 3 performing models based on AUC Score

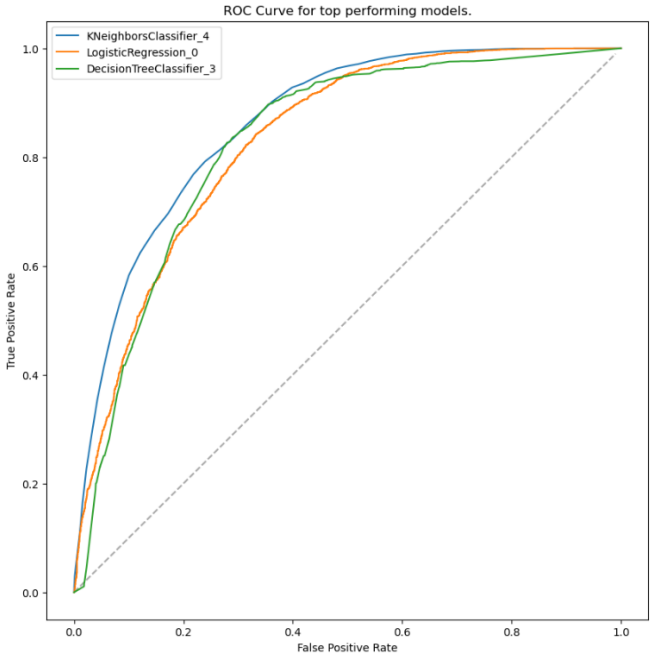
	model_type	C	penalty	solver	max_depth	n_neighbors	train_score	val_score	test_score	auc_result	run_time
KNeighborsClassifier_4	KNeighborsClassifier	NaN	NaN	NaN	NaN	50	0.769802	0.761114	0.769731	0.863065	8.3572
KNeighborsClassifier_5	KNeighborsClassifier	NaN	NaN	NaN	NaN	80	0.759994	0.754739	0.765663	0.861220	8.8367
KNeighborsClassifier_3	KNeighborsClassifier	NaN	NaN	NaN	NaN	15	0.791427	0.765968	0.768592	0.854944	7.6993

Interesting observation here is that unlike mean accuracy score, the AUC score appears to favor a more balanced results that relies on the average of 50 nearest points rather than merely 8 nearest points.

** Please note - ***From here on we will measure any future model's performance based on its AUC score***.

4.5 ROC Curve:

The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds. A high ROC curve is typically associated with a classifier that has a high true positive rate and a low false positive rate.



It is very visible how the KNeighbors can keep a balance ratio of TPR to FPR across all thresholds.

It's also clear that the Logistic regression model is performing poorly, which indicates that the Classification problem is NOT linear and cannot be resolved as such.

Confusion matrix for our best performing model

	P.Flop	P.Hit
Flop	1870	1150
Hit	265	2860

Our model appears to be biased towards predicting hits, which increases its False Positive Rate

4.6 Feature importance:

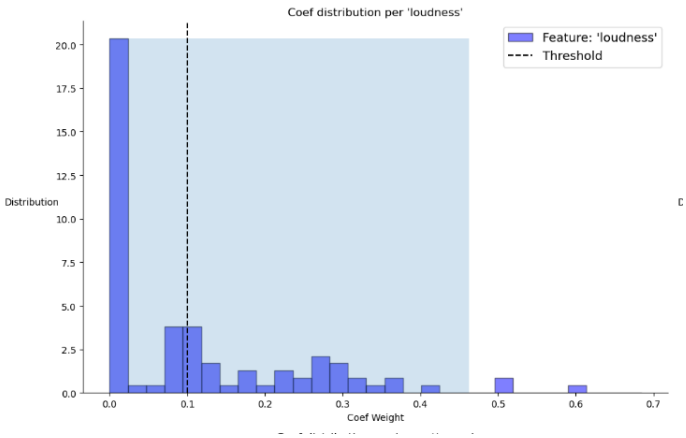
Feature importance is a measure of how useful each feature is for making predictions with a trained classifier.
Since KNeighborsClassifier doesn't provide a clear way to calculate feature importance, we will go to the 2nd best model type which is DecisionTree.
This classifier can provide a list of all coefficient weights by using the feature_importances_ property.

4.6.1 Bootstrapping:

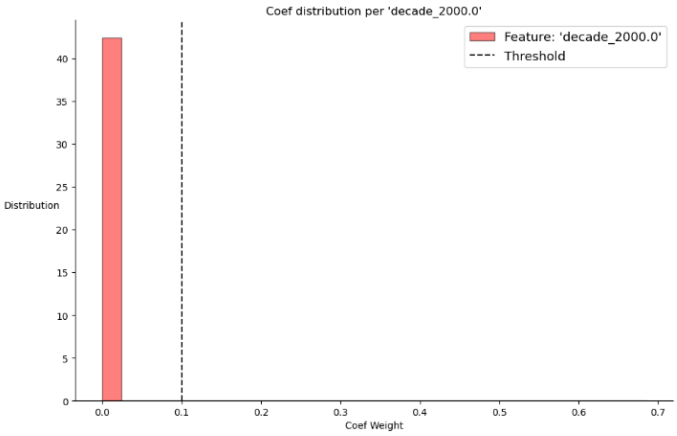
To reduce the randomness effect of the feature weights, we will repeat the fitting process of the model 100 times.
Each time we will:

- Create a randomly sampled training.
- Fit the model on the sample training data, extract and save the coefficient weights for each iteration
- Once completed, we will have a coefficient table with a 100-weight measure for each feature.
- We will then calculate the 5% upper and lower percentile of these weights.
- We will set a threshold of 0.01, removing each feature that doesn't reach this significance.

Example of feature with a high importance



Example of a feature without any significance



Once we finished the process, we were left with 16 features out of the original 38:

'mode', 'danceability', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'chorus_hit', 'is_missing_gender', 'key_0.0', 'key_5.0', 'artist_gender_Female', 'artist_gender_Male'

Note that the artist **gender, as well as the **is_missing_gender** features were marked as significant, indicating that we did the right thing to search for them on Spotify.

4.7 Building a new model from selected features*:*

Once we have reduced our dimensionality, we can train our winning model on the new dataset.

DecisionTreeClassifier_thin - Val: 0.764, Test: 0.761, AUC: 0.82

This did not improve our model's performance and we will need new methods for improving our score.

5 Further Model Exploring - Ensemble Method

Our next step is to examine different models with the Bootstrapping method.

Bootstrapping is a resampling method used to estimate the sampling distribution of a statistic by sampling with replacement from the original sample.

It is a way of simulating many experiments with the same sample size as the original data, but with different samples drawn from the original data.

5.1 Splitting train data*:*

For this method, we will not use cross-validation, but instead split our train data into train80 and val.

5.2 Boosting - AdaBoost*:*

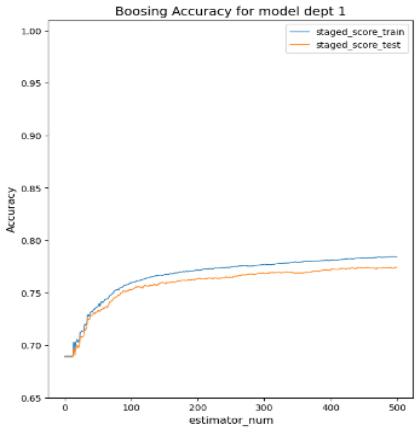
Boosting is a method used to improve the performance of a model by combining the predictions of multiple weak models. It works by training a sequence of weak decision trees, where each successive tree is trained to correct the mistakes of the previous tree. This is done by giving more weight to the examples that were misclassified by the previous tree, so that the next tree in the sequence will focus on learning to classify these examples correctly.

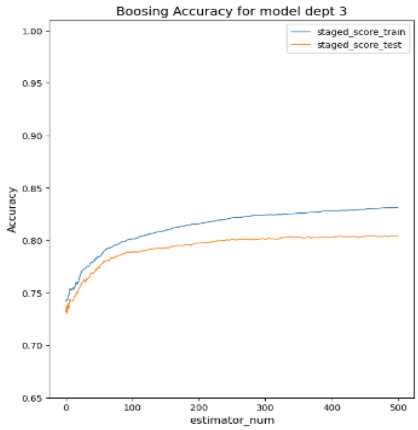
Due to the long computation time for this process, we could not fine-tune all hyper parameters at one run, and instead manually tested and agreed on:

Learning_rate = 0.05

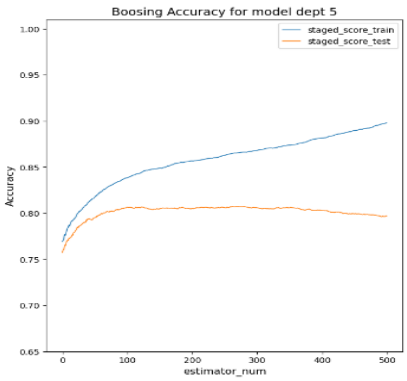
Algorithm: SAMME.R - *The SAMME.R algorithm typically converges faster than SAMME*.

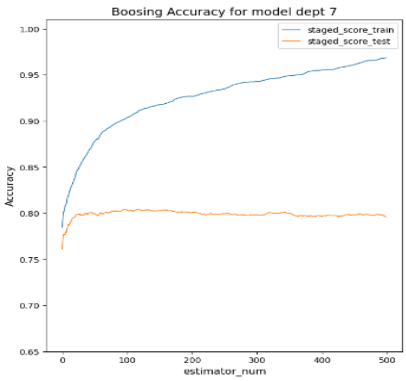
Since our data is large with over 40,000 records, we could try and use simpler trees that are still somewhat deep. We testes trees with depths of 1,3,5 and 7, and plotted their outcome:





For Boosting, it is recommended to keep the base trees less complex, otherwise we risk over-fitting as seen in the lower right plot where we attempted to Boost with a 7-depth tree classifier





Eventually, our best score of 0.804 was found for depth of 3, with n_estimators 462

5.3 BaggingClassifier:

Bagging (Bootstrap aggregation), method that involves training multiple models on different samples of the training data and then combining the predictions of these models to make more accurate and stable predictions.
The key advantage of bagging is that it can reduce the variance of the model and improve its generalization performance, which makes it a useful technique for improving the accuracy of many different types of machine learning models.
For our Bagging, we will set our base model as a DecisionTreeClassifier with a high depth of 20.

5.3.1 Training BaggingClassifier

We chose the DecisionTreeClassifier and not KNeighborsClassifier even though KNeighborsClassifier performed better on our single classifier test. This is because the KNeighborsClassifier takes a long time to fit due to its algorithm and bootstrapping it might take longer than our intended goal.

	train_score	val_score	test_score	auc_result	feature_n	run_time
BaggingClassifier_500_estimators_20_dept	0.9819	0.8076	0.8099	0.8859	38.0	156.8129

Since the model runs in a sequential way and each tree relies on its previous, training and validation time were very long, however, the result was a big jump from our single classifier models.

5.3.2 – Feature importance

Since Bagging trees use the same features with somewhat different sample data, the first feature of the cut (The one who produces the biggest 'gini' loss) will almost always be the same.

This could be seen when we observed each of our trees individually.

Since we are aggregating multiple tree predictions, we have the luxury of adding randomness to each of our individual trees. The bagging method takes away some of that potential randomness due to it way of choosing the leading features for the stub.

Top Count - Bagging	
instrumentalness	500
mode	0
decade_1960.0	0
key_10.0	0
key_11.0	0
time_signature_0.0	0
time_signature_1.0	0

Top Count - Random Forest	
instrumentalness	79
acousticness	69
danceability	66
loudness	39
valence	38
is_missing_gender	37
duration_ms	30
speechiness	26
time_signature_4.0	21
artist_gender_Female	21
artist_gender_Male	13
time_signature_3.0	12

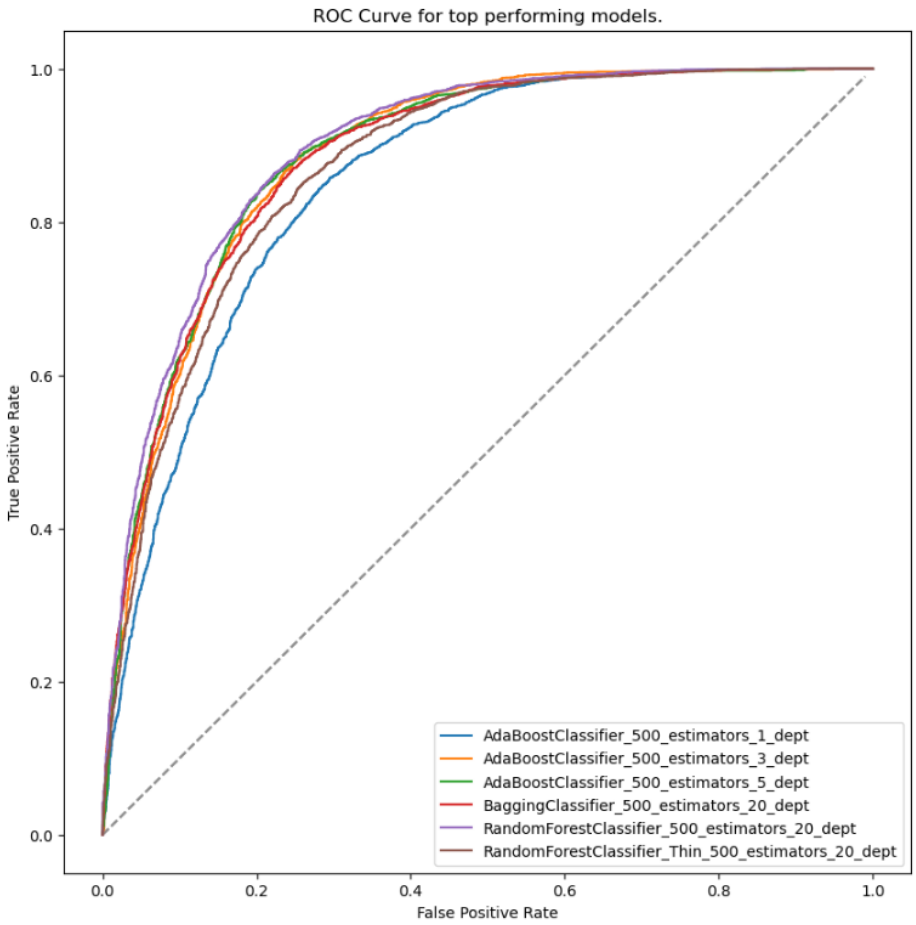
5.4 RandomForestClassifier

The RandomForestClassifier is the solution to the Bagging issue.
This method is very similar to Bagging, except for one key element which is the random feature selection. Each time the tree splits, we choose a random n number of features from our dataset and choose the strongest feature among this narrow list. That way, we can force our trees to have various stub features which will make them less correlated to each other.

Indeed, when inserting this element of randomness to the equations we can see that our result improved by about 1%:

	train_score	val_score	test_score	auc_result	feature_n	run_time
RandomForestClassifier_500_estimators_20_dept	0.9693	0.971	0.8194	0.8967	38.0	40.7143

5.5 Ensemble ROC Curve:



Looking at the ROC curve for all our Ensemble models, we can see a visible improvement from our single classifiers, in term of size of the curve, as much as the equal distribution across thresholds.

Confusion Matrix of Best Performing model

	P.Flop	P.Hit
Flop	2318	702
Hit	408	2717

Our ensemble models are much more balanced in term of predicting Hits and Flops

6. Comparing all Classifiers

Now that we have trained and tested all our models, we can compare them to each other:

	train_score	val_score	test_score	auc_result	feature_n	run_time
RandomForestClassifier_500_estimators_20_dept	0.969306	0.970994	0.819365	0.896674	38.0	36.5299
AdaBoostClassifier_500_estimators_5_dept	0.866097	0.807582	0.817575	0.888304	38.0	128.9548
AdaBoostClassifier_500_estimators_3_dept	0.830342	0.804566	0.814483	0.887052	38.0	103.5875
BaggingClassifier_500_estimators_20_dept	0.981943	0.807582	0.809927	0.885893	38.0	137.0323
RandomForestClassifier_Thin_500_estimators_20_dept	0.969522	0.791499	0.796094	0.872850	13.0	27.9354
KNeighborsClassifier_4	0.769802	0.761114	0.769731	0.863065	38.0	7.2283
KNeighborsClassifier_5	0.759994	0.754739	0.765663	0.861220	38.0	7.7779
KNeighborsClassifier_3	0.791427	0.765968	0.768592	0.854944	38.0	6.7714
AdaBoostClassifier_500_estimators_1_dept	0.784176	0.774268	0.781937	0.854002	38.0	47.0139

When we set on our way our goal was to find the best performing model which will provide us not only with good accuracy, but also with a balances False-To-Positive ratio across different thresholds.

After examining all various models and features, we have found that our most reliable model will be a **RandomForestClassifier** with a depth of 20 for each DecisionTree.

The model should be trained on the entire dataset (regardless of feature importance) and predict with all features.

7. Answering our original question:

When proving our result to our imaginary Record Producer, we will tell him the following:

We were able to build a model that will predict the probability of song being a hit or a flop with an accuracy of about 82%.

The model is very balanced regarding its output probabilities, meaning that it can be used to predict a hit with just as much accuracy as it can predict a flop.

The model will provide you with a probability from 0 to 1, 1 being an undouble hit and 0 being a total failure. It is up to you to decide what is your threshold for signing an artist.
If you have a limited capacity and are only interested in hiring a selected group of artists, you might want to set the threshold high (0.75 and above).
If you are interested in signing a mass amount of artists that are cost effective, you could lower your threshold under the assumption that not all purchased records will be hits.

An additional option is to use the model is to look for an artist that has over 2-3 songs with decent probabilities of becoming hits.

8. Data Fairness – Gender:

When collecting sensitive features such as Gender we are exposing our model to bias against a group.

When we examined feature importance, we noticed that being a Female provides a stronger probability of a Hit than being a Male. Does this mean that production companies should stop signing up Male Artists? Of course not!

Data provided based on gender criteria should always be taken with a decent amount of caution.

9. References:

Papers:

- https://journals.sagepub.com/doi/full/10.1177/1536867X20909688
- https://www.ibm.com/cloud/learn/random-forest
- https://medium.com/all-about-ml/bagging-random-forests-and-boosting-8c728e91a85d

Data Scraping:

- Spotify.com

Library assistance:

- SkLearn - https://scikit-learn.org/stable/
- Kaggle - https://www.kaggle.com/