Machine Learning Engineer Nanodegree

Capstone Proposal

Ben Walsh
January 24th, 2021

Proposal

Domain Background

Music recommendation algorithms have powered the growing dominance of audio streaming applications, despite constantly changing content and the inherently subjective nature of any art. While established artists and engaged users can provide sufficient historical data to enable a machine learning solution, there are challenges for an algorithm to predict the preferences for a new user or a new artist. Personal curation by a human is not scalable for a massive user base, so in order for online applications to efficiently scale while growing engagement with users, an automated algorithm is critical.

I have always been fascinated by the application of machine learning to music, and hope to use my newly acquired skills to explore this problem space. In particular I'm curious if song metadata provides enough information for reasonable predictions, instead of a more involved audio processing approach which would be more computationally intensive.

Problem Statement

Given a song-user pair, the algorithm should be able to predict whether a user will like it or not, as evidenced by a recurring listening event, i.e. listening to the song a certain number of times within a certain time window. The target value for a song-user pair will be 1 or 0, representing whether a recurring listening event occured. Overall algorithm performance will be evaluated on a test set of many user-song pairs, resulting in a single aggregate percent correct.

Datasets and Inputs

The datasets originate from a Kaggle competition: WSDM - KKBox's Music Recommendation Challenge

The primary training data contains:

- msno: user id
- song_id: song id
- source_system_tab: the name of the tab where the event was triggered
- source_screen_name: name of the layout a user sees
- source_type: an entry point a user first plays music on mobile apps
- target: the target variable. target=1 means there are recurring listening event(s) triggered within a month after the user's very first observable listening event, target=0 otherwise

Additional information on users are also available, which can be linked with msno (user id):

- msno

- city
- bd: age
- gender
- registered_via: registration method
- registration_init_time: format %Y%m%d
- expiration_date: format %Y%m%d

Additional information on songs are also available, which can be linked with the song_id:

- song_id
- song_length: in ms
- genre_ids: genre category. Some songs have multiple genres
- artist_name
- composer
- lyricist
- language

The primary information I plan on linking and expect to a performance driver is the genre_ids for each song. Secondary information I will explore and expect to increase performance are song:length, song:language, user:city, and user:age.

Solution Statement

A full solution to the music recommendation challenge requires data cleaning, feature engineering, feature selection, and algorithm implementation.

Data Cleaning

The required approach is mostly unknown until I explore the data, but as an example, I would fill in or remove any missing values.

Feature Engineering

I anticipate an unsupervised learning method such as k-means clustering will be helpful to categorize similar genres into a new feature: genre_group, which will have less unique entries compared to genre_id and should lead to better predictions.

Feature Selection

I will ensure only meaningful variables are input into an algorithm, removing variables such as IDs.

Algorithm Implementation

I will explore a few approaches to supervised learning with binary classification. I would like to start with a simpler, interpretable algorithm such as logistic regression. I will also compare this with a popular, more powerful, but less (directly) interpretable algorithm like XGBoost. My final solution will be based off of hyper-parameter tuning each approach and comparing the aggregate accuracy on a test set. I'll also compare the training times and interpretability.

Benchmark Model

As is typical for Kaggle competitions, there is a leaderboard displaying the top performing submissions and notebooks which can be upvoted for relevancy and usefulness.

The top score on an unknown test set is 0.747, with the top 10% scoring at 0.692 and a median submission of 0.671. While it is impossible to know how much experience the top submissions had and how much time they put into their solution, I hope I can at least generate a solution that would score in the top half, or above 0.671 on the test set. Specifically, I will compare my solution to benchmark performance by evaluating on a test set of many user-song pairs, resulting in a single aggregate percent correct.

Since the highest performing submissions have unknown approaches, the upvoted notebooks will be a proxy to benchmark models. This notebook has been upvoted and has a respectable score of 0.68138 (most are not public), which is above the median. The algorithm in the notebook is from LightGBM, a gradient boosting framework that uses tree based learning algorithms. Hyper-parameters in this solution to help compare to another tree-based approach are num_leaves: 108 and max_depth: 10.

Evaluation Metrics

My solution will be evaluted against the benchmark models by comparing the aggregate accuracy on a test set which the model has not trained on. In code:

```
num_correct = (test_predictions == test_truth).sum()
test_acc = num_correct / len(test_prediction)
```

Project Design

My intended approach to the capstone project entails data exploration, data cleaning, feature engineering, feature selection, algorithm implementation, and model evaluation.

Data Exploration

To start the project, I'll want to explore the data. Importing the source file into `pandas` and using `.describe()` is a good start to summarize each feature. Some visualization of feature distribution using `matplotlib.pyplot.hist` will also be helpful. As a result of this step, I expect to identify any features that need to be cleaned or removed.

Data Cleaning

After data exploration, I expect to identify features that need to be cleaned. The required approach is mostly unknown until I explore the data, but as an example, I would fill in or remove any missing values. With data imported as a dataframe in `pandas`, methods such as `.fillna` or `.dropna` will be useful.

Feature Engineering

In addition to the cleaned features, I expect additional engineered features will be valuable. For instance, genre_id is useful information, but clearly distinguishing each genre with individual IDs misses the intuition that certain genres are much more similar to each other than others. I expect an unsupervised learning

method such as k-means clustering will be helpful to categorize similar genres into a new feature: genre_group, which will have less unique entries compared to genre_id and should lead to better predictions. To implement the clustering, the library `sklearn.cluster` includes a `KMeans` library. I will explore a range of `n_clusters` to identify intuitive groupings.

Feature Selection

To finalize the feature input data, first the separate tables for training information with labels, song information, and user information must be joined. Song information will be joined with an outer join on song_id, while user information will be joined with an outer join on msno. Joins can be performed in `pandas` with `DataFrame.join`.

Additionally, only meaningful variables are input as features to train on. For instance the msno and song_id will only be used for joining and will be omitted from feature data, since the IDs are arbitrary and do not contain useful information for music recommendation. Dropping columns of data can be performed in `pandas` with `DataFrame.drop`, for instance with the argument `columns=['msno', 'song_id']`.

Algorithm Implementation

With clean feature data ready as input, I will explore a few approaches to the problem, which is a supervised learning, binary classification problem. I plan to start with a simpler, interpretable algorithm such as logistic regression which is available in `sklearn` under `linear_model.LogisticRegression`. I will also compare this with a popular, more powerful, but less (directly) interpretable algorithm like XGBoost, which is available under the `xgboost` library. My final solution will be based off of hyper-parameter tuning each approach and comparing the aggregate accuracy on a test set. I'll also compare the training times and interpretability.