SU2023 CS 667-VT / 767-VT Machine Learning

Final Project Report

Group# 4

Trenton Tidwell, David Rubey, Yoga Murugan

Project Partner	Distribution of Effort
Trenton	34%
David	33%
Yoga	33%

Introduction

Spotify Music Playlist Recommendation (Initial Project Ideas)

Initially, our team wanted to apply machine learning to Spotify music data. In 2018, Spotify, a popular online music streaming service, sponsored a machine learning challenge, http://www.recsyschallenge.com/2018/, with the Association for Computing Machinery (ACM) to generate new songs for existing playlists. In the spirit of this challenge, we planned to predict musical genres based on characteristics of songs. We also discussed the idea of generating playlists from songs with similar characteristics. In searching for sources of Spotify data we found the 2018 RecSys challenge's public Spotify dataset. This dataset included playlists of tracks represented by Spotify track id's. This dataset looked promising because the playlists on which the tracks were found could be used as the ground truth value. However, the track information was lacking. Only the Spotify track id was included in the dataset. The challenge expects users to download track characteristics using the Spotify API and the track id's found in the playlists dataset. This looked like a fun and interesting project to tackle. One problem that we faced was defining a quantifiable ground truth for each song. Would the ground truth value of a song be a list of every playlist on which that song is found in the dataset? How would that compare to another song that is found on a different number of playlists in the dataset? Are the ground truth values for those two songs equatable? As we started creating a proof of concept to download track information, including track lyrics, using the Spotify API we soon ran into a blocker that forced us to pivot from our initial project idea. We found a notice on Spotify's Web API documentation, https://developer.spotify.com/documentation/web-api/reference/get-audio-features, stating that data downloaded with the API should not be used to train machine learning models. After further research, we found verbiage in Spotify's Developer Terms, https://developer.spotify.com/terms, Section IV Restrictions, stating that using Spotify content for machine learning model training is a misuse of the platform and is restricted. After spending a significant amount of time thinking about project ideas using Spotify data, we found ourselves without the ability to download track characteristics to use as classifying dimensions for predicting the playlist ground truth.

Titanic Survival Prediction

From https://titanicfacts.net/, the Titanic RMS was a ship built in 1909 costing $7.5 million (~$236 million in todays terms). It was a luxury passenger cruise ship and a mail carrying ship (RMS=Royal Mail Steamer). The ship was carrying approximately 1317 passengers, of which, only 492 survived its sinking in the North Atlantic in 1912.

Our dataset comes from the Kaggle Titanic machine learning competition, https://www.kaggle.com/competitions/titanic. They provide the data in the form of two files; train.csv containing 891 training samples and test.csv containing an additional 418 test samples. Because this is a competition, they do not provide the ground truth values for the test portion of the dataset. This led us to the Complete Titanic Dataset on Kaggle, another source of the same dataset with all of the ground truths included, https://www.kaggle.com/datasets/vinicius150987/titanic3. The dataset contains characteristics about 1309 Titanic passengers that were on the ship when it sank. The data also includes the outcome for the individual passenger, a binary value indicating whether they survived or expired. The characteristics of the passengers that will serve as the dimensions for our predictive model are:

- **Pclass**: Ticket class (1st, 2nd, 3rd)
- **Sex**: Male or Female
- **Age**: Age in years (0.92 data point represents an 11 month old child)
- **sibsp**: Number of siblings aboard the ship
- **parch**: #of of parents / children aboard the ship
- **fare**: ticket price
- **cabin**: Cabin number
- **embarked**: Port of embarkation (C=Cherbourg, Q=Queenstown, S=Southampton)

The problem that we would like to solve is to predict an individual passenger's survival based on the other characteristics of that passenger. This is a binary classification problem.

Data Preprocessing

One-Hot Encoding

We transformed our categorical dimensions into numeric data so that it could be operated on by multiple machine learning algorithms. We used a one-hot encoding technique.

- **embarked**: Embarked_C, Embarked_Q, Embarked_S
- **Pclass:** Pclass_1, Pclass_2, Pclass_3
- **cabin**: deck_A, deck_B, deck_C, deck_D, deck_E, deck_F, deck_G, deck_even

    - deck_A -> deck_G indicates the vertical, A being the top and G being the bottom
    - deck_even -> whether the cabin number is even or odd indicates which side of the ship, port/left or starboard/right respectively, the cabin is on

Other Preprocessing

- **cabin**: deck_num -> in addition to one-hot encoding the cabin dimension into deck level and cabin parity (deck_even), we parsed and stored the cabin number as a numeric ordinal value
- **Sex**: we transformed this dimension from the textual values (female/male) into a binary representation of (0/1)

Missing Data (NaNs)

We encoded some of our missing data using interpolation techniques to improve the accuracy of some models.

- **deck_even**: We turned NaNs in the cabin even/odd feature into 0.5 since it is between 0 and 1 and reflects our knowledge of the value. This however negates the feature's status as a categorical variable.
- **deck_num**: For, Cabin number (deck_num) we represented missing values with the midpoint of the range of values since that is a reflection of the likely position of the person on the boat. This is under the assumption that all decks have the same number of rooms of equidistant spacing (this is untrue). The range of this data is 2 to 121, so we replaced missing values with the midpoint of these values: 60.
- **age**: NaNs were replaced by the average age which is 29.88 for the full dataset
- **fare**: NaNs were replaced by the average Fare which is 33.29 for the full dataset
- **embarked**: NaNs were inherently ingested into the Embarked encoding as [0, 0, 0]

For comparative purposes, we will train models on both the full dataset with NaNs replaced as well as a reduced NaN-dropped dataset that has all rows that have NaN values in them removed.

Feature selection

We decided to drop the passenger name and ticket number from our dataset since it was not apparent how those characteristics would be useful in training the models.

Methods

N-fold Cross Validation

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
We used train_test_split to randomly split our datasets into train and test sets. We set the stratify parameter to handle imbalance in the distribution of target classes.

Decision Tree

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

The approach we took was to start simple and try one of the most widely used classifier algorithms, the decision tree. The hyperparameter involved with this model is max depth. We started with the most simple case, removing all of the rows with missing data and training the decision tree with the resulting dataset. We ran the predictive model against the test portion of our dataset and recorded the accuracy score. We re-trained the model with the same data modifying the max depth from 1 to 20 and recorded the accuracy scores. Next we trained the decision tree using the full dataset with missing data (NaN) values replaced using the data preprocessing scheme described above. Following the same methodology as above, we re-trained the model modifying the max depth at each iteration and recorded the accuracy scores.

Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Our next approach was to increase the complexity by using the random forest classifier. The random forest generates multiple decision trees and combines the output of all of them to generate the final output of the model. As before, we trained the model using a version of the dataset with all of the rows that have missing data removed. While tuning the max depth parameter of the random forest classifier from 1 to 20, we recorded the accuracy of the model against the test portion of our dataset. And again we re-trained the model using the full data set with missing data replaced using our data preprocessing scheme and recorded the accuracy while tuning the max depth hyperparameter from 1 to 20.

Random Forest with normalized data

With the random forest, we went a step further and introduced a normalized version of our dataset to see how the accuracy scores would be affected. We performed the same training sessions as above, modifying the max depth hyperparameter and recording results using the normalized versions of the full dataset as well as the NaN-dropped dataset.

Random Forest with PCA

At this point, we introduced principal component analysis into our random forest classification model. PCA takes our normalized dataset and transforms it into another space with a maximum number of features as a hyperparameter. Consistent with our previous training sessions, we started with a normalized version of our NaN-dropped dataset. Iterating over both hyperparameters (max depth and max number of PCA features) from 1 to 20, we recorded accuracy scores of our predictive model against the test portion of our dataset. Next we performed the same training sessions using PCA transformed data from the full dataset with NaN values replaced. Again we recorded accuracy of the model against the test portion of our dataset while varying the two aforementioned hyperparameters.

Histogram XG Boosted Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html This version of the Random Forest classifier has native support for missing data (NaN) values. With it, we introduce another hyperparameter, learning rate. We performed similar training sessions as before, however, we are now able to use the full dataset with missing data values included. We recorded accuracy against a test portion of the dataset while varying the learning rate and the max depth hyperparameters. We then performed the same training and test sessions with the NaN-replaced dataset. And again, we performed a training and test session on the full dataset with missing values included, while varying another hyperparameter, max leaf nodes from 2 to 10, along with max depth from 1 to 20. Taking that same scenario one step further, we varied the hyperparameter, max leaf nodes, from 1 to 47 white varying max depth from 1 to 19. One last XG Boost training and test session used the NaN-replaced dataset while varying the hyperparameter, max leaf nodes from 2 to 10, along with max depth from 1 to 20.

Logistic Regression

https://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html#logistic-regression-3-class-classifier Finally after all of the work in the forest with the trees, we began training sessions with the logistic regression classifier. We began training and testing with logistic regression using the normalized NaN-replaced dataset. We followed this up with the normalized NaN-dropped dataset. In both cases, we used default options for the logistic regression classifier in scikit-learn and recorded the accuracy result. Next we used PCA to transform our normalized NaN-replaced dataset and iterated over max number of PCA features while recording accuracy results. After that we compared logistic regression accuracy scores on the PCA transformed normalized NaN-filled dataset using different regularization parameters, L2, L1, None and Elasticnet. With the Elasticnet regularization, we also varied the L1 ratio parameter.

Results

Dataset	Best Classifier	Accuracy	Models Trained
NaNs included (m=1309)	Histogram XG Boosted Random Forest	82%	305
NaNs replaced (m=1309)	Random Forest	84%	16040
NaNs dropped (m=267)	Random Forest with PCA	87%	3462

Analysis

Histogram XG Boosted Random Forest performed comparably to Random Forest, so it seems to do a pretty good job with its missing data replacement scheme. Although 87% appears to be the best accuracy score, we should be aware that the NaN-dropped dataset (m=267) is significantly smaller, approximately 20% of the complete dataset (m=1309) size. We varied the max depth hyperparameter from 1 to 20 and it appears that the best performing decision tree and random forest models had a max depth from 3 to 9. The best performing PCA transformed model had a max number of features = 3, but we should be aware that this was on the NaN-dropped dataset. The best performing PCA transformed models on the NaNs replaced dataset had a max number of features = 17 to 18.