DSCI-633 Foundations of Data Science & Analytics Project

December, 2021

Title:

ANALYSING WATCH TIMES OF HOTSTAR USERS AND PREDICTING THE USER SENTIMENT

Team Members:

Name: Omkar Khanvilkar
Email: osk8557@rit.edu
RIT ID: 347001066

Name : Sujan Dutta
Email: sd2516@g.rit.edu
RIT ID: 782000242

Name: Pranav Nair
Email: pn1922@g.rit.edu
RIT ID: 359006909

Name: Varun Tandon
Email: vt9438@g.rit.edu
RIT ID: 572002978

DEVELOPMENT OF QUESTION/HYPOTHESIS

Over the past decade, we have seen an increase in viewership for streaming services across the globe. Sites like Netflix, Hulu, and Hotstar dominate the streaming market with the content that they provide. The common denominator across these sites is the userbase and the feedback they provide to get a better experience.

Hotstar is a platform that has over 100 million user count, and more than 35,000 hours of content accessible over a variety of genres. As is the case with most major streaming services, using the sheer scale of users and content to create tailor-made recommendations and content for users generates a lot of value for Hotstar. Thus comes our problem statement.

How does a service like Hotstar capture its users' sentiments? Regular questioning often breaks immersion and can cause a biased answer. To approach this problem, we look at an alternate method, deducing the user sentiment by analysing the user behaviour.

The data the user generates is watch time in seconds, over multiple parameters. The users were also asked if they had a positive or a negative sentiment based on their overall experience. Our goal is to find the correlation between these parameters and the sentiment.

DATA RESEARCH

Upon searching for relevant data for the proposed problem, we found a dataset on HackerEarth that comprises of two files:

- *train_df.json*, which consists of watch times and the sentiment for 200,000 users.
- *test_df.json*, which consists of the sentiments for 100,000 users.

The data itself consists of 7 unique columns:

- *id:* A unique identifier variable.
- *titles:* Titles of the show watched by the user along with the watch time.
- *genres:* Names of the genres watched by the user along with the watch time.
- *cities:* Names of the cities the user was present in along with the watch time.
- *tod:* The time of the day in hours the user began watching along with the watch time.
- *dow:* The day of the week the user began watching along with the watch time.
- *segment:* The sentiment of the user (as positive or negative.)

The data itself is open to the public, present on HackerEarth.

LITERATURE REVIEW

The topic of customer (user) classification and segmentation has gained a lot of research attention in the last two decades because of the emergence of multiple OTT platforms. It all started with the release of the famous Netflix dataset [1] that contains 100M movie ratings in 2006. Since then many researchers have used this dataset to build movie recommendation systems and customer segmentation models. However, our dataset is a much newer one and it comes from Hotstar, a different OTT platform. While doing literature survey we found that there exist many works that use techniques like matrix factorization, nearest neighbors approach [2], clustering approaches [3] to recommend movies (or products), and perform customer segmentation. In this work, however, we are treating the problem as a classification problem as the data contains a label denoting the segment (positive or negative) corresponding to a customer. Treating the problem as a classification task will allow us to use the popular tree-based algorithms like Random Forest and XGBoost.

ANALYSIS: STRATEGY

There were two main issues faced with this data set. Primarily, the data was purely categorical in nature, in the form of "*value*:*watchtime*", separated by commas. Preprocessing this kind of data tends to be pretty time consuming and heavy on the computational load.

To resolve this, the preprocessing was done on the json files itself, that allowed the local systems to utilize the most of their computational power without wasting computational time.

The preprocessing itself was to create a function that parsed through the row of a single user, separate the value and the watch time, and add the value as a column with the user as the row, and the watch time as the data for that cell. With each user parsed, common columns were merged together, and a sparse data frame was created, with every single value type as a new column.

The second issue faced was the heavy class imbalance. The negative sentiments overshadowed the positive sentiments with a 23:2 ratio. With such a heavy imbalance, the models were sure to overfit and predict the incorrect class.

The solution proposed for this issue was to oversample the positive class and undersample the negative class. The oversampling technique used here was SMOTE. After doing so, the ratio of the negative class to the positive class reduced to 5:2.

The main features used for the final dataset were:

- cities
- gernes
- titles
- dow
- tod

Upon further analysis, the following assumptions were made based on EDA:

- The cities with users of positive sentiment did not follow the trend of the cities with users of negative sentiment. Furthermore, there were more cities with negative sentiment than positive.
- Cricket was the highest watched genre, but there was no other sport in the top ten most-watched genres.
- The most-watched hour of the day was 9pm, followed by 10pm
- The most-watched day of the week was Saturday, followed by Friday. This indicates that the users tend to watch on weekends rather than weekdays. This also corroborates with the time of the day analysis; Users watch on weekend nights, but not on Sunday nights.

ANALYSIS: CODE

After preprocessing, the final number of columns ended up being 1448. To reduce this number, feature engineering was performed on the dataset.

Each feature was approached in order with the following results:

- Genres:
  - All sports (except for cricket) were combined into a single column, with the watch times being summed.
  - A *genre_count* feature was added to keep track of the total genres watched by the user.
- Cities:
  - Only the top 15 cities based on watch time were added to the final dataset.
- dow:
  - Two new columns were added: *total_watch_time*, to keep track of the user's total watch time of the week, and *count*, to keep track of the number of days the user has watched for.
- tod:
  - Only the top 5 hours based on watch time were added to the final dataset
  - An additional column for the total hours watched by the user.
- titles:
  - An additional column for the total number of titles watched by the user

The final data frame consisted of 78 unique features excluding the target after the engineering. The over and under sampling was done to this data frame.

Next, the four models were tested on this data frame. Each model was trained without any tuning and then validated on a validation data set. Then the model underwent hyperparameter tuning and was validated on a validation data set.

These were the final results after the tuning:

- Logistic Regression:
  - Without Hyperparameter tuning: 0.78020
  - With Hyperparameter tuning: 0.78029
- Decision Tree:
  - Without Hyperparameter tuning: 0.6204
  - With Hyperparameter tuning: 0.6682
- Random Forest:
  - Without Hyperparameter tuning: 0.8068
  - With Hyperparameter tuning: 0.8056
- XGBoost:
  - Without Hyperparameter tuning: 0.80
  - With Hyperparameter tuning: 0.8119

All results above are indicative of the AUC ROC score.

Thus, XGBoost was the model chosen to predict on our test data as it had the highest roc_auc_score amongst the four.

The next step was to deploy the model, using a python library, "*streamlit*". This is an open-source framework used for building web apps for machine learning and data science models.

The UI of the web app allows input of the user number, and can predict the sentiment of the user.

WORK PLANNING AND ORGANIZATION

The entire workload was split equally amongst the four team members. Each part of the project was compartmentalized and broken into 4 equal parts, i.e., the preprocessing, EDA and model engineering was tackled by all 4 members.

IMPROVING TEAMWORK AND COLLABORATION

As mentioned above, the workload for each phase of the project was divided equally. For the preprocessing phase, we came up with the concept of parsing the first 100 rows of the data frame, and then decided to preprocess on the json file itself.

For the EDA, I worked on the tod feature, i.e., the time of the day feature. I analysed what trends were prominent in the dataset and how the watch time affected both the other features as well as the segment.

For the training, I focused on the Random Forest Classifier. I trained a model with default parameters to establish a baseline and then performed hyperparameter tuning on the RF model. The tuning however resulted in a 0.0012% loss, which was chalked up to error.

Overall, there was equal contribution for each segment, including the presentation, and the final notebook itself. All four members of the group worked on collaborating on the notebook with their individual findings.

INDIVIDUAL CONTRIBUTION

I majorly contributed to the below four aspects of the project.

1. **Data Preprocessing**
    - The training data in its original form was very unclean. Initially there were five features in the data - *genres*, *cities*, *day of the week (dow)*, *time of the day (tod)* and *titles*.
    - The values for every feature type consisted of many pairs where every pair consisted of the feature subtype and a number that represents the time in seconds for which that feature subtype was watched.
    - In order for the features to consist of only numerical values, we decided to split every feature type by creating a new column for every subtype in that feature and whose column values would be the watch time for that feature.
    - The above strategy was applied for the *genres*, *cities*, *day of the week (dow)*, *time of the day (tod)* columns using which we represented every column as a separate dataframe consisting of 35, 1358, 7 and 24 columns respectively.
    - Treating the *titles* column in the same manner would have resulted in a huge number of columns (~2000). Hence, for the *titles,* we selected the top 24 titles having maximum user watch times and represented them as a dataframe.
    - Finally, all these dataframes were combined together to get the main dataframe.
    - Since the size of the data was huge, I performed the preprocessing in two parts.
        - In the first part, I executed the code on the first hundred observations, so that we could ensure that we were getting the desired output. This code was implemented using Pandas.
        - But the same code would take a much larger execution time when trying to execute on the entire dataset. The main reason for this being that Pandas does not support multiprocessing. Hence its execution time increases as the size of the dataset increases.
        - Hence, in the second part, I preprocessed the entire data using dictionary and list comprehensions and the json_normalize module in Python. This reduced the execution time to a large extent compared to Pandas.
2. **Trying Vanilla Classification Models on a reduced set of features:**
    1. Since we had 1448 features in our dataset after preprocessing, I tried reducing the number of features by considering only the columns in the *day of the week (dow)* dataframe, since they contained the total watch time of the users within only seven columns.
    2. In addition to these, 5 new columns were added for representing the *total genres*, *total cities*, *total hours*, *total days* and the *watch time frequency* for every user. So, in total we had 12 features.
    3. I then applied Vanilla classification models like Logistic Regression, Decision Tree, Random Forest and XGBoost on these features and got the below results for the roc_auc_score.

Vanilla Models (Without hyperparameter tuning)	ROC_AUC_Score
Logistic Regression	0.63
Decision Tree	0.54
Random Forest	0.63
XGBoost	0.64

4. It can be seen that the results using this approach were a little better than random guessing and needed improvement by performing better feature engineering.

3. <u>Documentation</u>

- All the code in the Jupyter notebook was formatted and documented according to the PEP-8 format.
- A README file was added for the Project description.

4. <u>Deployment using Streamlit</u>

. An application to deploy the model using Streamlit was also created. It works as follows:

- A user id for the user needs to be inserted in the text box provided. This user id must be one of the indices present in the preprocessed test data.
- The application would then fetch the feature values for this user and make a prediction on these values using the XGBoost model. These procedures would happen at the backend.
- A feature label would then be assigned to this predicted value and displayed on the front end.
- While trying to run this application, it is recommended to create a separate new user environment in Anaconda.

REFERENCES

[1] J. Bennett, S. Lanning, et al., “The netflix prize,” in Proceedings of KDD cup and workshop, vol. 2007, p. 35, New York, NY, USA., 2007.

[2] G. Takács, I. Pilászy, B. Németh, and D. Tikk, “Matrix factorization and neighbor based algorithms for the netflix prize problem,” in Proceedings of the 2008 ACM conference on Recommender systems, pp. 267–274, 2008.

[3] P. Q. Brito, C. Soares, S. Almeida, A. Monte, and M. Byvoet, “Customer segmentation in a large database of an online customized fashion business,” Robotics and Computer-Integrated Manufacturing, vol. 36, pp. 93–100, 2015