Semester Project - Anime Genre Prediction from Synopsis

Kartikeya Jain, Akhil Dooliganti, Yash Trivedi - Team 10

8 December 2021

Abstract

Anime refers to any hand-drawn or computer animation from Japan. In this project, we try to predict the genre of each anime based on their synopsis. We use an API from MyAnimeList, a popular online anime and manga community and database, to fetch the data. We compare the prediction accuracy of three models: Logistic Regression, Linear Support Vector Classifier, and Random Forests.

Introduction

Anime refers to any animation originating from Japan. While the Japanese people refer to all animations as anime (a shortened word for animation), the people outside Japan refer only to the animation originating from Japan as anime. In this project, we will be looking at the MyAnimeList database and try to predict the anime genre from their synopsis. We access this database by making API calls as described in the API documentation.[1]

While MyAnimeList is similar to IMDB, where people can rate anime, create watchlists, and write reviews, users can create communities, discuss different animes and other shared interests. It also allows the user to do the same for Mangas, which are Japanese comics. The API can be used to collect data about animes and mangas, update users' watchlists, interact with the website's forum, and can also be integrated with other applications. The MyAnimeList API can be used freely for academic purposes. The API be accessed using bearer tokens generated using OAuth for authentication.

While the API can be used to query a variety of information about each anime, for the purpose of this project, we will only be querying the following fields:

- id: A unique identifier for each anime
- title: Name of the anime
- synopsis: A brief description of the anime
- genres: A list of genres to which the anime belongs.

A complete list of fields and examples can be found in the API documentation.

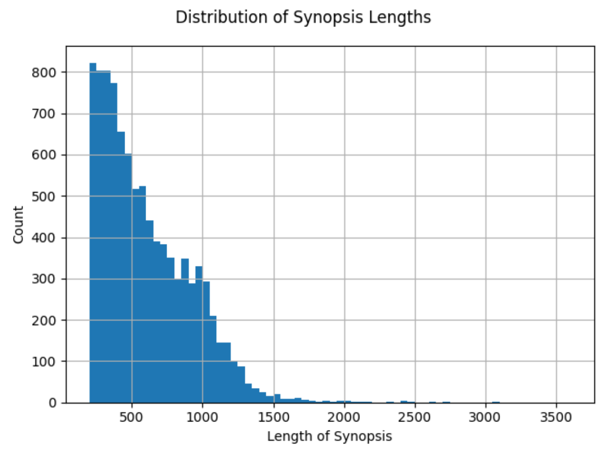

Figure 1: Distribution of Synopsis Lengths

The database assigns a unique integer id to each anime. We fetch the API responses for ids between 1 and 40000 and save the response as JSON files for the requests for the valid ids, with synopsis lengths of at least 200 characters, which leaves us with 9517 entries. At the time of writing, most ids after 40000 are for to-be-released animes, and they lack synopsis and genre information. The complete list of genres and their explanations can be found at https://myanimelist.net/anime/genre/info. In our data, we have a total of 43 genres. Figure 1 shows the distribution of synopsis lengths in our data.

The data distribution across various genres varies greatly, with Comedy, Action and Adventure being the most common genres.

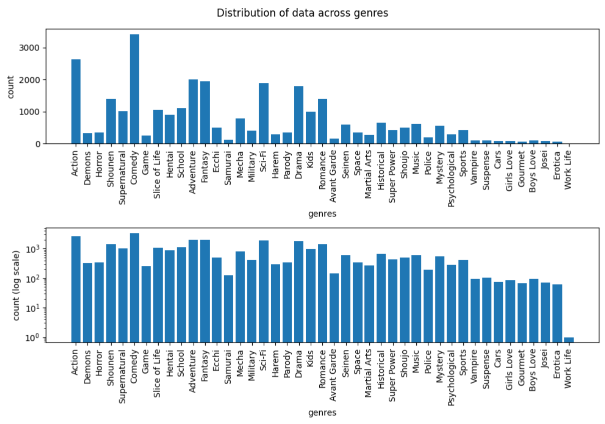

Figure 2: Anime Distribution across genres

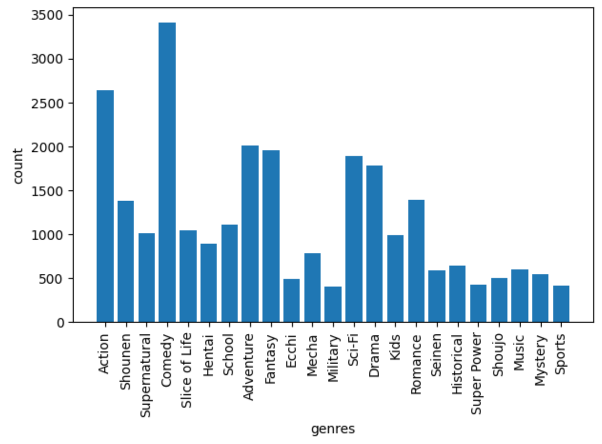

Figure 3: New Anime Distribution across genres

Since we already have labels (genres) available and we will be using them at the time of training our classifier models, our problem is a supervised problem. The problem is not as straightforward, as each entry can have multiple genres. The following section explains the methods used to clean the data and subsequently use it to train the classifier models. In this project, we will be using scikit-learn[2] and nltk[3].

Methods

Cleaning the data

Like any data science problem, we will begin by removing duplicates and handling any missing data. Each anime has a unique id in our data, and there are no duplicates to be removed. We found that some of the entries did not have the genres list populated and decided to drop these entries, reducing our entries to 9494.

One-Hot encoding

One-Hot encoding is a method to convert categorical data into features where the presence of a particular category is denoted by one and absence is denoted by zero.

We use one-hot to convert the list of genres into columns for each genre. Since the data distribution across various genres is imbalanced, we will drop the genres with less than 400 entries.

The genre tags in the database are assigned by the users, and most times, the users only use the common genre tags like Action, Comedy, Romance, Demons, etc., instead of the more uncommon tags like Police or Vampire. As a result, the list of genres in the database for an anime do not contain all the genres that it belongs to, but only the common ones.

We drop any entries that have no ones left in the corresponding genre columns. Our new data distribution is shown in Figure 3.

A correlation matrix between the genres is depicted in Figure 4. We see some interesting correlations in our data. For example, we see that the following genre pairs occur together quite often:

- *Sci-Fi* and *Mecha*
- *Adventure* and *Fantasy*
- *Romance* and *School*

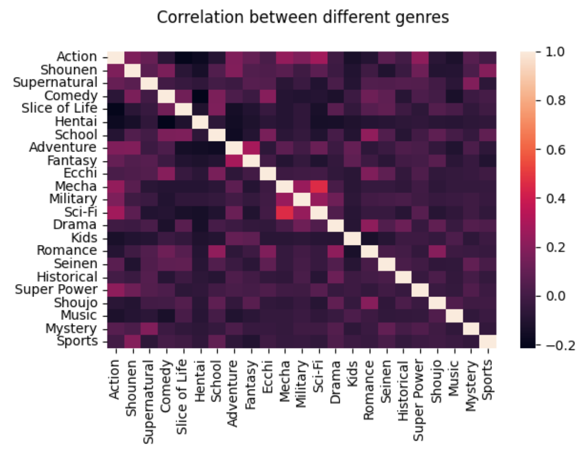

Figure 4: Correlation between different genres

Preprocessing the data

Allahyari et al. [4] discuss several classification, clustering, and extraction techniques for text mining in their survey. Based on this survey, as part of the data preprocessing, we perform the following steps:

Remove Source from the synopsis

Some entries have cited the original sources for the synopsis and have also given credit to people who wrote these synopses. These follow a specific predetermined format and can be removed using regular expressions. The sources and credit citations do not provide any useful information about an anime's genre and can thus be removed.

Replacing numbers

We replace any numbers/numerical words (e.g. 1, 1st, 14th, 1,000, 3.2) in our data with 'NUM' using regular expressions, thus reducing the number of features.

Stemming

Stemming refers to the process of removing any suffixes in a word, thereby reducing it to its root word form. Using stemming helps reduce the number of unique words in our data and find similarities between similar words. For example, the words *dance*, *dancing* and *dancer* can be reduced to *dance*.

We use the Porter Stemmer[5] from the nltk[3] library.

Tokenizing

Tokenizing a text refers to the process of breaking down a piece of text into small units like words or characters. N-grams refers to a continuous sequence of these units. An n-gram of size (value of n):

- 1 is called unigram
- 2 is called bigram
- 3 is called trigram, and so on.

When creating tokens, we can include n-grams of size more than one to retain the contextual information from the original text.

Removing Stopwords

Stopwords are words like 'a', 'an', 'the', 'that', 'it', 'him', 'her', etc. These words can normally be removed in text classification problems as these commonly used words do not provide any form of useful information and therefore cannot summarize the piece of text. The presence of stopwords can make it difficult to distinguish one text from another. nltk provides a list of stopwords for 11 different languages, including English, which we will be using in this project.

Count Vectorizer

One approach to convert these tokens into features suitable for machine learning algorithms is to convert each piece of text in our collection into a vector, which contains the frequency of each of the n-gram tokens. This approach creates a sparse vector for each piece of text, as it will mostly contain zeros for the token that do not occur in that text but occur in the collection of texts.

This approach enables us to use each token as a feature but has a fundamental drawback: even after removing stopwords, there might be tokens that are common across many pieces of text that frequently occur in the collection of texts but cannot help distinguish them from one another.

A common remedy to this drawback is to use Term Frequency and Inverse Document Frequency scores, as explained in the next section.

TF-IDF

TF-IDF stands for Term Frequency and Inverse Document Frequency, respectively. Term Frequency scores measure how frequently a term (in our case, an n-gram token) occurs in a document — the more frequent the term, the higher the Term Frequency score. The Inverse Document Frequency scores measure how often a term occurs across a collection of documents and penalizes the commonly occurring words. The product of the two scores is known as the TF-IDF score.

We use TfidfTokenizer from scikit-learn[2] to create a sparse document matrix representation of Term Frequency - Inverse Document Frequency scores for the unigrams and bigrams. During the process of tokenizing, we remove the stopwords as well.

Reducing the dimensions

Due to the large dimensions of the document matrix, we ran into memory issues when trying to reduce the number of features using Principal Component Analysis or Linear Discriminant Analysis as they require a large number of in-memory operations. We, therefore, decided to retain only the top 10000 tokens with the maximum term frequency to reduce the number of features. While many different values were tried instead of 10000, we found no noticeably improvements on increasing the number of features beyond 10000.

Training the classifiers

For this project, we build one-vs-the-rest classifiers for the following commonly used classification techniques as discussed in Allahyari et al. [4] and compare their performance by measuring their accuracy:

- Logistic Regression
- Linear Support Vector Classifier (Linear SVC)
- Random Forests

One-vs-the-Rest

One-vs-the-rest, also commonly called one-vs-all, is a technique where a classifier is trained for each class. These classifiers are essentially binary classifiers that classify an instance as belonging to that class (positive instance) or out of class (negative instance). Combining the results from these separate binary classifiers enables us to build computationally efficient discriminators. Interpretability is another advantage of using the one-vs-the-rest approach.

Logistic Regression

Logistic Regression is a basic binary and linear categorical classification model that uses a sigmoid function to output probability values for discrete classes. Using the sigmoid function ensures the values are bounded between 0 and 1. The sigmoid function is defined as follows:

$$P(x) = \frac{1}{1 + exp(mx + c)}$$

For binary classification, the instances with probability values of less than 0.5 are predicted as belonging to class 0 and for probability values greater than 0.5, the instances are predicted as belonging to class 1.

Linear Support Vector Classifier

Support Vector Machines (SVMs) are a set of powerful supervised learning algorithms which can be used for various classes of machine learning problems. SVMs can be used not just for classification problems, but also for regression and anomaly detection.

These versatile algorithms work well in high dimensional spaces and are memory efficient as they only rely on a subset of points called support vectors.

Linear Support Vector Classifier (Linear SVC) is an optimizer scikit-learn implementation of SVMs with a linear kernel and can be used in binary classification tasks.

Random Forest

Random Forest is an ensemble of multiple decision trees. Each decision tree is trained on a sample of

Table 1: Prediction accuracy scores for different n-gram combinations

	Classifiers		
n-grams	Logistic Regression	Linear SVC	Random Forest
1, 2	0.6854	0.7358	0.6502
2, 3	0.5269	0.6031	0.5101
1, 2, 3	0.6839	0.7415	0.6570

the training dataset, and the prediction results are averaged to combine the results. These average results can improve the predictive accuracy and overcome overfitting.

Like Logistic Regression, Random Forest can also provide class probabilities.

Calculating Prediction Accuracy scores

Instead of measuring the prediction accuracy by matching all the predicted genres with the actual list of genres, we will see if at least one of the predicted genres/genre predicted with the highest probability belongs to the list of the actual genres for that anime entry.

Since our data is imbalanced (as shown in Figure 3) and multi-labelled, techniques like stratifying the data and splitting it do not work. We, therefore, use k-fold cross-validation and average the scores over ten iterations.

Results

Table 1 captures the performance of various classifiers when using different n-gram combinations. We see that removing unigrams leads to a significant performance drop, and there isn't any significant performance improvement on using trigrams.

Table 2 summarizes the prediction accuracy scores for Logistic Regression, Linear SVC and Random Forest classifiers using K-fold cross-validation, where the value of K is 10. We observe that Linear SVC performs the best out of the three classifiers, correctly predicting the genre approximately 74% of the time.

Table 2: Prediction accuracy scores for different classifiers using K-fold Cross Validation

Classifier	Mean Acc.	Min. Acc.	Max. Acc.
Logistic Regression	0.6854	0.6756	0.7034
Linear SVC	0.7358	0.7255	0.7512
Random Forest	0.6502	0.6328	0.6553

Conclusions

In this project, we explored various classification techniques on a multi-label dataset. We compared the performance of these classifiers by using their prediction accuracy as a metric and found the Linear Support Vector Classifier to be most suitable. When calculating the prediction accuracy, we saw if the most probable genre based on our model was indeed one of the actual genres. Since the data has multiple labels, a possible extension of this work is to predict multiple genres for each instance.

We can extend the above analysis to various problems like predicting movie genres or book genres from plot summaries and can also be used in other multi-label problems. This approach can also be effective in document summarization tasks and labelling large amounts of unlabelled data.

References

[1] "MyAnimeList API (beta ver.) (2)," *MyAnimeList API (beta ver.)*. [Online]. Available: https://myanimelist.net/apiconfig/references/api/v2.

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] S. Bird, E. Klein, and E. Loper, *Natural language processing with python: Analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[4] M. Allahyari *et al.*, “A brief survey of text mining: Classification, clustering and extraction techniques,” *ArXiv*, vol. abs/1707.02919, 2017.

[5] M. F. Porter, “An algorithm for suffix stripping,” vol. 14, no. 3, pp. 130–137, Jan. 1980.