Course Review Sentiment Tagging using NLP Techniques

Payal Ahuja, Trupthi Hosahalli Chandrappa, Riya Simon

---

1. Introduction

1.1. Motivation

We were interested in exploring how we can gain more insight into students' feedback and understand the underlying sentiment from the colloquial language they use. Additionally, we also want to understand what factors students consider when reviewing a course or a professor (ex. nature of the professor, course usefulness, etc.) At the same time, we want to help the reviewed entity (professors, course designers) make sense of the large set of feedback and extract meaningful data to report and use for further decision-making or performance evaluation. By using NLP techniques to extract key phrases, and building a machine learning model to generate sentiment tags based on those phrases, we hope to gain a better understanding of the underlying sentiment in the feedback process.

1.2. Methodology

1.2.1. Datasets

We plan to scrape data from ratemyprofessors, a platform where students can leave reviews on their course instructors. We will use the review text and accompanying tags on the most rated professors at UIUC to train our model. Another dataset we are planning to use in conjunction is the Kaggle dataset on Coursera reviews, which includes reviews on over 600 different courses.

1.2.2. Methods (Packages Used)

- ➔ keras (building the LSTM model)
- ➔ sklearn (TF-IDF,linear model, preprocessing, cosine similarity, metrics, multiclass)
- ➔ ntlk (stem, corpus, tokenize)
- ➔ autocorrect

2. Implementation

2.1. Web Scraping

Our goal was to collect the reviews, the rating, and the tags attached to each review for professors at University of Illinois Urbana-Champaign. To do this, we had to first get a sense of how to find the links of every relevant professor. We were able to get a list of all the professors by using a "sid" value which we use as a query parameter in the url. After getting the right sid value for UIUC, we could fetch the names and id of all the professors by using the relevant API for professor list.

The output was a json file with a list of professor objects with fields like name, id, and department. We only selected professors who had more than 20 ratings. We used the names and tid to get the professor review page, from which we can scrape the reviews. From this page, we first loaded the page multiple times programmatically since there was a "Load More" button at the end of the page.

It was vital to do this step because we wanted to fetch all the reviews for the professors selected. If this step is not done correctly, it would only fetch the reviews on the first page. Then, we extracted the review text data, the numeric rating and the tags provided by RateMyProfessor. The class tags on the div were used to find the values since each review div had the same class. We also added the user selected tags to the csv file. Users are able to select tags from a set list to add to their review. These tags would be helpful to train the data to determine how the review affects the choice of tags. We fetched data for 150 professors and obtained approximately 1400 rows of data.

2.2. Keyword Extraction

2.2.1. Data Sampling and Cleansing

We considered two kinds of reviews for building our keyword extraction functionality. We used a subset of Coursera Reviews on a few courses provided in English and the webscraped RateMyProfessor reviews to build our classification model. This gave us a good training and test subset for building our model to perform keyword extraction. Size of the selection was limited to avoid long running scripts or processes and to get a good mix of both course review texts and professor review texts.

For cleansing, we kept it simple by removing all special characters and numericals leaving the alphabets. This involved removing html tags, if any, special characters etc. Once this was done, we tokenized(nltk.tokenize) the words, removed stopwords (nltk.corpus), performed spell check(autocorrect `Speller`) on these words and finally converted the text sentence into lowercase, before providing it for lemmatization. This was applied to the review text field of the Coursera data set and comment field of the RateMyProfessor data set. Also removed and possible duplicate review text to avoid unnecessary data for the model input.

In order to bring these two data sets to the same structure, we pulled them into a new dataframe with required fields and renaming them when necessary.

2.2.1.1. Lemmatization

It was important to lemmatize the data before we could feed this text field for keyword builder. We utilized nltk.stem's `WordNetLemmatizer` for this purpose. This helps us to use the root word whilst considering the context of the word.

2.2.2. Feature engineering

For feature building, one could slice and dice the data based on multiple variables. In some cases this list may be pre-built and in some cases we can customize this by building our very own feature list.

The dataset that we chose to utilize for training our model, needed some possible keyword/feature engineering, for each review. In order to train our model and derive more meaningful insight, we are obligated to build the domain knowledge of the data; this engineering helps us to create a better dataset which helps the model understand it well and provide reasonable results.

In order to build this for each review, we utilized the process of POS tagging and then building two word phrases, with a (Adjective, Noun) combination. Once this extensive list was built we eventually created a preferred feature list, by picking a few frequent phrases and purposeful keyword phrases for feature selection.

We availed the preferred feature list to do a final selection of keyword phrases for each review text which will then be used for our classification model.

2.2.3. Build and Train the model

Our dataset can now be called as a multi-label dataset, with the new keyword list created for each review text. We decided to use the binary relevance approach to transform our target variable, using MultiLabelBinarizer in sklearn. With the dataset, we were able to identify 972 keyword phrases that were common and interesting, hence we had a target variable of size 972.

We split all our data into training and test sets for training and evaluating our model's performance. We used the TF-IDF features for having the features created for the training and test data set, with a **custom** tokenizer that builds two word features/keywords from a review text or sentence.

Since we have 972 target variables, we will have to fit 972 different models with the same set of predictors (TF-IDF features), this definitely added processing times for use to build the model. The data size is not as big as we thought initially and also not small enough either, so we decided to keep it low-key, by using the Logistic Regression model as it's quick in training the model.

We used the OneVsRestClassifier class to solve our keyword extraction problem. Now, once the dataset was split, target variables fitted, features built with TF-IDF; it was time to build the model. We fit the model on the training set and then use the validation set

for prediction. Since the predicted values are going to be in binary, its necessary for an inverser_transform to actually display the predicted Keyword for our purpose. So we used the multilabel_binarizer's inverse_transform to obtain the values from the prediction.

We saved the model, tf-idf features and multilabel_binarizer variables in a .sav file, using pickle, to avoid multiple re-runs or increasing execution times for keyword prediction for our application.

2.2.4. Evaluation

For evaluation of the generated model, we used the sklearn's metrics class to calculate the accuracy of the model. We used the accuracy_score function to check on the validation dataset.

```
from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(yval[384], y_pred[384]))
print("Accuracy:",metrics.accuracy_score(yval[0], y_pred[0]))
print("Accuracy:",metrics.accuracy_score(yval[20], y_pred[20]))
```

```
Accuracy: 0.9955005624296963
Accuracy: 0.9966254218222722
Accuracy: 0.9966254218222722
```

2.2.5. Inference Function for Keyword extraction

We created a simple function that loads the generated model, from the pickle dump file that we saved, and then utilizes the saved parameters to now predict keywords for the text that comes from our UI.

To achieve this, we followed the below steps in our inference function:

- Clean the text
- Remove stopwords from the cleaned text
- Extract features from the text
- Make predictions
- Return the predicted keyword phrases

Since our process needs further finesse and is not as extensive as we want it to be, for the given project time, we worked with a reduced dataset, which could result in finding text from UI which may be far different, leading to inability of the model to predict any

keyword phrase at all, in such cases we would return a raw feature extraction from the given text for user to evaluate and choose from.

2.3. Sentiment Analysis

2.3.1. Building the dataset

Our goal was to not only identify the sentiment associated with each review, but also to determine the aspects that most contributed to the sentiment. In other words, our objective was to explore the various factors students might consider when writing a review.

Our chosen datasets used a 1-5 rating schema, with 1 being a poor score and 5 being an excellent one. Based on this, we added a column to our training dataset that mapped this numerical range to a qualitative range going from "strongly negative" to "strongly positive." Note that the data we scraped had so few neutral (3) values that we eliminated neutral reviews from our task. In total, we had 944 rows of data.

	A	B	C	D	G	H	I	J
1	professor	comment	rating	sentiment	strongly pos	positive	negative	strongly nega
116	Jason Anema	His lectures are very organize	4	positive	0	1	0	0
117	Jason Anema	Professor Anema is an amazi	5	strongly positive	1	0	0	0
118	Jason Anema	Superb and chill dude. Great	5	strongly positive	1	0	0	0
119	Jason Anema	Professor Anema is genuinely	4	positive	0	1	0	0
120	Jason Anema	I thought he wasn't very goo	2	negative	0	0	1	0
121	Jason Anema	I had him for Calculus I, and I	5	strongly positive	1	0	0	0
122	Lawrence Angra	The practice exam is not rela	1	strongly negative	0	0	0	1
123	Lawrence Angra	Amazing lecturer, among the	5	strongly positive	1	0	0	0
124	Lawrence Angra	SO overrated. Lectures bette	1	strongly negative	0	0	0	1
125	Lawrence Angra	He is an awesome lecturer! H	5	strongly positive	1	0	0	0
126	Lawrence Angra	The class in general was badl	5	strongly positive	1	0	0	0
127	Lawrence Angra	Second time taking a class wi	5	strongly positive	1	0	0	0
128	Lawrence Angra	This course is difficult, I can s	4	positive	0	1	0	0

Fig. 2.3a: a subset of values from sentiment_data with a column for each sentiment. Each comment and sentiment pair has either a 1 value (both values correspond) or a 0 value.

The ratemyprofessors interface allows reviewers the option of adding up to three relevant tags to their reviews, with examples such as "amazing lectures" and "participation matters." We used their built-in list of 21 tags as our aspect labels and scraped reviews that had at least one associated tag. For simplicity, we only considered the review's first tag. Therefore, each comment only corresponds to one aspect label.

	B	C	E	F	G	H	I	J	K	L	M	N	O
1	professor	comment	aspect	aspect_tag	skip class?	get ready t	hilarious	gives good	respected	amazing le	tough grad	inspiration	clear gradi
2	Gretchen A	Gretchen i	skip class?	0	1	0	0	0	0	0	0	0	0
84	Gretchen A	Gretchen i	get ready t	1	0	1	0	0	0	0	0	0	0
144	Gretchen A	She is reall	hilarious	2	0	0	1	0	0	0	0	0	0
166	Gretchen A	Her lecture	gives good	3	0	0	0	1	0	0	0	0	0

Fig. 2.3b: a subset of values from aspect_data with a column for each aspect label. Each comment and aspect pair has either a 1 value (both values correspond) or a 0 value.

We believe these qualitative labels, when combined with the aspect labels, build a clear picture of a reviewer's intent.

2.3.2. Classifying sentiment

The sentiment analysis portion of our task is a multi-label classification problem, where the machine learning model outputs a probability distribution across all sentiment labels. Therefore, we defined our output, y, as the data corresponding to each sentiment column.

```
y = sentiment_data[['strongly
positive','positive','negative','strongly negative']]
```

We defined the input to our model, X, as the reviews from the dataset. Note that before we added a review to the input dataset, we called our preprocessing function to clean the data (remove punctuation, extra spaces, etc).

```
X = []
sentences = list(sentiment_data["comment"])
for sen in sentences:
  X.append(preprocess_comments(sen))
```

After creating the input and output datasets, we converted their text inputs to vectorized word embeddings using the Global Vectors for Word Representation (GloVe) algorithm. We used the resultant embedding matrix as the weights to the embedding layer of our model.

The architecture of our sentiment analysis model includes a Long Short Term Memory (LSTM) network, a kind of Recurrent Neural Network commonly used for these types of classification problems. The output from the LSTM layer is used as the input for each of the four dense output layers, corresponding to the four sentiment classifications. We used the sigmoid activation function for these output layers, ensuring that each output predicts a value between 0 and 1 for their corresponding label.

```
def create_model(tokenizer, embedding_matrix):
  input_1 = Input(shape=(100,))
  vocab_size = len(tokenizer.word_index) + 1
  embedding_layer = Embedding(vocab_size, 100,
  weights=[embedding_matrix],trainable=False)(input_1)
  LSTM_Layer1 = LSTM(128)(embedding_layer)

  output1 = Dense(1, activation='sigmoid')(LSTM_Layer1)
  output2 = Dense(1, activation='sigmoid')(LSTM_Layer1)
```

```
    output3 = Dense(1, activation='sigmoid')(LSTM_Layer1)
    output4 = Dense(1, activation='sigmoid')(LSTM_Layer1)

    model = Model(inputs=input_1, outputs=[output1, output2,
    output3, output4])
    model.compile(loss='binary_crossentropy',
    optimizer='adam', metrics=['acc'])

    return model
```

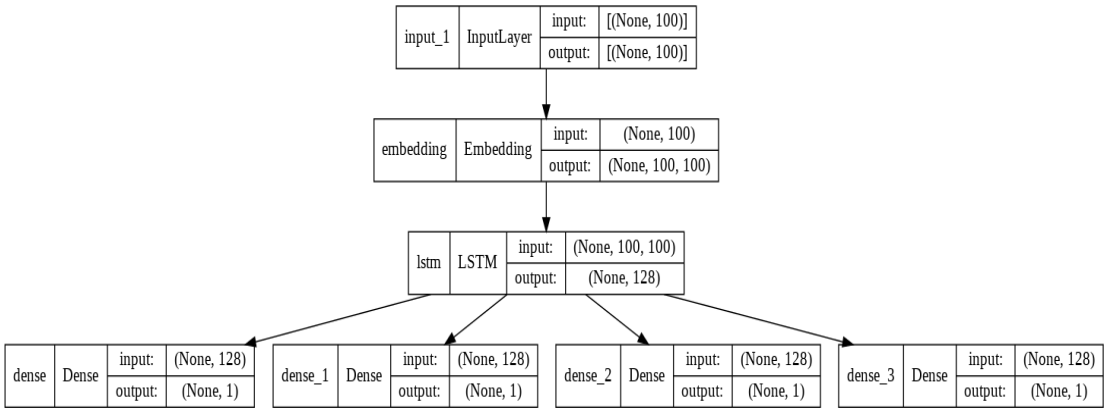

Fig. 2.3c: a plot showing the architecture of our model.

2.3.3. Understanding the aspects

We compared each label to the user input using a technique known as cosine similarity. To implement this, we treated each label, as well as each sentence of user input, as a unique document. Next, we converted the collection of documents into a matrix of TF-IDF features and applied the cosine similarity function on this matrix. From the cosine similarity values, we calculated the top three most similar labels.

```
label: do not skip class, input:  if you study that you will do well
['class' 'do' 'if' 'not' 'skip' 'study' 'that' 'well' 'will' 'you']

label: do not skip class, input:  there are few progets that really bring up your grade if you do poorly on tests
['are' 'bring' 'class' 'do' 'few' 'grade' 'if' 'not' 'on' 'poorly'
 'progets' 'really' 'skip' 'tests' 'that' 'there' 'up' 'you' 'your']


label: get ready to read, input:  she makes the class fun by telling jokes
['by' 'class' 'fun' 'get' 'jokes' 'makes' 'read' 'ready' 'she' 'telling'
 'the' 'to']

label: get ready to read, input:  she is very personable and easygoing
['and' 'easygoing' 'get' 'is' 'personable' 'read' 'ready' 'she' 'to'
 'very']
```

Fig. 2.3d: examples of TF-IDF feature matrix of combinations of labels and parts of user input.

	labels	input
doc_1	1.000000	0.087687
doc_2	0.087687	1.000000

	labels	input
doc_1	1.000000	0.068657
doc_2	0.068657	1.000000

Fig. 2.3e: examples of cosine matrix where doc1 is some label and doc2 is a part of the user input.

2.3.4. Verification

To verify the sentiment labels, we split our training dataset so that 20% of it would be used for verification. Upon evaluating our model, we found the test accuracy to be around 65%. This could be further improved after scraping more data and modifying the hyperparameters.

Since the aspect labels were a unique task to our project, the best way of verifying the accuracy of the top three tags was with human testers. In all of our test cases, we found that at least one of the returned tags was relevant to the user review.

2.4. User Interface

The application used to build the user interface for the application and deploy for users to test was Streamlit. It is an open source app framework built specifically for machine learning and data science projects. The package provides several functions to show headings, text, and markdown in the application. It also enables users to add visualizations, show data frames and add various widgets. For our application, we wanted to show the following information.

1. Show subset of dataset found from scraping RateMyProfessor
2. Show the relevant keywords found from the review
3. Show the sentiment rating (positive or negative)
4. Show the tags most relevant to the review. These tags are from RateMyProfessor and the tags will be predicted based on the content of the review.

We used several functions from the Streamlit package to build an easy and intuitive experience for users. An important issue we had to tackle was how to use pre-built models for prediction. Smaller files could be added to github and the application which is deployed on Streamlit Cloud would fetch them from the repository. However, larger files could not be committed to Github. These files were saved to Google Drive. These files were then downloaded from this location with the url being saved to a secrets file. This file is used by the server at startup. Large files and the data files were saved through a caching mechanism. This ensured that the application would be fast when handling multiple requests.

3. Results

3.1. Output

The application is deployed at https://share.streamlit.io/rps2ff23/courseproject/main. We used Streamlit Cloud to deploy this application using the public github repository as a source.

3.2. Future Work

When developing and working on this project, we had some ideas on how to further develop this application. The first addition would be a page for professors and other faculty members describing the tags most relevant to reviews about their course. This would enable them to assess what aspects of the course mean the most to students and what would improve their understanding of the topics taught in the course. Another feature would be to add a greater variety of sentiment tags that truly summarize the course. We used the list provided by Rate My Professor but feel that they can be improved and expanded. We can also scrape more data from Rate My Professor, taking data from a larger number of schools. This would help in training the model that generates the sentiment labels.

Members

Payal Ahuja

Worked on Sentiment Analysis Code and integration of this code with Streamlit

Trupthi Hosahalli Chandrappa

Worked on Keyword Extraction Code and integration of this code with Streamlit

Riya Simon

Worked on Scraping Rate My Professor Code and building Streamlit front-end application

References

1. https://github.com/Rodantny/Rate-My-Professor-Scraper-and-Search
2. https://realpython.com/beautiful-soup-web-scraper-python/
3. https://docs.streamlit.io