Using Natural Language Processing (NLP) to Implement Sentiment Analysis and Keyword Extraction on Yale Course Evaluations

Alexander J. Abinader

Advisor: Prof. Robert Wooster

May 2024

Abstract

In this project, we collect Yale course evaluations from the student run website CourseTable. Using these text reviews, we then perform two natural language processing (NLP) techniques: sentiment analysis and keyword extraction. The overall goal is to identify suitable NLP tools that can quickly and effectively summarize key course information for the Yale community. In theory, sentiment analysis would be used to calculate the percentage of students who recommend a course, while keyword extraction would be used to identify the key skills, strengths, weaknesses, and areas of improvement for Yale courses. The sentiment analysis portion includes several pretrained and newly trained models using a manually labeled dataset. The pretrained models prove extremely ineffective, however, the machine learning models perform quite well. These include random forest, logistic regression, support vector machine, and neural networks. Support vector machine (SVM) proves to be the most robust model, boasting an F1 score of 85.9% and 77.6% for the three-class and five-class datasets respectively. To better understand SVM's effectiveness, we also discuss some of the mathematical theory behind this machine learning algorithm. We then test several pretrained keyword extraction models, all of which produce unsatisfactory results. As an alternative, we create a ChatGPT API to handle keyword extraction on the Yale course reviews. This model performs extremely well and is fairly cost effective. Finally, we combine these sentiment analysis and keyword extraction methods to produce a proof of concept dashboard. This serves as an example implementation of what a real application of these NLP techniques would look like for the Yale community.

1 Background

Natural language processing (NLP) is a modern technology that allows computers to interpret and manipulate human language. The NLP field has experienced explosive growth in the past several years, mainly due to its vast number of important applications. Popular technologies like speech recognition, email filtering, chatbots, predictive text, and many others rely heavily on NLP techniques, making it a highly sought after skill for programmers and data scientists worldwide.

Two of the most popular NLP areas include sentiment analysis and keyword extraction. Sentiment analysis is an extremely valuable procedure that allows computers to evaluate written text through a human lens. For example, there exist many pretrained sentiment analysis models that have been used for a variety of popular applications such as evaluating Amazon product reviews or rating the sentiment of social media posts such as those on Twitter (now X).

Keyword extraction is a separate technique that automatically identifies the keywords which summarize the main topics of a written text. A similar process called topic modeling also seeks to extract the key terminology present in text but goes one step further to identify what underlying themes these words represent. Generally, in the context of shorter written pieces (a few words to a short paragraph), keyword extraction proves more effective at identifying terms of interest.

2 Introduction

At the end of each semester, Yale students are given optional surveys to complete where they review their classes for that term. Within these surveys are sets of questions that provide five multiple choice responses. For example, one of the most important questions is "What is your overall assessment of this course?" to which students can reply with "poor", "fair", "good", "very good", or "excellent". Notably, this type of question and its discrete set of responses can be converted into a numerical score. This facilitates the process of creating summary statistics for course evaluations using this quantitative data. A student founded, student run website named CourseTable does just that.

CourseTable is a course review website that serves the Yale community by providing information about Yale classes. In addition to official registrar data like class enrollment size, professor, location, distributional requirement, and others, CourseTable also uses the official end of semester surveys that students fill out in order to publish a host of descriptive statistics and reviews for Yale's thousands of courses. Namely, CourseTable uses the responses from the multiple choice survey questions about course overview, intellectual challenge, workload, and others in order to create three quantitative ratings: average course rating, average workload rating, and average professor rating.

In addition to multiple choice survey questions, Yale's official end of semester surveys also provide room for written responses, free from any constraints. This data is presented on CourseTable in its raw form. No analysis is performed on these textual responses. CourseTable simply lists the questions that were asked in the registrar survey and then prints the entire set of student replies. Analyzing this textual data is the perfect task for NLP.

3 Related Empirical Literature

Given the practical value of implementing sentiment analysis on student course reviews, other researchers have tackled this same problem using a variety of methodologies. In "Sentiment Analysis of Students' Reviews on Online Courses: A Transfer Learning Method", the authors use several of the same techniques we will apply in this paper [7]. These include BERT (neural networks), decisions trees, and support vector machine. Notably, the dataset contains over 20,000 observations which is greater than ours. Nonetheless, the authors were quite successful, achieving nearly 90% accuracy with their best model. Given the similar nature of our dataset and proposed sentiment analysis algorithms, this bodes well for our attempt at this NLP task.

More complex methods have also been applied to the classification of course reviews. In "Sentiment Analysis of Online Course Evaluation Based on a New Ensemble Deep Learning Mode: Evidence from Chinese", researchers attempt to train an extremely powerful model using deep learning techniques beyond that of a simple neural network [8]. Instead, they combine several models together in a process called ensemble learning which integrates a variety of learning frameworks. These researchers were quite successful, boasting an F1 score (an evaluation metric defined in Section 6) of over 91%. While this approach is quite advanced and beyond the scope of this project, some aspects of the data preprocessing are applied in our work such as word vector generation. Moreover, the use of deep learning is a positive sign for our attempt at utilizing neural networks in this paper.

A similar amount of research exists on the application of keyword extraction to course reviews. These papers focus on MOOCs (massive open online courses) which are favorable to our dataset given the greater number of observations and large pool of reviews that exist per individual course. Notably, our dataset suffers from each distinct course having a very small number of reviews. The models struggle to recognize keywords as these vary drastically between courses that have nothing in common. In “Application of keyword extraction on MOOC resources”, researchers apply both supervised and unsupervised learning techniques to the keyword extraction task [15]. As mentioned, the extreme difference in keywords between Yale courses makes supervised approaches highly impractical. Fortunately, these researchers also found success with unlabeled models using a graph based approach. We will implement several methods, one of which also utilizes a graph based framework.

Overall, the existing literature suggests that successfully implementing sentiment analysis and keyword extraction on Yale course reviews is highly possible, although limitations in the training datasets may prove insurmountable.

4 Data Collection and Cleaning

Collecting data for this project was a challenging task. After navigating the CourseTable website, we found a GraphQL page which essentially provides an interactive graphical user interface for querying data. The problem is that results were truncated to one thousand responses and individual courses could only be queried for their reviews one at a time. This meant that for a few hundred courses, we would have to manually query each one for three different sets of reviews and then copy paste this information to be consolidated later on. This would involve making almost one thousand manual queries and copy pastes.

After communicating with the CourseTable team about best approaches to collecting the data, they made it clear that there were no API keys available for CourseTable information. This meant there was no way to query their database directly using a programming script. The solution came in the form of special Python libraries which would allow us to make calls to the GraphQL page mentioned prior. Of course, this would require authentication which we could pass by copying our cookie signature from an active GraphQL page. In simpler terms, as long as someone was logged in to an active CourseTable session using an authenticated Yale NetID, this would have an associated key that could be used to authenticate calls from our Python scripts to the GraphQL endpoint. For now, this is a temporary solution as once a session ends (the page stays active for several days) that cookie signature becomes invalid and needs to be replaced with a fresh one.

From here we were able to query all of the data using only Python code and then perform some data cleaning in R. No more manual labor was required except for the occasional updating of our cookie signature for authenticating GraphQL requests. We first collected a set of classes that would be suitable for their reviews. We queried every class from the Spring 2023 semester that had an enrollment size of at least 30 students, was worth 1 credit, and belonged to Yale College. We requested as much information as possible to leave the door open for further data analysis down the line. This data included course id, course code, professors, average rating, professor rating, workload rating, enrollment size, and distributional requirement, among others. See Figure 1 for the exact GraphQL query.

Figure 1: GraphQL query for retrieving specific course listings.

```

{
  computed_listing_info(where: { season_code: {_eq: "202301"},
   last_enrollment:{_gt: 30}, credits:{_eq: 1}, school:{_eq: "YC"}}) {
    course_id
    crn
    all_course_codes
    course_code
    credits
    professor_names
    title
    average_rating
    average_professor
    average_workload
    average_rating_same_professors
    average_workload_same_professors
    school
    areas
    skills
    last_enrollment
  }
}

```

The next step involved cleaning our data and extracting some necessary information in R. After writing the data from Python into a CSV file, we read the table into R to make some modifications. First, we removed any observation that contained a NULL value to ensure that the entire dataset was comprised of complete cases. Next, we removed any duplicate course codes. For example, each query included a field called "course code" and another called "all course codes" meant to distinguish courses that are cross listed between departments. So, if our dataset included one observation of MATH 241 and another of S&DS 241 which are both Probability Theory, we only kept whichever code was listed first in the all course codes field and deleted the others. This ensured that data would not be duplicated and therefore have unfair weighting in our later analyses.

We continued the data cleaning process by dropping a few unnecessary columns, rounding the ratings to three decimal places, and creating new labels for each course that would permit further research in the sentiment analysis portion of this project. For simplicity, we removed any language classes (L1 - L5), classes that did not meet any distributional skill or area, and any classes that were dual listed for a skill or area. This allowed us to categorize the remaining courses into three groups: STEM, Humanities, and Social Sciences. After cleaning the data, we were left with 145 courses total.

The last step before collecting the course reviews and other responses was to find a unique identifier for each course. The GraphQL query for these replies did not permit course title or course code to act as a valid identifier. Fortunately, one of the fields included in our query was a course id, which was unique and functioned as a valid filter for future response queries. The last step of our data cleaning file was to extract these unique course ids and write them to a CSV for further use by our Python scripts.

Having obtained the course ids, the final step of data collection became trivial. We simply looped through each course id and made a query to the GraphQL endpoint with a specific question code in mind. We then wrote each response to one row of a CSV with the course id as an identifier. For example, if a course with id 92714 had 50 reviews, then the CSV file would contain 50 rows each with the id 92714 and an individual review. See Figure 2 for the GraphQL query that obtains student text responses. Additionally, see Table 1 for the three question codes and their associated survey prompts.

Figure 2: GraphQL query for retrieving student text comments.

```

{
  {
    evaluation_narratives(where: {{ question_code:{{_eq: {question_code}}} course_id: {{ _eq: {course_id} }} }}) {{
      comment
    }}
  }
}

```

Table 1: Question Codes and Descriptions

Question Code	Description
YC409	Would you recommend this course to another student? Please explain.
YC401	What knowledge, skills, and insights did you develop by taking this course?
YC403	What are the strengths and weaknesses of this course and how could it be improved?

After finishing the data collection, we were left with three CSV files corresponding to each of the questions asked in the Yale end of semester surveys. There were 4,522 observations for the recommendation prompt, 4,185 observations for the skills prompt, and 4,359 observations for the strengths and weaknesses prompt. These comments were now ready for any pretrained sentiment analysis and keyword extraction models. However, in order to train our own models, we would need to label these datasets from scratch, manually. We explain this process in Section 5.

5 Data Labeling

The first attempt at labeling involved simply going through the reviews dataset and categorizing each response as positive, negative, or neutral. We created a script to make the process more efficient and user friendly by providing a command line interface for labeling. Each review got printed to the command line with a prompt for what it should be labeled. Only one review would be printed at a time and the program could be quit at any time with our progress saved for later. The first round of labeling was more experimental and served as a good benchmark for evaluating the rule-based sentiment analysis models that will be discussed in Section 6.

Importantly, this was the first full scan through the dataset and there was no set of guidelines as to how labeling should be performed. Given the nature of Yale's reputation and the tendency of students to emphasize positive feedback, the resulting dataset was largely imbalanced. Roughly 72% of reviews were labeled as positive, whereas only ∼17.5% and ∼10.5% were labeled as neutral and negative respectively. Even so, there were still several hundred neutral and negative reviews which would suffice for the training tasks later on.

Three changes were made to the second round of labeling in order to maximize the accuracy and potential of our dataset. First, this would be the second time having looked through the dataset reducing any potential bias from the first time viewing. Second, the dataset was now categorized into five classes that would directly answer the prompt as stated: strongly recommend (SR), recommend (R), neutral (NEU), don't recommend (DR), and strongly don't recommend (SDR). Recall that the survey question clearly states "Would you recommend this course to another student?" so labeling on the basis of answering this question directly is preferable to a simple positive, neutral, or negative categorization. Finally, this round of labeling was performed with a more meticulous methodology including a set of guidelines to encourage labeling consistency. While the guidelines are quite comprehensive, they would span multiple pages if listed here. Please see Table 2 for a few select guidelines and corresponding sample student responses.

Table 2: Labeling Rules and Examples

Category	Rule and Example
SR	**Rule:** A yes or equivalent followed by a very positive explanation with words like "great", "excellent", etc. **Example:** "Yes! Fantastic introduction course to African governments and politics."
R	**Rule:** A yes with some explanation that is not overwhelmingly positive. **Example:** "Yes, if they have any interest in primates"
NEU	**Rule:** Some positive and negative aspects to the review with no clear answer to the question of recommendation. **Example:** "Not a crazy amount of work and the content is definitely going to be helpful for interviews but the exams were super difficult."
DR	**Rule:** Student is on the fence and says they would recommend other courses before this one. **Example:** "No, I don't know if it's necessary to learn about global health at such a basic level. It's better to take a more advanced class."
SDR	**Rule:** Student says they do not recommend the course and gives several reasons why with little to no positive exceptions. **Example:** "Not at all. Only take it if you absolutely have to for your major (shouldn't be a major requirement). Worst class I've ever taken."

After relabeling, the data was still very unevenly distributed across classes. The labels SR, R, NEU, DR, and SDR made up roughly 29%, 49%, 12%, 8%, and 2% of the observations respectively. Note that 2% is incredibly small but is expected given the propensity of Yale students to avoid harsh criticism. We observe that the dataset is still heavily comprised of positive reviews. In addition to this five label dataset, we also created a three label dataset by combing the recommends (SR and R) and the don't recommends (DR and SDR). This is equivalent to our original labeling of positive, negative, and neutral with the exception that it benefited from our revised labeling methodology. With the raw responses and labeled responses in hand, we were ready to tackle sentiment analysis using both pretrained models and models trained from scratch.

6 Sentiment Analysis

6.1 Pretrained Models

In the following sections, we will discuss three pretrained models which could act directly on our raw dataset. This is to say that these models do not require a labeled training dataset and can leverage their previous training and or rule-based implementation to classify our text reviews into sentiment groups. We will first explain how each of the models works and then present the results and assess their individual accuracy.

6.1.1 Valence Aware Dictionary for Sentiment Reasoning (VADER)

VADER is a sophisticated sentiment analysis tool that was designed primarily for social media excerpts [1]. With that said, we believed that the syntax of a Yale student's course review would largely resemble that of a social media post. VADER employs a unique lexicon that is comprised of a particular list of words and phrases. Each of these is assigned a sentiment score that reflects their positive, negative, or neutral valence. Unlike other models, VADER is slightly more advanced as it can also recognize the use of language in specific social contexts such as slang, abbreviations, and more. VADER then uses a composite scoring system that accounts for a weighted combination of individual word or phrase scores. The final output of a VADER model is measured on a continuous scale from -1 (very negative) to $+1$ (very positive) as opposed to a mere class label.

VADER's precision is furthered by a set of heuristic rules that can interpret more grammatical and syntactical nuances of language. This is particularly important for Yale course reviews. For example, VADER recognizes the significance of capitalization and factors this in to the intensity of a sentiment. Moreover, punctuation such as exclamation points can also drastically influence a sentiment score by bolstering whatever sentiment is at play. The algorithm also takes into account conjunctions such as “but” which can dramatically shift the sentiment of a sentence. Compound sentences, capitalization, and feisty punctuation are rampant in the course review dataset, so these features are extremely desirable. Overall, VADER is extremely adept at analyzing human sentiment and is one of the best rule-based, lexicon powered sentiment analysis models available.

6.1.2 TextBlob

TextBlob is a simple, yet versatile library offered in Python, which allows for a powerful set of NLP options in processing textual data, including sentiment analysis [11]. It utilizes a combination of pretrained machine learning models coupled with a lexicon-based approach.

Unlike VADER, TextBlob's sentiment analysis separates the polarity and subjectivity scores for a given text. Polarity scores range from -1 (very negative) to +1 (very positive) while subjectivity scores range from 0 (very objective) to +1 (very subjective). This allows TextBlob to not only assess the sentiment of a response but also how opinionated it is.

TextBlob utilizes part-of-speech tagging and noun phrase extraction to capture the context and grammatical structure of text. This allows it to understand more complex sentences and take into account word positioning, similar to VADER. Again, this is desirable for handling the nuances of Yale course reviews.

6.1.3 Robustly Optimized BERT Approach (RoBERTa)

RoBERTa is a modified version of BERT (Bidirectional Encoder Representations from Transformers) which optimizes its predecessor by refining the training procedures and data volume [14]. Unlike the simpler rule-based and lexicon-based methods of VADER and TextBlob, RoBERTa uses deep learning (neural networks) and a vast amount of training data to learn complex patterns and identify sentiment expressed in text. This is a much more powerful approach that allows it to capture the nuances of text beyond just capitalization and punctuation to include things like irony, sarcasm, and more.

RoBERTa's strength lies in its bidirectional understanding of language, which allows it to consider the full context of a word relative to words that came before and after it in a sentence; this is much more powerful than simply considering small phrases or groupings of words like the other models do. RoBERTa processes swaths of text data when training the neural network which allows for a deeper understanding of textual sentiment. There are many training datasets to choose from when running the RoBERTa model. After testing several, we chose to implement a training set comprised of Twitter tweets as these would be most similar in length, structure, and phraseology to a Yale course review. Unlike VADER and TextBlob, the RoBERTa model simply outputs a list of probabilities corresponding to the positive, negative, and neutral classes.

6.1.4 Results

In order to assess the accuracy of these pretrained models, we utilized the labeled dataset which included just "pos", "neg", and "neu" labels. We read the outputs from each model into R and then proceeded to classify each score into a label using several different techniques.

Recall that for VADER, the model outputs a simple polarity score between -1 and +1. We first plotted the scores to analyze the shape and spread of the distribution. Using that information, we then tested a variety of score cutoffs to represent our VADER labels. After a bit of experimentation, we found the best breakdown to be the following:

$$
\begin{aligned}
&\text{pos} = \textit{polarity} > 0.05 \\
&\text{neu} = -0.05 < \textit{polarity} < 0.05 \\
&\text{neg} = \textit{polarity} < -0.05
\end{aligned}
$$

After classifying the data using this breakdown, we simply calculated a percentage accuracy using our manually labeled dataset as the "correct" one.

We took a similar approach to assessing the TextBlob model except that we now needed to consider the subjectivity score mentioned prior. This meant that we needed to consider two different measures that affect accuracy. Like VADER, we tested a variety of different cutoffs for the polarity scores in order to achieve the best results. We factored in the subjectivity scores using the following formula:

$$\textit{sentiment} = \textit{polarity} \times (1 - \textit{subjectivity})$$

This effectively weights the polarity of a text based on how subjective it is. For example, perfect objectivity of 0 would simply make it such that $\textit{sentiment} = \textit{polarity}$ whereas perfect subjectivity of 1 would just make $\textit{sentiment} = 0$, signaling that the text was extremely difficult to score due to ambiguity. We then tested several cutoffs on the final *sentiment* score as we did for VADER.

Surprisingly, we found that *polarity* alone was actually more accurate than including *subjectivty* as part of the calculation. For that reason, we ended up calculating our final results using just the polarity score by itself with the same -0.05, 0.05 cutoffs that were used for the VADER model.

Finally came the RoBERTa model which was the simplest to assess. As the model just output probabilities for each of our classification labels, the most obvious approach was to just choose the label that had the highest output probability:

$$\text{label} = \max\{p_{pos}, p_{neg}, p_{neu}\}$$

Overall, the pretrained models did not perform well. Recall that our "pos" label comprised roughly 72% of the dataset, effectively setting the baseline for accuracy. For example, an algorithm that only output "pos" would be correct 72% of the time without employing any sensible strategies. We summarize the results in Table 3 and also list the model run time.

Table 3: Pretrained Model Accuracy and Efficiency

Model	Accuracy	Running Time
Baseline	72.0%	N/A
VADER	75.4%	30 seconds
TextBlob	64.8%	5 seconds
RoBERTa	76.7%	10 minutes

The results speak for themselves. We observe that TextBlob did a terrible job and actually achieved a lower accuracy than the baseline. The VADER and RoBERTa models achieved accuracies of just a few points above the baseline. Moreover, given the neural network approach of the RoBERTa model, we see that it takes roughly 10 minutes to label the entire dataset which is only 4,522 observations. If we had to choose, the VADER model clearly has the best balance of speed and accuracy, however, even its accuracy is virtually insignificant.

There are two main reasons for the failure of these pretrained models. First, they were all trained and designed to run on different types of text such as social media posts. More importantly however, is the simple fact that our dataset is not designed to perform strict sentiment analysis. Recall that the specific prompt we are working with in these course reviews is "Would you recommend this course to another student? Please explain." Note that this is a yes or no question with an optional explanation. When labeling the dataset, we observed many responses that were as simple as "yes" or "no" and for that reason the models often get these classifications wrong. In fact, we investigated the models and found that for answers like these they were more likely to give a classification of neutral rather than positive or negative. Fortunately, using our own labeled datasets and training models from scratch, we could easily capture these cases and achieve much higher accuracy than the pretrained models ever could.

6.2 Models

We may now leverage the training datasets that we manually labeled. In doing so, we could train models from scratch by utilizing cross validation, splitting the data 80-20 for training and testing respectively. In the following sections, we will provide a high level overview of the four models we tested. Then, we will present the results and assess model efficacy using a variety of metrics.

6.2.1 Random Forest

Our first method for sentiment classification was random forest. Random forest is an ensemble learning method that operates by constructing multiple decision trees during the training process and then outputting the class that was most frequently identified during classification of individual trees [5]. This is important as basing the results on many trees helps to prevent overfitting as well as using randomness to enhance model learning.

In our implementation, we begin by preparing the dataset for training and evaluation. This involves reading in the data, shuffling it, and creating the standard 80-20 cross validation split. This helps to test the model on data it has never seen which is important for generalizing the model's abilities to new scenarios and preventing overfitting.

An extremely important step is processing the text data such that it is viable for a machine learning method like random forest. Here, we use a hashing vectorizer which converts our text reviews into a numerical format as numerical data is necessary for machine learning algorithms. This hashes word occurrences while preserving the sparsity of the dataset which essentially means keeping track of word frequencies throughout our set of reviews. Such an approach is extremely useful for datasets with a large vocabulary.

After the data has been read and processed, a random forest classifier is ready to be trained. The training process involves learning the patterns and relationships between words in our reviews which are then used to predict the sentiment label of reviews in our testing set. We also record a variety of assessment metrics and record the time required for the data preprocessing, processing, training, and evaluation cycle. These will be discussed and compared with our other models in the results section.

6.2.2 Logistic Regression

The preprocessing approach is largely the same as for random forest. We read in the labeled dataset and apply cross validation to obtain separate sets of training and testing reviews. Equivalent to the hashing vectorizer, this program uses count vectorization which effectively converts the text into a matrix of token counts. These tokens are small text components which in NLP are usually single words. Thus, this process essentially converts the text into a numerical format representing word frequency.

There is also TF-IDF transformation (Term Frequency-Inverse Document Frequency) which is a technique that reflects the importance of a word to a document in a collection or in this case a single review to the entire corpus of reviews. This helps to adjust for words that are naturally more common. The components are calculated as follows:

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

$$IDF(t) = \log(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}})$$

$$TFIDF(t,d) = TF(t,d) \times IDF(t), \text{for a term } t \text{ in document } d$$

This approach to processing the data and constructing a feature vector has a number of benefits. It better enables the model to understand relevance and context of words. It also helps to reduce the dimensionality by eliminating features that have minimal or no informative characteristics [3].

Once features have been extracted, the program is ready to train a logistic regression classifier. While this machine learning method usually deals with binary classification tasks, we can set the relevant parameter to "multinomial" and the solver to "lbfgs" in order to extend logistic regression to classification with several labels. In our case, this is 3 or 5 labels depending on our dataset. Finally, we extract the relevant evaluation metrics and record the total time of the model life cycle.

6.2.3 Neural Network

Next, we apply deep learning to our sentiment analysis task. Our program takes similar data preprocessing steps. In this case, a tokenizer converts the text into a sequence of integers that represent specific words in a dictionary. This is quite similar to the hashing vectorizer and count vectorization methods mentioned prior.

The neural network uses a sequential model which is especially important for text classification tasks given that sequences of words are common and vary drastically in meaning. An embedding layer maps each word index to a dense vector which is followed by an LSTM (long short-term memory) layer. This LSTM layer is crucial as it captures the memory of prior elements. There is also a dense layer which uses the ReLU activation function and provides additional learning to the model. This is complemented by a dropout layer which randomly selects a subset of input units to negate, preventing overfitting. The final layer is a dense layer which uses the softmax activation function to output a probability distribution over the sentiment classes.

Our neural network was created using the keras library [4]. The model was compiled with categorical crossentropy as the loss function which is optimal for multi-classification problems. Deep learning involves a host of parameters that can be modified to enhance performance. These include number of LSTM units, number of dense layer neurons, dropout rate, number of training epochs, activation functions, and loss functions, among others. We experimented with a variety of combinations in order to yield the best results.

6.2.4 Support Vector Machine

As with logistic regression, feature extraction is performed by transforming the text reviews into a TF-IDF matrix. The same 80-20 cross validation split is used for creating the training and testing data. We are then able to train an SVM classifier which can be modified to support multi-label classification tasks. The same evaluation metrics are extracted as with our previous models.

The driving principle behind SVM is to find a hyperplane that best divides the dataset into classes. SVM performs well in high dimensional spaces which makes it extremely desirable for a task like sentiment analysis. This is because in text classification the feature space that results from vectorization can (and likely will be) extremely large. There is also versatility in choosing what kernel to use: these determine how data points will be separated. We discuss more in depth SVM theory in Section 7.

6.2.5 Adjustments

Two key improvements were made when preprocessing the data in order to achieve better results. This involved disabling and changing default behaviors for the packages and libraries used in all of our models. First, we disabled the "lowercase" setting which is the default case when vectorizing the data. Recall that these models have been designed to handle sentiment analysis which usually implies a simple positive, negative, or neutral label. The problem here is that we aren't performing sentiment analysis exactly but rather answering the question of recommendation with regard to Yale courses. Moreover, recall that one of our datasets was labeled such that it includes 5 classes as opposed to just 2 or 3. For this reason, the nuance of capitalization is extremely important and the programs were augmented to reflect this. For example, according to our labeling rubric as demonstrated in Table 2, the response "yes" receives a label of "recommend (R)" whereas "YES" receives a label of "strongly recommend (SR)". One unique characteristic of the Yale course evaluation dataset is that a sizeable percentage of replies are only one or two words. Thus, this distinction in capitalization is indispensable.

Second, we modified the regular expression (regex) pattern that determines how sentences are to be broken down into tokens. For our purposes, a token is essentially just a word and long phrases or sentences are "tokenized" so that the data can then be vectorized. While the exact syntax differs between each of our scripts, consider the token pattern below which is the default behavior of our SVM package using sklearn:

```
r'(?u)\b\w\w+\b'
```

This regex pattern matches any two or more word characters (primarily letters but can also include digits, underscores, etc.) meaning that it doesn't handle punctuation. It also doesn't consider words that are only one letter, however, this should work fine for our purposes. Much like capitalization, punctuation is extremely important for our dataset, especially for the 5 label iteration of our data. Specifically, it's extremely important that we capture the nuance of the exclamation point (!) as this is a key determinant of whether a student simply "recommends" or "strongly recommends" a class. The same holds true for "don't recommend" and "strongly don't recommend". For example, according to our labeling guidelines, a simple "yes" receives a label of R while a "yes!" receives a label of SR. This distinction is important and extremely abundant in the dataset. While the syntax varies by model we showcase the SVM updated token pattern below for illustrative purposes:

```
r'(?u)\b\w\w+\b|!'
```

This modifies the tokenizer to capture exclamation points and is a key driver of improved performance in the 5 label dataset.

6.2.6 Results

Now we can summarize our results and identify potential areas of improvement. Although the dataset is extremely unbalanced, we first illustrate our results using a simple accuracy measure and also list the time each model took to completely train and run. This is exactly how we evaluated our pretrained models, except now we will evaluate them using the two newer datasets henceforth referred to as the "3-Label" dataset (R, NEU, DR) and the "5-Label" dataset (SR, R, NEU, DR, SDR). We showcase the results in Tables 4 and 5.

Table 4: Machine Learning Model Accuracy and Efficiency (3-Label)

Model	Accuracy	Running Time
Logistic Regression	85.3%	1 second
Random Forest	78.1%	4 minutes
SVM (Linear)	87.1%	2 seconds
Neural Network	81.7%	1 minute

Table 5: Machine Learning Model Accuracy and Efficiency (5-Label)

Model	Accuracy	Running Time
Logistic Regression	75.1%	1 second
Random Forest	64.6%	6 minutes
SVM (Linear)	78.8%	3 seconds
Neural Network	77%	1.5 minutes

We quickly notice that support vector machine (SVM) boasts the highest accuracy closely followed by logistic regression. Importantly, these two methods are also significantly faster than either random forest or neural networks making them extremely portable and efficient in addition to their strong performance.

We now move to more advanced metrics of accuracy which better capture the strength of classification models especially ones that have severe imbalances in the dataset. Given our multi-classification problem and highly uneven dataset, these metrics provide a more nuanced view [13]:

Precision: Precision is calculated for each individual class as the ratio of true positive predictions for that class to the total number of predictions made for that class. It quantifies the accuracy of the positive predictions made by a classification model.

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

- TP_i is the number of true positive predictions for class i
- FP_i is the number of false positive predictions for class i

Recall: Recall is defined for each individual class as the ratio of true positive predictions to the total number of actual positive predictions. It measures the ability of a model to identify all relevant instances within a dataset.

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

- TP_i is the number of true positive predictions for class i
- FN_i is the number of false negative predictions for class i

F1 Score: F1 is a metric balanced between precision and recall. It is actually the harmonic mean of precision and recall and is crucial for evaluating models with imbalanced datasets.

$$F1\ Score_i = 2 \times \frac{Precision_i \times Recall_i}{Precision_i + Recall_i}$$

For all three of these advanced metrics, we may take a macro average which weights all of the classes equally or take a weighted average. We utilize the latter in order to better compensate for imbalances in the dataset.

Now that we understand the relevant metrics we can better quantify our model performance. See the advanced classification metrics for the 3-Label and 5-Label datasets in Tables 6 and 7 respectively.

Table 6: Machine Learning Model Advanced Metrics (3-Label)

Model	Precision	Recall	F1 Score
Logistic Regression	83.4%	85.3%	82.6%
Random Forest	81.1%	78.1%	70.9%
SVM (Linear)	85.9%	87.1%	85.9%
Neural Network	84.9%	81.7%	82.8%

Table 7: Machine Learning Model Advanced Metrics (5-Label)

Model	Precision	Recall	F1 Score
Logistic Regression	74.4%	75.1%	73.8%
Random Forest	71.3%	64.6%	59.1%
SVM (Linear)	79.5%	78.8%	77.6%
Neural Network	77.1%	77.0%	76.9%

Once again, we notice that SVM performed the best even when using these advanced metrics. We also observe that the percentages for these metrics are a bit lower as compared to our simple accuracy measure. This is expected given that our models are being penalized for having such high proportions of R and SR reviews. Nonetheless, these results, especially SVM, are quite promising. The pretrained models assessed at the beginning of this section are not nearly as robust.

6.2.7 Extensions

As an additional exercise, we also investigated whether these results would differ based on the type of reviews the models were trained on. Recall that in Section 4, we categorized the reviews into the following academic disciplines: STEM, Humanities, and Social Sciences. We hypothesized that given the varying student demographics in each of these fields, the composition of reviews would differ and therefore lead to inconsistent results.

For example, we believe that STEM students are less likely to write long, verbose reviews as compared to Humanities students (which includes all writing classes). Moreover, we also maintain that Humanities students use more specific language which might make the models better able to identify important words that signal a specific sentiment. As SVM was our most effective model, we decided to rerun it on both the 3-Label and 5-Label datasets using these three academic areas as subsets to train and test separately. We report our results in Tables 8 and 9.

Table 8: SVM Model by Academic Discipline (3-Label)

Discipline	Accuracy	Precision	Recall	F1 Score
STEM	84.7%	83.3%	84.7%	82.7%
Humanities	92.0%	92.7%	92.0%	89.1%
Social Sciences	89.7%	90.1%	89.7%	87.9%

Table 9: SVM Model by Academic Discipline (5-Label)

Discipline	Accuracy	Precision	Recall	F1 Score
STEM	75.8%	75.8%	75.8%	74.3%
Humanities	71.2%	75.2%	71.8%	67.1%
Social Sciences	72.2%	73.5%	72.2%	70.6%

The results are fascinating. It appears that for the 3-Label dataset, the STEM model certainly performed worse than both the Social Sciences and Humanities models. This is consistent with our predictions and having manually inspected the data, we maintain that the STEM classes undoubtedly had shorter, less meaningful reviews. Looking at the 5-Label dataset, however, we notice that the scores are much more consistent with STEM actually performing marginally better than the others.

This actually makes sense given the imbalanced size of the subsets. As we know, the more observations the model has to train on, the better it will perform. This becomes increasingly important as the number of classes increases such as from 3 labels to 5 labels. In this case, STEM comprises 2,455 of the reviews, whereas Social Sciences and Humanities make up only 1,256 and 811 observations respectively. This difference in dataset size is very likely the key contributor to the STEM model's relatively better performance.

Another possibility is that certain academic disciplines just have better reviews in general. For example, Humanities classes might have almost all positive reviews as compared to STEM whose distribution is more even. This would cause the 3-Label model to have better accuracy but the 5-Label would still suffer from the insufficient training set. Regardless, these results tell an interesting story, worthy of further exploration and analysis.

7 Theory

Given the superior performance of support vector machine on our sentiment analysis task, in this section, we will provide some discussion of SVM theory [2]. We will also test and assess the use of other kernel functions as Section 6 only focused on the linear kernel. Through this theoretical lens, we may find insights as to why SVM is such a powerful machine learning algorithm for sentiment analysis in particular.

SVM is an extension of the simpler classifier: maximal margin classifier. In a p-dimensional space, a hyperplane constitutes a flat, affine subspace of $p-1$ dimensions. The most intuitive, visual example is a two-dimensional plane in a three-dimensional space. In a p-dimensional setting, the hyperplane is defined as follows

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + B_p X_p = 0$$

where X is a vector of length p. If this equation were greater than zero, than X lies on one side of the hyperplane whereas less than zero designates the other side. In essence, the hyperplane divides the p-dimensional space into two halves.

Now suppose that we have n observations in this p-dimensional space which can be represented in an $n \times p$ matrix X. Assume that each of these observations belongs to one of two classes: $y_i \in \{-1, 1\}$. The maximal margin classifier relies on the identification of a separating hyperplane, a hyperplane that perfectly separates our observations into these two classes. This can be represented as

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + ... + B_p x_{ip} > 0$$

indicating that an observation belongs to one class, whereas

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + ... + B_p x_{ip} < 0$$

means that observation belongs to the other class. Naturally, if a hyperplane does exist that can satisfy these constraints, then an infinite number of hyperplanes exist. Consider a line separating two groups of points on a plane. We can rotate that line around in infinitesimally small amounts and still divide those two groups. This is where the "maximal" portion of the maximal margin classifier plays a role.

The idea is to create the optimal separating hyperplane by maximizing the distance of the plane from the training observations. This is done by calculating the perpendicular distance of each point from the plane and then maximizing the minimum of these distances. In other words, we want the closest of these points to be as far as possible when compared to any other hyperplane configuration. The observations with minimum distances to the hyperplane constitute our support vectors as their position dictates how the hyperplane would have to change in response to their movement. The maximal margin hyperplane can be solved for by the following optimization problem:

$$\max_{\beta_0,...,B_p,M} M, \text{ subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + B_1 x_{i1} + ... + B_p x_{ip}) \geq M, \forall i = 1, 2, ..., n$$

Notably, such a separating hyperplane does not always exist. If one did, the classification task would become extremely trivial. Thus, we must handle the non-separable case. This is where the simpler maximal margin classifier evolves into support vector classifiers.

When it is mathematically infeasible to separate the observations into exactly two classes, we can instead solve for a hyperplane that almost separates the two classes using a soft margin. This is actually preferable even if a separating hyperplane does exist as it provides greater resistance to outliers. For example, there could be one single observation that causes the hyperplane to shift drastically whereas if it were ignored, the remaining observations would be better classified. This is analogous to a single extreme outlier completely altering the line of best fit on a scatter plot. This modification improves robustness in the overall model. Misclassifying a couple of observations in exchange for better classification of the remaining training set is a worthwhile tradeoff when generalized to testing data. The optimization problem is quite similar:

$$\max_{\beta_0,...,B_p,\epsilon_1,...,\epsilon_n,M} M, \text{ subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + B_1 x_{i1} + ... + B_p x_{ip}) \geq M(1 - \epsilon_i)$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$$

The only modifications are the inclusion of slack variables $\epsilon_1, ..., \epsilon_n$ and a nonnegative tuning parameter C. Coupled together, these allow the model to violate pure separation which helps prevent overfitting. Specifically, the slack variables permit observations to be on the wrong side of the margin or hyperplane while C determines the quantity and severity of violations the model can tolerate. Observations that lie on or on the wrong side of the margin for their class are referred to as support vectors.

Technically speaking, the support vector classifier we just discussed is the same as a support vector machine using a linear kernel. In fine-tuning our support vector machine for optimal performance, we tested several other kernels. The results are presented in Tables 10 and 11.

Table 10: SVM Model Performance by Kernel (3-Label)

Kernel	Accuracy	Precision	Recall	F1 Score
Linear	87.1%	85.9%	87.1%	85.9%
RBF	84.5%	83.6%	84.5%	81.9%
Sigmoid	88.4%	87.5%	88.4%	87.0%
Poly2	85.4%	84.9%	85.4%	83.2%
Poly3	83.4%	83.2%	83.4%	79.7%
Poly4	79.0%	79.1%	79.0%	72.2%
Poly5	78.1%	80.1%	78.1%	70.0%

Table 11: SVM Model Performance by Kernel (5-Label)

Kernel	Accuracy	Precision	Recall	F1 Score
Linear	78.8%	79.5%	78.8%	77.6%
RBF	78.2%	78.8%	78.2%	76.4%
Sigmoid	74.6%	75.1%	74.6%	73.9%
Poly2	80.3%	80.9%	80.3%	79.4%
Poly3	70.3%	73.1%	70.3%	66.6%
Poly4	64.3%	70.0%	64.3%	57.7%
Poly5	63.1%	71.8%	63.1%	55.9%

We notice that the linear kernel performs well on both datasets. For the 3-Label data, sigmoid actually achieves the highest accuracy and F1 score. The same holds true for the second degree polynomial in the 5-Label dataset which indicates that the feature space can be better separated using a nonlinear boundary. Regardless, the difference in performance is minimal at best. Another interesting observation is that all of our evaluation metrics decline as the polynomial degree increases. This means that separation of the dataset is not captured well by these more complex patterns and the model suffers from overfitting. Given the strong performance of the linear kernel coupled with its simplicity, we will continue to utilize this version of SVM for the remainder of this paper.

Importantly, we have not yet discussed how SVM works for multiple classes as the theory discussed thus far only models two groups. Generalizing SVM to more than two classes is actually quite simple and can be executed through two different approaches. The first is one-versus-one classification in which an SVM is created for every pair of classes $\binom{k}{2}$. Afterwards, each observation is classified by every SVM and the class which was output with the highest frequency is then chosen. Alternatively, there is also one-versus-all classification in which an SVM is fit for every class against all other classes simultaneously. An observation is then assigned to the class for which an SVM model classified it with the highest degree of confidence. The multi-class approach used in our programs is one-versus-one.

Overall, the theory behind SVM is intriguing and it provides some clarity as to why this machine learning algorithm is ideal for sentiment analysis. SVM is able to handle extremely high dimensional spaces with tact mathematical maturity. While number of dimensions varies by data preprocessing approach, feature vectors can easily reach thousands of dimensions when vectorizing large sets of text with wide ranging vocabularies. Thus, SVM is able to handle these extremely large cases in an effective, yet rapid manner. They are also very robust due to the support vectors which allows them to prevent overfitting on such dense feature spaces as compared to other classifiers.

8 Keyword Extraction

8.1 Models

We attempted to perform keyword extraction on the corpus of student reviews that list what key skills they learned in a class. We used several different models, however, the performance of each was lackluster. This is mainly due to drastic variations in the dataset between classes. For the recommendation question, the format of student replies was relatively consistent such that the models could learn off of the 4,000+ observations. However, the difference in responses between the key skills learned in a class like "Discrete Math" vs. "The Rise of China" for instance is too much for the models to effectively determine what terms qualify as keywords.

For this reason, we attempted to batch the replies into smaller groups where the keywords would be similar. There is no official, numerical metric we can provide to quantify the models' success. However, simply scrolling through examples of course titles and their corresponding key skills indicate that the models had little to none. For that reason, we do not recommend using any of these models for the task at hand. Nonetheless, we will briefly summarize each model and its implementation.

8.1.1 KeyBERT

KeyBERT is a model used for keyword extraction which is derived from the same BERT embeddings that were used in the sentiment analysis portion from Section 6 [6]. It constructs a neural network and attempts to identify potential keywords by analyzing word occurrences and frequencies in specific reviews relative to the corpus of documents. There are modifiable parameters such as *ngram_range* and *stop_words* which allow the user to influence how many words comprise a key phrase and what kinds of stop words should be ignored or removed. After experimenting with different parameters we were unable to obtain satisfactory results.

8.1.2 RAKE and YAKE

RAKE (Rapid Automatic Keyword Extraction) and YAKE (Yet Another Keyword Extractor) are both very similar unsupervised learning algorithms [12] [9]. Despite subtle differences, they function in largely the same way. Both models tokenize the data by removing stop words and punctuation. They then analyze the frequency and co-occurence statistics of words within each review to filter out potential key phrases. Unlike RAKE, YAKE also takes into account word position within a sentence or paragraph and therefore yielded slightly better results. These models also offer modifiable parameters such as *ngram_range* which we attempted to fine tune. YAKE was slightly more successful than the others but still not suitable for this task.

8.1.3 TextRank

TextRank was adapted from another algorithm originally designed for ranking web pages [10]. It builds a graph with words as vertices and edges to represent co-occuring relationships between words in the larger text. This data structure is then used to derive the importance of a word and it is iteratively updated to factor in the importance of words that it connects to. TextRank also offered modifiable parameters which we aimed to perfect. Once again, the algorithm was highly ineffective. Words and phrases drawn from the set of skills reviews were often meaningless, grammatically illogical, or included irrelevant stop words such as "is", "a", "at", etc.

8.2 ChatGPT API

Recall that the motivation of this project was to identify suitable strategies for keyword extraction that would prove effective in a real implementation. Given the limitations of the dataset and the failure of the keyword extraction models tested thus far, we decided to implement and fine tune a ChatGPT API to address this challenge. We purchased a

ChatGPT API and tested a variety of models, prompts, and parameters to achieve the best results in a cost effective manner.

In fact, to run the API hundreds of times while testing, tweaking, and implementing our proof of concept dashboard (Section 9), we only incurred a cost of roughly $0.50. This was extremely good news. Consider the fact that CourseTable only lists around 4,000 courses each semester. If we wanted to use the API to synthesize important information, we would only need to run it one time per class per survey question. In this case, we would run it on the skills survey prompt as well as the strengths/weaknesses/improvements prompt. This means that the API would only have to run about 8,000 times to generate all of the requisite information needed for a semester's listing. This would only cost tens of dollars to successfully implement.

Choosing the best model and set of parameters as well as crafting a satisfactory prompt proved quite the challenge. We opted to use the *gpt-3.5-turbo-instruct* which yielded great results yet was much more cost effective than any 4.0 model. One downside of the model is that it imposes a token limit inclusive of the prompt and the corresponding reply. The reply is not a concern, as we are requesting the model to return a simple list of words or phrases no more than a few dozen tokens. However, for extremely large classes with a sizeable collection of survey replies, the token limit can be easily exceeded.

To remedy this, we decided to batch a random sample of 20 reviews or just take all of the reviews if there were less than 20. By randomly sampling, we fairly represent the students' perspectives and are able to batch all of the reviews into one singular API call. This is extremely efficient in terms of cost and speed. See an example API creation in Figure 3.

Figure 3: ChatGPT API Call

```

    response = client.completions.create(
        prompt=prompt,
        model="gpt-3.5-turbo-instruct",
        max_tokens=100,
        temperature=0.2
    )

```

In the above, we set the maximum number of tokens to 100 forcing ChatGPT to limit the size of its response and achieve a concise list of keywords. We also set the temperature parameter to 0.2 which determines how random or deterministic we want the API to behave when synthesizing the results. While we achieve some randomness when grouping the reviews to pass into the model, we opt for a more deterministic response. For the prompt argument, we crafted a meticulously worded set of instructions and then passed the cleaned set of reviews to ChatGPT as a numbered list. Below, we list the specific prompt used for the skills reviews and the strengths/weaknesses/improvements reviews respectively.

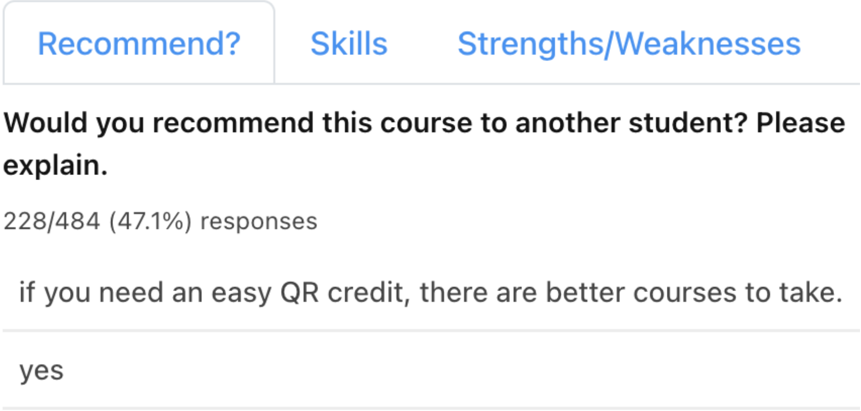
> The following are a set of student responses to this end of semester survey question: “What knowledge, skills, and insights did you develop by taking this course?” I would like for you to read through the responses and summarize the key knowledge, skills, and insights in a list. These can be words or SHORT phrases on broader topics and please order them by importance. Only list 4 to 7 items! Here are the responses: \n{skills_string}
>
> The following are a set of student responses to this end of semester survey question: “What are the strengths and weaknesses of this course and how could it be improved?” I would like for you to read through the responses and summarize the key strengths, weaknesses, and improvements in 3 separate lists. These can be words or SHORT phrases. Only list 3 to 5 items for each list! Here are the responses: \n{swi_string}

We used the above prompts to extract keywords for those two survey questions. Notably, {skills_string} and {swi_string} get replaced with a set of twenty (or less) reviews that have been cleaned and organized into an easily legible numbered list. After fine tuning the prompts, parameters, and overall request structure, we found the ChatGPT API to be far superior to the previous models. Moreover, the performance was so impressive we would even recommend it for a practical implementation of this concept. In Section 9, we demonstrate such an application.

9 Dashboard

Ultimately, the goal of this project was to test different sentiment analysis and keyword extraction models in the hopes of identifying suitable methods for a practical implementation. In theory, this would exist as an additional feature to CourseTable or an equivalent website used by faculty and students alike to extract key course information quickly and accurately. As currently designed, CourseTable just lists every student reply to the three registrar survey questions as shown in Figure 4.

Figure 4: CourseTable Reviews List



Would you recommend this course to another student? Please explain.

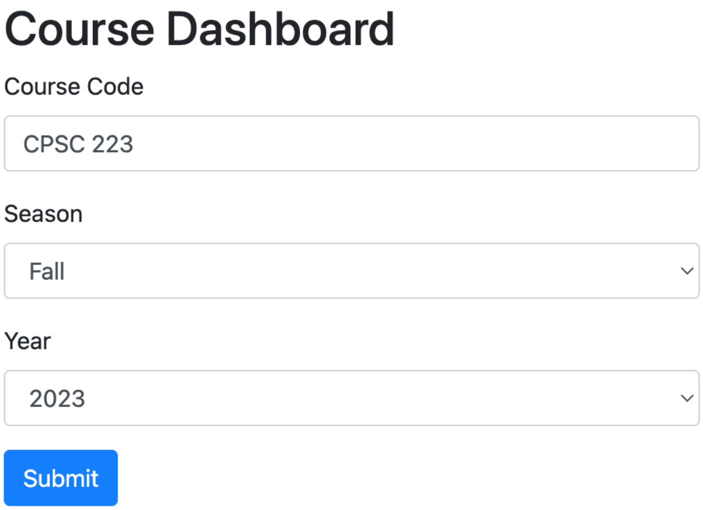
228/484 (47.1%) responses

if you need an easy QR credit, there are better courses to take.

yes

While this is a great feature of CourseTable, there would clearly be value to having NLP tools that could instantaneously list the key skills, strengths, weakness, and areas of improvement from a course's reviews as well as quantify the percentage of students who recommend that class. This would be useful for professors to quickly glean information about their course but more so for students who could grasp the essence of a class in mere seconds while searching through hundreds of potential choices. To demonstrate this functionality, we went ahead and created a simple dashboard, powered by the best methods we have uncovered in this project thus far. Specifically, it uses support vector machine to classify the recommendation reviews into our five categories (SR, R, NEU, DR, SDR) and uses the ChatGPT API as discussed in the previous section to extract keywords regarding skills, strengths, weaknesses, and improvements. When the user accesses the page, they are presented with a simple interface to enter a course code, season (fall, summer, or spring), and year (2012 to 2023) as shown in Figure 5.

Figure 5: NLP Dashboard Input Form



Course Dashboard

Course Code

CPSC 223

Season

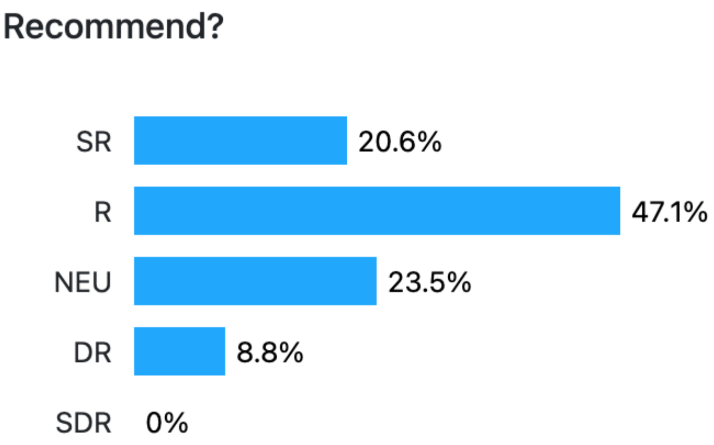
Fall

Year

2023

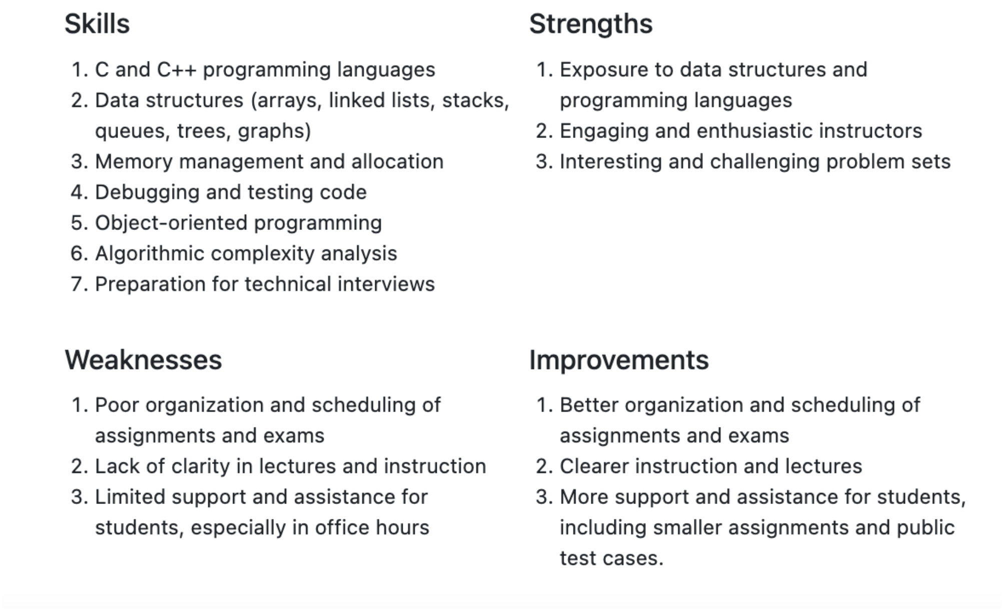
This includes some error handling such as ensuring a valid course is provided and drop down menus to enforce selection of a valid season and year. After the user submits, the backend will take their entry, scrape the necessary data from the CourseTable database as performed in Section 4, run the SVM sentiment analysis model, make the ChatGPT API requests, and finally output the results to the user. This process takes a mere five seconds. In Figure 6, we provide an example of what the recommendation output looks like.

Figure 6: NLP Dashboard Recommendation Output



Similarly, Figure 7 illustrates the keyword extraction output for skills, strengths, weaknesses, and areas of improvement.

Figure 7: NLP Dashboard Keyword Output



Skills

1. C and C++ programming languages
2. Data structures (arrays, linked lists, stacks, queues, trees, graphs)
3. Memory management and allocation
4. Debugging and testing code
5. Object-oriented programming
6. Algorithmic complexity analysis
7. Preparation for technical interviews

Strengths

1. Exposure to data structures and programming languages
2. Engaging and enthusiastic instructors
3. Interesting and challenging problem sets

Weaknesses

1. Poor organization and scheduling of assignments and exams
2. Lack of clarity in lectures and instruction
3. Limited support and assistance for students, especially in office hours

Improvements

1. Better organization and scheduling of assignments and exams
2. Clearer instruction and lectures
3. More support and assistance for students, including smaller assignments and public test cases.

Overall, the dashboard signals promising potential for these NLP techniques. Students and faculty would surely benefit from easy access to this consolidated information.

10 Conclusion

In this project, we tested several sentiment analysis and keyword extraction models on Yale course reviews. The hope was to identify a set of suitable NLP tools for a practical application which would serve the faculty and student body. We described the data collection and cleaning process in detail such that our datasets can be replicated.

For sentiment analysis, we tested three pretrained models: VADER, TextBlob, and RoBERTa. These consist of both rule-based and deep learning models, however, they yielded disappointing results. We then tested some models that were built from scratch using our own labeled training datasets. These included the machine learning algorithms logistic regression, random forest, neural networks, and support vector machine. The results varied, but SVM was the most robust model with an accuracy of 87% and 78% for the 3-Label and 5-Label datasets respectively.

Given the success and portability of SVM, we then discussed some mathematical theory and explained why this machine learning technique was particularly effective. We also tested several other kernels including RBF, sigmoid, and multiple polynomials. A few of these performed well, however, none performed significantly better than linear. We opted to keep the linear kernel due to its balance of efficacy and simplicity.

Next, we tackled keyword extraction. We utilized a number of preexisting models including KeyBERT, RAKE, YAKE, and TextRank. These employed a variety of methods including deep learning and graph based frameworks. All four of these models were extremely ineffective. As an alternative, we created and fine tuned a ChatGPT API to handle the keyword extraction tasks. The API worked extremely well and was certainly viable for a real NLP application.

Finally, we combined all of the best procedures we uncovered throughout the project. This was comprised of an SVM model using a linear kernel for sentiment analysis and the ChatGPT API for keyword extraction. We combined these programs into a backend that powered a proof of concept dashboard meant to realize the vision behind this project. While experimental, the dashboard provides sophisticated information about Yale courses in just a few seconds. Overall, the potential of these applications is quite promising and would serve the Yale community well.

References

[1] C. J. Hutto and E. Gilbert, "VADER: A parsimonious rule-based model for sentiment analysis of social media text," in Proc. ICWSM 2014, 2014, pp. 216-225.

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani, "Support Vector Machines," in An Introduction to Statistical Learning, New York: Springer, 2013, pp. 337-372.

[3] J. Doe, "Sentiment analysis using logistic regression and naive Bayes," Towards Data Science, Oct. 1, 2020. [Online]. Available: https://towardsdatascience.com/sentiment-analysis-using-logistic-regression-and-naive-bayes-16b806eb4c4b.

[4] J. Smith, "An easy tutorial about sentiment analysis with deep learning and Keras," Towards Data Science, Feb. 15, 2021. [Online]. Available: https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91.

[5] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[6] M. Grootendorst, "KeyBERT: Minimal Keyword Extraction with BERT," GitHub, 2020. [Online]. Available: https://github.com/MaartenGr/KeyBERT.

[7] Ngoc, Thanh & Thi, Mai & Thi, Hang. (2021). "Sentiment Analysis of Students' Reviews on Online Courses: A Transfer Learning Method," 10.46254/AP01.20210122.

[8] Pu, Xiaomin, Guangxi Yan, Chengqing Yu, Xiwei Mi, and Chengming Yu. (2021). "Sentiment Analysis of Online Course Evaluation Based on a New Ensemble Deep Learning Mode: Evidence from Chinese" Applied Sciences 11, no. 23: 11313. https://doi.org/10.3390/app112311313.

[9] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, "YAKE! Keyword extraction from single documents using multiple local features," Information Sciences, vol. 509, pp. 257-289, 2020.

[10] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in Proc. of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004), 2004, pp. 404-411.

[11] S. Loria, "TextBlob: Simplified Text Processing." [Online]. Available: https://textblob.readthedocs.io/en/dev/.

[12] S. Rose, D. Engel, N. Cramer, and W. Cowley, “Automatic keyword extraction from individual documents,” in Text Mining: Applications and Theory, M. W. Berry and J. Kogan, Eds. Wiley, 2010, pp. 1-20.

[13] T. Kanstren, “A look at precision, recall, and F1 score,” Towards Data Science, Mar. 3, 2022. [Online]. Available: https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec.

[14] Y. Liu et al., “RoBERTa: A robustly optimized BERT pretraining approach,” arXiv preprint arXiv:1907.11692, 2019.

[15] Z. Jiang, C. Miao and X. Li, “Application of keyword extraction on MOOC resources,” in International Journal of Crowd Science, vol. 1, no. 1, pp. 48-70, March 2017, doi: 10.1108/IJCS-12-2016-0003.

Acknowledgments

There were many more sources used in the creation of this project beyond those that were directly cited in this report. All of these are included as hyperlinks in our code archive as well as our GitHub repository.

We give a special thank you to the CourseTable staff for allowing us to use their dataset and helping us troubleshoot the data collection process!