**Assessment 3: WebCrawler and NLP System**

By: Jagraj Gill

Date Submitted: 28 Aug 22

**Overview**

**Aim and Motivation**

The aim of this study is to deconstruct movie scripts that are associated with Oscar nominations via a subfield of sentiment analysis known as emotion detection (ED), whose primary goal is to extract the perceived emotions of textual data. The motivation behind this study is to better understand the commonalities and differences between the perceived emotions in these scripts via statistical analysis. Machine learning (ML) will be implemented to determine what, if any correlations between features suggest what comprises an award-winning narrative, that would generate enough predictive accuracy to be a viable tool on unseen data. Current applications include hate speech detection, emotion retrieval from suicide notes, multimedia tagging, and analyzing e-commerce review sentiments (Yang et al., 2012; Wang et al., 2019; Allouch et al., 2018). Additionally, as noted in *Figure 1* below, the ratio of text-based emotion detection is a limited study as compared to emotion detection in general, as seen in both IEEE Explore, and Scopus sample databases (Acheampong et al., 2020).

Chart, bar chart

Description automatically generated Chart, bar chart

Description automatically generated

Figure 1: The IEEE Explore Database is a digital repository that provides technical, journalistic, and conference-related materials related to computer science, electrical engineering, and electronics, containing over 5.7 million individual items (“Advancing Technology”, 2022). The Scopus Database is a provider of citation indices and abstracts pertaining to academic books, journals, and conference proceedings worldwide, containing over 39,000 titles (“Research Intelligence”, 2020). The increase in the frequency of research output in ED in multimodal forms such as speech, body language, and facial expressions as compared to text-based ED from 2010-2020 is evident in both academic databases.

Furthermore, the disparity in text-based ED is outlined mathematically as

 (1)

where *T* is the set of text to which the emotions are to be derived, *A* is the set of authors of *T*, *E* is the set of all emotions and *r* is a function that describes the emotions *E* of author *A* from text *T* (Kao et al, 2009). The ED from text problem is derived from linking the input text to the actual projected emotion that leads the author to pursue specific artistic avenues in writing. Subsets of both *T* and *E* become non-trivial as they are either loosely defined, ambiguous, subject to language evolution, and finally, there are no set relations or classifications that exist to encompass all human emotions.

**Emotional Model Theories**

When undertaking text-based ED, it is vital to define the model of emotion for use, as described in *Text-based emotion detection: Advances, challenges, and opportunities* by Anchempong et al. (2020). There are two primary models used in ED: discrete emotion, and dimensional emotion models. Discrete emotion models involve bucketing emotions into discrete categories, while dimensional emotion models assume that emotions are not independent and are related to one another, hence should be mapped in a spacial format. There are unidimensional and multidimensional models that are utilized to depict the degree of relation between emotions, and they typically reflect on two behavioral states, good, and bad. Both models are affected by relative degrees of their occurrences; however multidimensional models are more prominent and frequently used.

The two commonly used discrete emotion models include the Paul Ekman and Robert Plutchik models. The Paul Ekman model distinguishes emotions based off six basic categories. This theory hypothesises that there are six fundamental emotions that originate from separate neural networks in the brain, and result from how one perceives an event, hence emotions are manifested independently of one another. The six emotions are: happiness, sadness, anger, disgust, surprise and fear. Furthermore, this theory states that more complex emotions may be derived from the synergy of the fundamental emotions.

The Robert Plutchik model suggests that there are few primary emotions which take place as opposing pairs, and produce complex emotions by combination, as shown in *Figure 2* below. He adds two emotions to the Ekman model, acceptation, and anticipation. Plutchik postulates that for each emotion, there exists various degrees of intensities that occur because of how events are formulated by the experiencer.

Additionally, there is the Orton, Clore, and Collins model, which opposed both Plutchik and Ekman’s basic emotion theory, proposing that emotions arise from the way they were perceived by the individual, and by the degree of their intensity. All three may be implemented in ED system design, but the OCC model allows for a broad spectrum of emotion representation, with 22 in total.

The primitive dimensional emotional model is presented by Russell as a two-dimensional representation known as the circumplex of affect, shown in *Figure 2* below. The model separates emotions in the arousal valence domains, with the arousal splitting the emotions by activations and deactivations, where the valence differentiates emotions by pleasantness and unpleasantness. In this model, emotions are not independent, but related. Discrete models have been accepted for their emotional classification capabilities because they are simple but are not exhaustive of the full range of emotional classes available. They can map the intensities or degrees of occurrences as compared to dimensional models. Dimensional models are suggested for projects that involve comparing similarities in emotions.

Diagram

Description automatically generated Diagram

Description automatically generated

Figure 2: Plutchik’s wheel of emotions and Russell’s circumplex model of affects.

**Text-based ED Techniques**

There are three primary techniques that are explored for text-based emotion detection, rule construction, machine learning, and a hybrid approach. The rules-based approach uses grammatical and logical rules to adhere to for ED. This method uses both keyword recognition and lexical affinity. Keyword recognition is reliant on emotion dictionaries or lexicons, some of them being Wordnet-Affect, EmoSenticNet, and the National Research Council of Canada lexicon. The emotion lexicons have emotion key words where the occurrences of them in text occurs at the sentence level. Once the keyword is determined, the sentence is labeled. A challenge that may be faced using this method will be the constraint imposed by the number of available emotions in the dictionary, navigation of word sense ambiguity, and processing of linguistic information when sentences are short or sparse of key words. To augment these limitations, the lexical affinity method may be explored, as random emotional words are assigned a probabilistic affinity on top of identifying keywords.

An example of challenges in the rule-based approach was set out in *Emotion and Sentiment Analysis from a Film Script: A Case Study,* by Yu et al. (2017), whose aim was to examine the emotions in the thriller movie script *A Hard Day*, as laid out by Plutchik’s wheel of emotions. Manual sentiment (positive, negative, and neutral) and intensity tagging were conducted by undergraduate students and was compared with the results of the Python-based NLTK (Natural Language ToolKit) package VADER (Valence Aware Dictionary and sEntiment Reasoner) 2.5. The result was a 42.6% matching rate when comparing both ED and intensity measures after averaging the negative and positive emotions. The barrier faced in this study was analogous to finding a universal relation function *r* for subsets of text and authors as outlined above in Kao et al. (2009). Future aspirations of the researchers included adding more human taggers and adopting ML algorithms.

ML-based models have been widely implemented for text-based classifications, where supervised machine learning models have performed better in emotion detection rates as compared to traditional unsupervised machine learning (TUML). The more widely explored unsupervised methods have been traditional in nature (i.e., Support Vector Machines, Naïve Bayes), which have proven to be less robust, and do not explicitly extract semantic information from texts which is needed for effective emotion detection. The current science supports deep learning (DL) approaches as layers can extract, or learn the intrinsic properties that text carries, hence some models suggested by the initial paper outline DL outperforming TUML.

The final avenue that is typically considered is a hybrid approach, where rule construction and ML will deliver a unified model which takes in the strengths of each and mitigate their limitations. Acheampong et al. (2020) also indicates in its survey that this results in “*satisfactory results*”, referring to TUML and rule construction methods, and points to a void in exploring effective approaches with deep learning models, as this is the latest advancement and optimal modalities have not been established.

**Domains**

Four websites were considered candidates for text-based web scraping: Simply Scripts (<https://www.simplyscripts.com>), Script-O-Rama (<http://www.script-o-rama.com>), The Internet Movie Script DataBase (<https://imsdb.com>), and Scripts Plug (<https://www.scriptslug.com>). It was confirmed with the *response.status\_code()* method found in the *requests* library the HTTP response status code for all candidate sites was 200, indicating a successful request and response to the requested data. One reason for utilizing these databases was significant script volume pertaining to Oscar-nominated scripts and similarly for unremarkable titles. All websites were HTML (HyperText Markup Language) and XML (Extensible Markup Language)-based hence there were multiple options available for parsing their data: Python’s *request* module, BeautifulSoup, Scrapy, and Selenium. Determining the avenue that will be more efficient and beneficial to use came down to the structure of the website of interest and the ability to effectively integrate the tool with Python and Jupyter Notebook (i.e., Selenium handles core JAVA applications well), whereas in terms of speed and efficiency, Scrapy would be more beneficial as it supports multithreading and pre-built request and parsing modules. BeautifulSoup is regarded for its comprehensive documentation and ease-of-interpretation, while the standalone *requests* module is known for performing common HTTP requests (i.e., GET, POST, PUT, PATCH, and DELETE) in a low-code manner.

Simply Scripts is a site containing downloadable movie scripts, screenplays, produced and unproduced movies, television shows, anime, plays and radio shows. Currently it contains a total of 1176 movie scripts with 95 Oscar contenders. The scripts are displayed in HTML, *.txt*, and *.pdf* format, which are not hosted on the Simply Scripts server, hence the website acts as an aggregator instead of the originator of the content. This will complicate the scraping process as each script page may potentially have a different site structure (i.e., varying tags used in markup, URL trees have varying levels and names, different *robots.txt* parameters), and viewing modes (i.e., online viewing, file download). Simply Scripts displays an expired copyright (1999-2020) stating “*No graphics, pages, html, banners, etc. found on this site may be reproduced, posted, or sold without the consent of Simply Scripts*” (“SimplyScripts Disclaimer”)*.* One would have to refer to the hosted site for their disclaimer to determine legality of use, which is tedious for large document retrieval tasks. Finally, there were no scraping security obstacles or limitations to overcome (i.e., CAPTCHA, crawl rate or request rate declarations in *robots.txt*).

Script-O-Rama contains draft, revised, and final script versions of 1430 movie titles. Additionally, there are TV, haiku, and anime works. The scripts are displayed in HTML*, .txt, .doc, .pdf,* and *.zip* format, which are not hosted on the Script-O-Rama server, hence the same programmatic and validation of legality challenges are faced as Simply Scripts. There were no scraping security obstacles or limitations to overcome, and there was no copyright or legal policy present.

The Internet Movie Script DataBaseis claimed to be the “*Web’s largest movie script resource*”, with over 1200 movie titles. The site also contains select TV and French transcripts. All movie scripts are displayed in HTML format and are hosted on the server, which suggested ease of scraping. A fair use policy was posted, pursuant to Section 107 of the United States Copyright Act of 1976.

Scripts Plug was the final website to be considered from the candidate list. All the movie scripts were hosted on the site server, and were organized by genre, studio, and Oscar lists. The scripts URLs also followed an intuitive format for ease of collection, were all in *.pdf* format and were contained on all HTML-based pages with congruent formatting. It was the only website that contained a comprehensive privacy and legal terms of service policy. For these reasons, it was chosen to be the primary source of data retrieval. Finally, there were no scraping security obstacles or limitations experienced.

To put into perspective the number of total movie scripts that each of the sites represent, there are an estimated 500,000 movies released worldwide, suggesting that readily available content is sparse, and/or these sites put a heavy focus on Hollywood titles (Follows, 2021). Although each of the above sites and downloaded scripts contain copyright-protected content, the fair dealing exception in the Copyright Act permits the use of copyright material for the purposes of research and is specifically stated as a user’s right according to the *Copyright Act of Canada* (Legislative Services, 2022).

**WebCrawler Workflow**

The code development was executed in *Jupyter Notebook* version 6.4.11 utilizing Python version 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)]. A private GitHub repository was set up for version control and backup storage purposes, which may be found at [*https://github.com/JagrajGill/ScreenplayEmotionDetection/*](https://github.com/JagrajGill/ScreenplayEmotionDetection/) . To ensure PEP (Python Enhancement Proposal) compliance, the *nbQA* (notebook Quality Assurance) package was applied through the command line, which acted as a wrapper for the *flake8* style guide enforcement tool specifically for *iPython* and .*ipynb* extension-style notebooks.

The tool used for data extraction in this study was *BeautifulSoup Version 4,* with the package name *beautifulsoup4* (BS4).BS4 is a Python-based library that is implemented for parsing HTML and XML-based files. It was installed with the *pip* package manager along with the *lxml* parser as it is superior in execution time to the native Python parser (ElementTree 1.3) primarily due to being executed completely at the C level, without any interaction with the Python code (Behnel, n.d.).

The initial step to text scraping was to ensure that the status code of 200 was returned from the candidate sites, and *BeautifulSoup* along with the *lxml*  parser was able to print the page content for preview, as shown in *Figure 3* below.

Text

Description automatically generated

Figure 3: The candidate list sites are stored in a list. A function, *get\_url\_status\_preview()* takes in a URL argument and initially stores the *request.get()* method output in the *response* variable. If the *response.status\_code* is not 200, then the function states that the site is unable to fulfill the request, otherwise the response content is parsed, stored in the *soup* variable, and returned. A for loop appends an empty list *url\_info* with the *soup* content and prints the result for preview. Consideration for a crawl delay was made, and it was determined to exclude this feature in the scraper for multiple reasons: there was no documentation requesting a timeout, the chosen site Scripts Plug was sparse on data loaded on each page, and there was not an extensive amount of bandwidth needed for document retrieval.

Additionally, the *user-agent* header was not requested by the four sites. The next step was to obtain a list of URLs that are on the Script Plug homepage with the *soup.find\_all()* method shown in *Figure 4* below.



Figure 4: A URL list is defined, the *soup* function is instructed to find all *a* class tags and extract the results for the tags containing the *href* attribute only. The *get()* method returns the URL found for *a* and stores it’s value in the list.

The above was accomplished for 493 mixed genre tiles and 55 Oscar-nominated films for the years 2017-2022. This resulted in a URL list of web pages that contained a *.pdf* link of the movie script. In order to rectify any class imbalances, the *random.shuffle()* method was invoked to mix the URL list , and then an equivalent number of unremarkable movies was selected from it. *Figure 5* uses a similar list comprehension method as the URL list retrieval in *Figure 4*.



Figure 5: A PDF list is defined; the *soup* function is instructed to find all *a* class tags and extract the results for the tags containing the *.pdf* extension. The *re.compile()* method allows for setting the regular expression to find the raw string of the .*pdf* extension. This list was generated for Oscar-nominated and unremarkable movies.

The final step was to take the PDF hyperlinks and download them to a local directory project folder. This was executed with a for loop as shown in *Figure 6* below.

Text

Description automatically generated

Figure 6: For all the URLs contained in the Oscar-nominated and unremarkable movies script list, if the response status code permits, then a file path is defined using the *os* library, and the file obtained from the response variable is opened to be written in binary mode to the output directory.

**Data Wrangling**

Before the PDF text could be utilized for statistical analysis and machine learning implementation, it needed to be extracted from the individual documents into a single columnar table format. This was accomplished with the *Apache Tika* parser, a content extraction library for structured text and file metadata. The *Apache Tika REST* service is able to be called natively within Python but required *Java Runtime Services* to be installed prior to use. *Figure 7* below describes the steps pursued.

Graphical user interface, text, application

Description automatically generated

Figure 7: The extension of the files of interest and path to the files are defined. The parameters of the *Tika* parser are stored in the headers dictionary. The parameters instruct *Tika* to convert any inline images in the document to text and generate a timeout of the OCR (Optical Character Recognition) after 300 seconds has passed. An empty files list is created and a for loop appends the list with the files matching the *.pdf* extension. This is done by the *os.walk()* method going through the directory tree starting at the path specified in a top-down fashion, then the *glob.glob()* method retrieves the files with the matching argument of the directory path and *.pdf* extension. An empty *DataFrame* is created, a for loop requests the *Tika* parser to execute on the files list, store the content in a *text* variable, and finally assign both the *filenames* and *text* to an index, *idx*. The *enumerate()* method provides a running count of the iterations. Finally, the *DataFrame* is stored as an *.xlsx* document as these two formats function seamlessly together in-code for manipulation and storage, respectively.

Two movies were unable to be parsed due to the OCR tool being unable to recognize the images (*The Santa Clause* and *Touch of Evil*). Preprocessing steps were as follows:

**1. Removing all Punctuation**

Punctuation in language partitions text into sections, paragraphs, sentences, and sometimes lists. The *str.replace()* method allows one to use a regular expression (REGEX) to remove all non-alphanumeric characters in a string, by replacing it with an empty space, or whitespace. Regular expressions are an ordered arrangement of characters that describe a particular search design. This is a necessary step to obtain correct word frequencies and individual words, which is described in *Figure 8* below.

Graphical user interface, text

Description automatically generated

Figure 8: A punctuation removal function takes in a string argument, and an attempt to replace the non-alphanumeric characters with whitespace is made, returning the cleaned text. The *removing\_puncutation()* function is applied across the entire *DataFrame*.

**2. Lowering Text**

Lowering the text tackles sparsity issues, where words may be read and interpreted similarly, however, written in a different manner. A vectorizer or encoder may interpret such words as different entities, hence assigning different numerical transformations to them. The *lower()* method is used in the text column as shown in *Figure 9* below.

Text

Description automatically generated

Figure 9: A lambda function is defined to *lower()* the text if the type of the text is considered to be a string, otherwise return the original text. This function is applied using the *applymap()* method, which implements the function on the *DataFrame* in an element wise fashion.

**3. Removing all Numbers**

The decision to remove numbers from a string of text is dependant on whether numbers will provide valuable information in text analysis or in the model development phases. In sentiment analysis, it is not considered to be informative, hence is removed as shown in *Figure 10* below:



Figure 10: A REGEX expression defines all numbers and the *str.replace()* method is used to replace the numbers in the text columns with whitespace.

**4. Removing Single Letters**

It was noticed in the previous step that there were letters present next to places in the string of script text where non-alphanumeric characters were removed. To remedy these single letters, two REGEX were invoked in sequence as shown in *Figure 11* below.

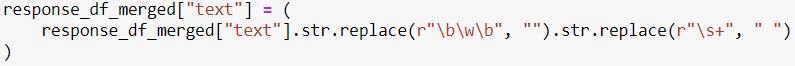


Figure 11: The first REGEX denotes finding the raw string of a word characters that are encompassed by a word boundary, and the second REGEX finds the sequences of more than one whitespace and replaces them with uniform whitespace.

**5. Keeping only English Language Words**

In order to apply English-based lexicons to the processed text, the non-English words were removed. This was accomplished due to the limited understanding of other languages and their lexicons, as shown in *Figure 12*.

Text, letter

Description automatically generated

Figure 12: A corpus of English language words are downloaded and assigned to a variable, *words*. The *apply()* method takes in the *lamba* function as an argument, which returns only the strings that are contained in *words,* and joins the strings back together in the *text* column.

**6. Tokenization**

Tokenization is a type of text parsing, which separates a piece of text into smaller units called tokens. It was applied to the *text* column as described in *Figure 15* below in order for a vectorizer to properly segment and assign the representative vectors.

A picture containing text

Description automatically generated

Figure 13: *NaN* columns of the *DataFrame* are filled with an empty string as tokenization requires strings to be present in all observations. The *word\_tokenize* function is applied from the NLTK package.

**7. Removing Stopwords**

Words that are contained in a negative dictionary are referred to as stopwords. Implementing them in an ED-based NLP model will be insignificant for the addition of predictive capabilities, hence are removed according to *Figure 14* below.

Text

Description automatically generated with medium confidence

Figure 14: A list of English stopwords are downloaded and defined. Using list comprehension, a *lambda* function returns all strings that are not contained in the stopwords dictionary. The *apply()* method executes the function across the *text\_tokenized* column.

**8. Lemmatization**

Lemmatization is a process that reduces words down to a common root form or base, while also taking into account the vocabulary and morphological analysis of the word. *Figure 15* shows the implementation of the *WordNetLemmatizer()* to the tokenized text.

Text

Description automatically generated with medium confidence

Figure 15: The *WordNetLemmatizer()* is applied via list comprehension to the words in the *text\_tokenized* column.

After text preprocessing, the Python-based emotion detection package *LeXmo* was invoked on the text column to feature engineer the emotion and sentiment polarity column scores (positive and negative), as shown in *Figure 16*. The *LexMo* package contains 14,182 unigrams, approximately 25,000 word senses, and is based off the NRC Emotional Lexicon and Robert Plutchik’s wheel of emotions model as previously discussed, containing eight primary emotions: anticipation, anger, fear, trust, disgust, surprise, sadness and joy. The annotations provided were manually conducted via crowdsourcing on Amazon’s *Mechanical Turk*. The association scores are binary (0 for no association, and 1 for associated), and there are four levels of association for the senses (not associated, weakly, moderately, and strongly associated) (Mohammad & Turney, 2011). There are no hyperparameters to consider as this is a rigid model.

Text

Description automatically generated

Figure 16: The *LeXmo* emotion and sentiment detection for the script text is applied via a *lambda* function.

**Data Summarization**

After all preprocessing steps were applied, the cleaned corpus consisted of 220,986 total tokens from 105 distinct movie scripts (54 Oscar-nominated, and 51 unremarkable titles), an average document length of 2085 tokens and 13,064 distinct tokens. A distribution of the script length is visualised in *Figure 17* below.

Chart, histogram

Description automatically generated

Figure 17: The majority of scripts fall between a cleaned corpus length of 2000-2300 tokens.

Next, the most frequent tokens used are shown in the plots in *Figure 18* below.

Chart, bar chart

Description automatically generatedChart, bar chart

Description automatically generatedChart, bar chart

Description automatically generated

Figure 18: The tokens ‘back’, ‘day’, and ‘one’ fall within the top five most used in both the Oscar-nominated and unremarkable scripts, where ‘day’ is the most prominent overall.

Finally, the relative emotions across the Oscar-nominated and unremarkable scripts are observed as shown in *Figure 19*.

Oscar-Nominated and Unremarkable Script Relative Sentiment and Emotion Scores

Chart, bar chart, histogram

Description automatically generatedChart, bar chart, histogram

Description automatically generated

Figure 19: The Oscar-nominated scripts (left) exhibit less perceived negativity and fear than their unremarkable counterparts (right).

A major consideration in calculating the frequency of tokens, detected emotions and sentiments is the choice of titles. Since choosing the entire population to sample is unfeasible, attempts to cover a breadth of genres and randomize selection were made to create as diverse a corpus as computing resources and availability permitted.

**Machine Learning Structure**

The *labelbinerizer()* method was invoked on the label column in order to transform the binary targets to a column vector. Fitting and transforming were then applied to the column, along with the *ravel()* method, to return a flattened 1-D array. After defining the feature columns as the script text, emotional scores, and sentiment polarity and target variable (*X and y respectively*), the data was split into training, testing, and validation sets using *Sklearn’s* (Scikit-Learn) *train\_test\_split()* method, with the training, test, and validation sets comprising of 60, 20, and 20 percent of the data respectively. The *random\_state* was set to preserve the distributions to each set, along with the *stratify* argument which was set to *True* to keep label distributions uniform.

Observations that were text-based required encoding to facilitate target predictions, hence the *TfidfVectorizer()* was applied via the *Columntransformer()* method to passthrough all emotional score and sentiment columns. This generated a sparse matrix of feature columns, mapping a column of category indices to corresponding vectors containing the respective *TF-IDF* (Term Frequency, Inverse Document Frequency) scores, *w*. *Equation 2* and *3* below describe the score calculation as:

A picture containing logo

Description automatically generated,A picture containing text, clock, watch, gauge

Description automatically generated (2)(3)

where *j* describes the document, *i* is the term, and *n* is the total number of documents. The greater the *TF-IDF* score, the more relevant a term is to the collection of documents. Lastly, *fit\_transform* was applied to the *X\_train* set, while transform was applied to the *X\_test* *and X\_val* sets. No feature scaling or normalization was applied as *Sklearn’s* *Tfidfvectorizer()* normalizes document lengths to 1, such that the relative frequency values are taken, introducing no additional biases. Additionally, the vectorizer applied cross-document normalization which weighed each word with the inverse of the corpus inclusion frequency. This indicates that taking the absolute against the relative frequency will produce different distances, although these will be a constant factor across the generated vectors of the corpus. The relative emotional scores are generated for the emotions and sentiments across the script documents.

Three supervised machine learning algorithms were instantiated: *RandomForestClassifier(), LogisticRegression(),* and *SVC(),* which were obtained from Scikit-Learn. This was due to their superior predictive capabilities used in conjunction with pre-defined lexicons (i.e., the hybrid approach) compared to TUML approaches, as noted previously by Acheampong et al. (2020).

The random forest (RF) algorithm utilizes an ensembling process where a random sample of the training set is individually training on, and a model of rules is generated from the samples derived from the original data, with replacement; this is known as row sampling. Ensembling here refers to a combination of models analogous to combining multiple trees to generate a forest. Row sampling with replacement is commonly referred to as bootstrapping. The output of each decision tree, or model, generates a vote for a class, and the class which gets the majority of votes determines the final class output; this is known as the aggregation step, hence the nomenclature of the RF algorithm’s technique, bootstrap aggregation (Analytics Vidhya, 2022). One distinction from a traditional decision tree (DT) model is that when growing the nodes of a tree, the DT determines the best feature for node splitting at all depths of the tree, whereas the RF algorithm determines the best feature from a random subset of features, which in turn results in greater tree diversity. A greater diversity typically generates a model that trades off a higher bias for a lower variance.

As this study focuses on a binary classification problem, a baseline model that is typically employed for such scenarios is logistic regression (LR), for its ease of interpretability and explainability. LR is commonly used to determine the probabilities of observations or instances belonging to a particular class, with the threshold probability initially set at 50%. If the model determines that the probability is above this threshold, it will output that the instance belongs to that class, otherwise the instance does not belong. The estimation is based off the probabilities from a weighted sum of input features and a bias term, resulting in the logistic represented by the vector *Equation 4* below (GeÌron, 2019) :

 (4)

where *θ* is the models parameter vector, containing the bias term *θ0 , x* is the instance feature vector, *hθ* is the hypothesis function using the model parameters of *θ*, and *σ* is defined as the logistic function below:

Chart, box and whisker chart

Description automatically generated (5)

This function is chosen as it keeps the probability between 0, and 1, since there are horizontal asymptotes that exists at *t=0* and *t=1*. When the logit model has calculated the probability of an event belonging to a particular class, the prediction, *ŷ* may now be made as:

A picture containing text, clock, watch, gauge

Description automatically generated (6)

where 0.5 is known as the probability threshold value (Ageron, n.d.).

Finally, the Support Vector Machine (SVM) algorithm was chosen as it is an algorithm that contrasts both RF and LR. (Pedregosa et al.,2011). The aim of this approach is to construct a hyperplane in *n*-dimensional space that maximizes the distance between the observations in the classes and the plane itself. When the hinge loss function is minimized, the hyperplane margin is maximized.

The baseline results (i.e., macro F1-Score) indicated that RF was the optimal choice, hence it was the candidate for further exploration and hyperparameter optimization.

The random forest estimator was optimized using the *GridSearchCV()* method. This was accomplished by passing a dictionary of parameter names and values to iterate over through a passed list. The two parameters included tree depth, and number of trees. The tree depth is the number of node splits that occur in a tree, and the number of trees refers to how many trees make up the ensemble. The F1-score was used as the scoring metric, and the combination of chosen parameters that produced the highest score was to be used to fit the estimator again with the *refit=True* argument. The estimator fit 5 folds for each of 25 candidate combinations, totalling 125 fits, with the optimal results given by the *best\_estimator\_* attribute as *max\_depth=75*, and *n\_estimators=125*. The training time complexity of a single training instance was *O((n)log(n)dk)*, where *k* is the number of trees, *n* is the number of observations, and *d* is the number of dimensions of the data.

The computation environment for all executed code was an AMD64 bit architecture-based Windows 10 system (Version 10.0.19044-SP0), with an 11th Gen Intel Core i5-1145G7 CPU (Family 6 Model 140 Stepping 1) running at a clock speed of 2.60GHz, and 16 GB of RAM.

**Evaluation of Model**

The initial evaluation of the model revolved around the macro F1-score, which was used for model and hyperparameter tuning. This score does not consider the distribution of classification in the data, but weighs each class equally, which is a more representative metric for a balanced dataset. The F1-score is defined as:

*F1 = (2 \* Precision \* Recall) / (Precision + Recall)*  (7)

where recall is the sum of the true positives across classes divided by the sum of the true positives and false negatives across classes, and precision is the sum of the true positives across classes divided by the sum of the true positives and false positives across classes. The macro F1-score is defined as the ratio of the sum of the F1-scores in each class and the total number of classes:

F1Macro = ΣF1class / Number of Classes (8)

The initial macro F1-score of the testing and final score of the validation set was .68 and .62 respectively. The classification reports in *Figure 20* outline the details of the testing and validation processes. Additionally, the training F1-scores for the LR and SVM were .41 and .29 respectively.

Table

Description automatically generated Table

Description automatically generated with medium confidence

Figure 20: Precision, recall, F1-score and support of the training and validation sets. It’s noted that the validation set performed better on predicting unremarkable scripts. The macro average F1-score of the validation set was greater than the test set, suggesting that the model is not prone to overfitting. The poor training recall for unremarkable scripts suggests that the model must improve on minimizing false negative predictions for the class.

The receiver operator characteristic curve (ROC) is another common classification metric, which plots the true positive rate against the false positive rate and displays the performance of a classifier at all threshold levels. Taking the AUC (Area Under the Curve) indicates how well the model performs as compared to a baseline value of 0.5 (i.e., *y=x*) shown in *Figure 21* below, indicating a random guessing predictor, and 0 indicating a completely imperfect predictor.

Chart, line chart

Description automatically generated

Figure 21: An AUC of .68 is indicative that the model performance is better than a random guessing predictor.

**Discussion, Limitations and Conclusion**

One possible limitation to the study was the limited sample size, which was not representative of the complete population of movie scripts. This was partially due to accessibility issues, however, obtaining more international movie titles would better represent this population, and the study could be further expanded to encompass titles that met similar criteria of being award worthy. Another limitation was the *.pdf* parsing package LeXmo was extremely costly in time usage. Other options should be explored for lexicons such that deployment of the model – if need be – would not compromise runtime.

One consideration to be made is that popularity may be determined by uncontrolled factors such as availability, outreach, population size, and interpretability barriers. Although some modeling techniques such as a Bayesian fixed effects model may be able to find a signal for such attributes, this is beyond the scope of complexity that is targeted with this investigation, and does not directly target the aim and motivation.

The aim of this paper was to deconstruct movie scripts that are associated with Oscar nominated films via a subfield of sentiment analysis known as emotion detection (ED) via data scraping, text processing, exploratory text analysis, NLP (Natural Language Processing)/algorithmic implementation, and evaluation of model performance. Lexicon implementation and a refined RF model performed poorly as a hybrid model, with the macro F1-score for both the testing and test sets yielding .54 and .62, respectively, indicating that there was not enough or substantive data gathered to generate enough predictive accuracy to be a viable tool on unseen titles.

**References**

Acheampong, F. A., Nunoo-Mensah, H., & Wenyu, C. (2020, May 28). *Text-based emotion detection: Advances, challenges, and opportunities*. Wiley Online Library. Retrieved August 21, 2022, from <https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.12189>

Ageron. (n.d.). *Ageron/Handson-ml: A series of Jupyter notebooks that Walk you through the fundamentals of machine learning and deep learning in python using Scikit-learn and tensorflow.* GitHub. Retrieved August 26, 2022, from <https://github.com/ageron/handson-ml>

Allouch M, Azaria A, Azoulay R, Ben-Izchak E, Zwilling M, Zachor DA. Automatic detection of insulting sentences in conversation. Paper presented at: Proceedings of the 2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE); 2018:1-4; IEEE.

Behnel, S. (n.d.). *Benchmarks and Speed*. lxml - XML and HTML with Python. Retrieved August 23, 2022, from <https://lxml.de/performance.html#parsing-and-serialising>

E. C. -C. Kao, C. -C. Liu, T. -H. Yang, C. -T. Hsieh and V. -W. Soo, "Towards Text-based Emotion Detection A Survey and Possible Improvements," 2009 International Conference on Information Management and Engineering, 2009, pp. 70-74, doi: 10.1109/ICIME.2009.113.

Follows, S. (2021, July 6). How many films are released each year? TGDM. Retrieved August 28, 2022, from <https://stephenfollows.com/how-many-films-are-released-each-year/>

GeÌron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2nd ed.). O’Reilly.

IEEE. (2022, August 21). *Advancing Technology for Humanity*. IEEE Xplore. Retrieved August 21, 2022, from <https://ieeexplore.ieee.org/Xplore/home.jsp>

Legislative Services. (2022, August 17). *Consolidated federal laws of Canada, Copyright Act*. Copyright Act. Retrieved August 23, 2022, from <https://laws-lois.justice.gc.ca/eng/acts/C-42/Index.html>

Mohammad, D. S. M., & Turney, D. P. (2011, July 10). *NRC Word-Emotion Association Lexicon*. Saif M. Mohammad . Retrieved August 26, 2022, from <https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Random Forest: Introduction to random forest algorithm. Analytics Vidhya. (2022, June 21). Retrieved August 26, 2022, from <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>

*Research intelligence - Scopus Content Coverage Guide*. Elsevier. (2020). Retrieved August 21, 2022, from <https://www.elsevier.com/__data/assets/pdf_file/0007/69451/Scopus_ContentCoverage_Guide_WEB.pdf>

*SimplyScripts Disclaimer*. SimplyScripts. (n.d.). Retrieved August 23, 2022, from <https://www.simplyscripts.com/>

Wang Shangfei, Peng Guozhu, Zheng Zhuangqiang, Xu Zhiwei. Capturing Emotion Distribution for Multimedia Emotion Tagging. IEEE Transactions on Affective Computing. 2019;1–11. <http://dx.doi.org/10.1109/taffc.2019.2900240>

Yang Hui, Willis Alistair, De Roeck Anne, Nuseibeh Bashar. A Hybrid Model for Automatic Emotion Recognition in Suicide Notes. Biomedical Informatics Insights. 2012;5(1):BII.S8948. <http://dx.doi.org/10.4137/bii.s8948>.

Yu, H.-Y., Kim, M.-H., & Bae, B.-C. (2017, December 25). *Emotion and sentiment analysis from a film script: A case study*. Journal of Digital Contents Society. Retrieved August 20, 2022, from <http://koreascience.or.kr/article/JAKO201707356128225.page>