



## ESSAY GRADE PREDICTOR

# NATURAL LANGUAGE PROCESSING



# Table of Contents

<b>1. PROJECT OVERVIEW .....</b>	<b>2</b>
1.1 INTRODUCTION.....	2
1.2 PROBLEM STATEMENT .....	2
1.3 DATASET .....	3
<b>2. RELATED WORK .....</b>	<b>5</b>
<b>3. TEXT PREPROCESSING .....</b>	<b>6</b>
<b>4. FEATURE ENGINEERING.....</b>	<b>7</b>
<b>5. APPROACH AND EVALUTION .....</b>	<b>9</b>
<b>6. FUTURE SCOPE AND CONCLUSION .....</b>	<b>10</b>
<b>REFERENCES.....</b>	<b>12</b>
<b>APPENDIX.....</b>	<b>13s</b>

## 1. Project Overview

### 1.1 Introduction

Essays are very reliable in assessing academic understanding of concepts as well as ideas, depth of language, and fluency in elaborating over concepts. Essays are extremely popular academic assignment and most competitive exams that have language proficiency test have essay questions. Manually grading essays is tedious, more susceptible to human errors, time consuming, and an expensive process. It may not always be possible for a manual grader to maintain the same pattern for evaluating essays since human mind is prone to have an opinion and bias for specific content and concepts.

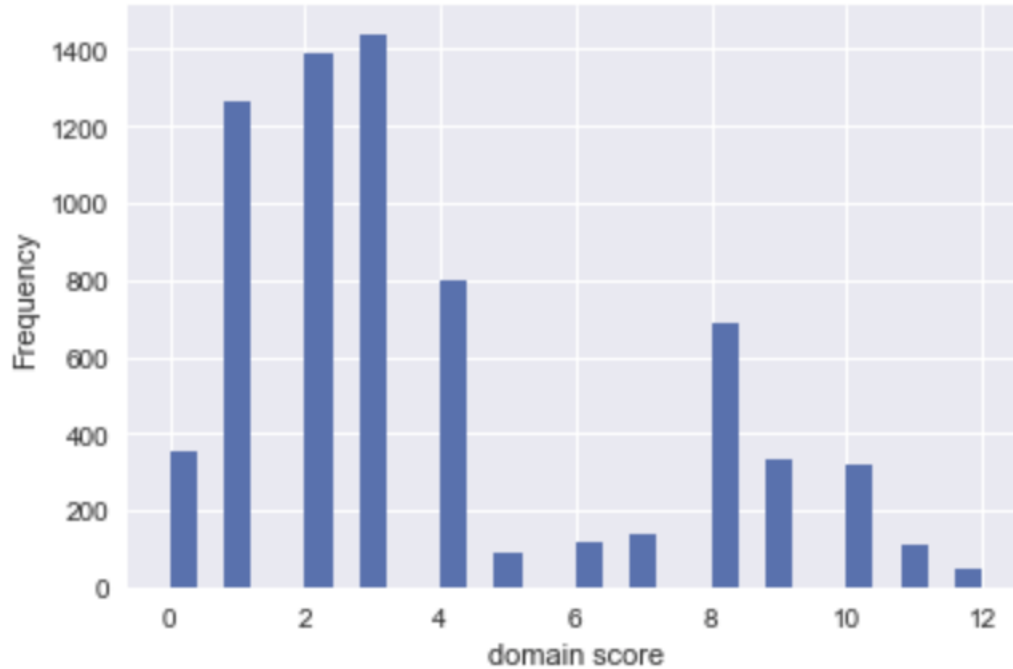
The proposed system aims to eliminate the need for manual grading and automate the process of essay grading by replicating the patterns followed by humans for grading essays and predict the results based on features derived on semantic analysis of the essays. The data was acquired from a Kaggle competition, an open source website, which consisted of 12K essays categorized into 8 different sets of domain. The goal is to train models which predict the most accurate score and use those models for our system to optimize the automated essay grading process.

### 1.2 PROBLEM STATEMENT

Natural Language Processing is one of the most complex problem to be analyzed and solved since it deals with free-form of text, inferences from human interactions and conversations, understanding the context and structure of the content. There are several interpretations of analyzing languages, and each implementation differs from other due to the features considered and underlying understanding of the language and the domain or context of the text that is to be analyzed.

The existing systems that are implemented are based on linear regression models which are well suited for the kind of data to be predicted but do not take into account the correlation among the features, for automating any manual process it is essential to design the system to follow the same instincts that would be followed by human doing it, the implementations in place focus on a large set of features to make the model more efficient and adaptable but the features lose their importance in cases where dimensionality is extremely large.


`Text(0.5,0,'domain score')`

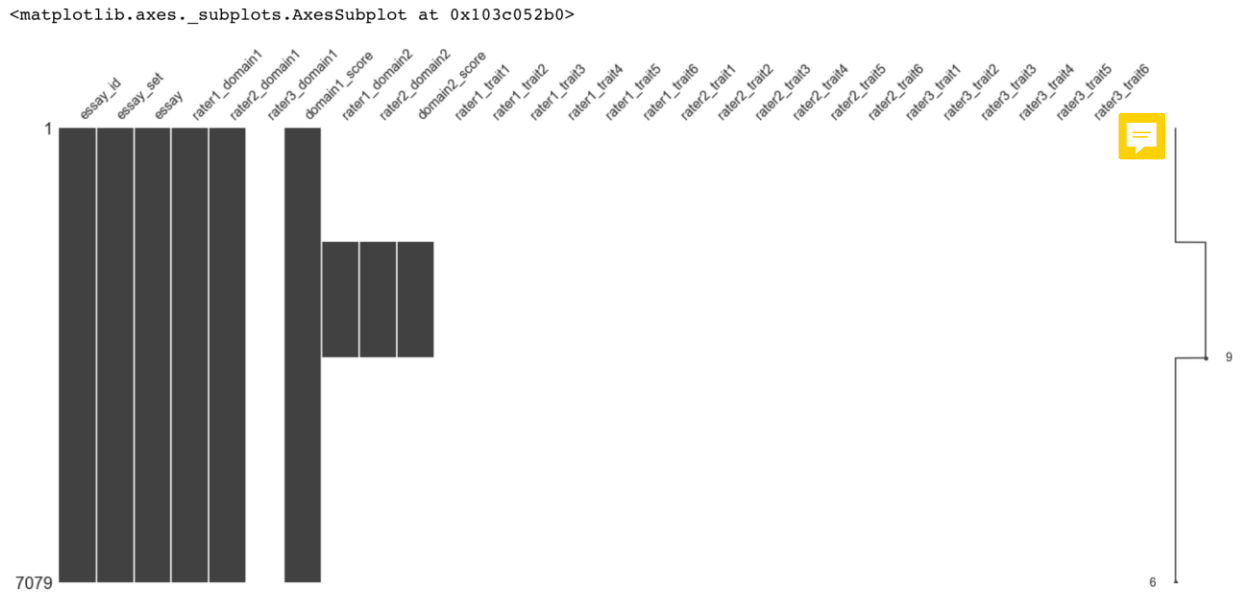


**Fig. 1 Frequency Distribution of Scores**

The focus of our implementation was to make sure that most relevant, correlated and logically features are integrated together while training models. Another important part of the implementation was to design multiple models, we trained our feature set using Linear Regression, Random Forest Regressor, Support Vector Machine and Multinomial Naïve Bayes. The features were created by considering factors related to the word count in the essay, the grammatical errors present in the essay, the score range determining the content of the essay i.e whether the sentences match the average sentence length of other essays from the same domain, the vocabulary used, i.e, the number of adjectives, verbs and nouns present in the composition of the essay and whether the system would be domain specific.

### 1.3 DATASET

The dataset originally contained 12000 essays categorized into 4 sets of persuasive/ narrative and 4 sets of source dependent responses, which is a total of 8 sets. The average word count of each of these essays were approximately 450 words. There were 28 columns present in the original dataset, of which we extracted 4 columns essay id which is a numeric index column, essay set denoting the numeric id of essay set in which the essay belonged, essay containing text data and **domain score containing human evaluations of the essay** 



**Fig. 2 Visualizing missing values in original dataset**

For the ease of implementation and adhering to the hardware restrictions of our systems we restricted the dataset to 4 sets of essays, giving us a count of 7079 essays. For the supervised learning models, the data was split into 80-20 training to testing ratio.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0dab04e0>
```

rater1\_domain2

rater2\_domain2

domain2\_score

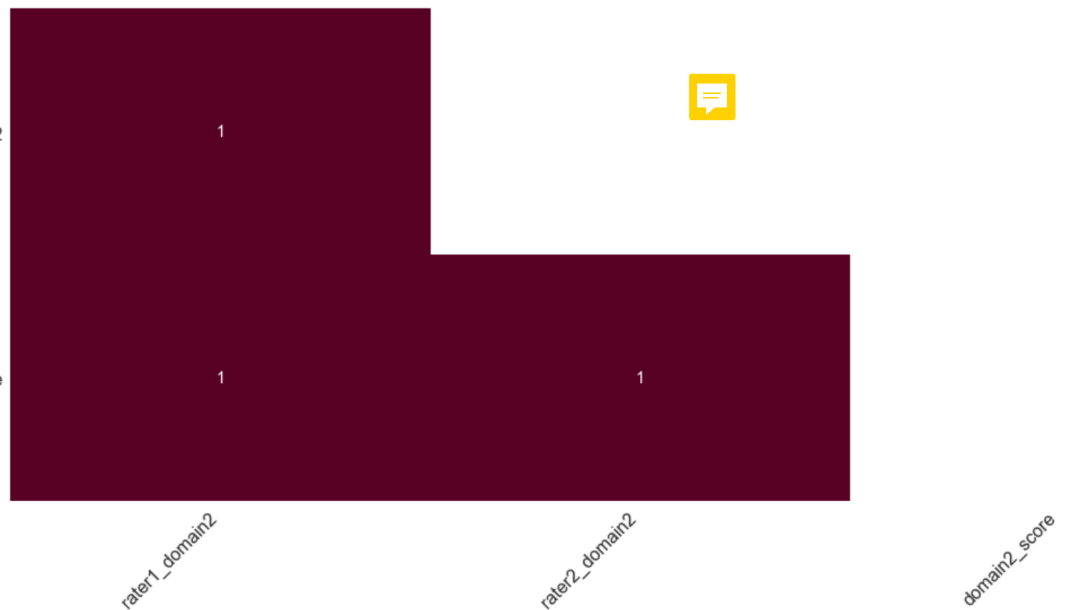


Fig. 3 visualizing correlation between the missing values

## 2. RELATED WORK

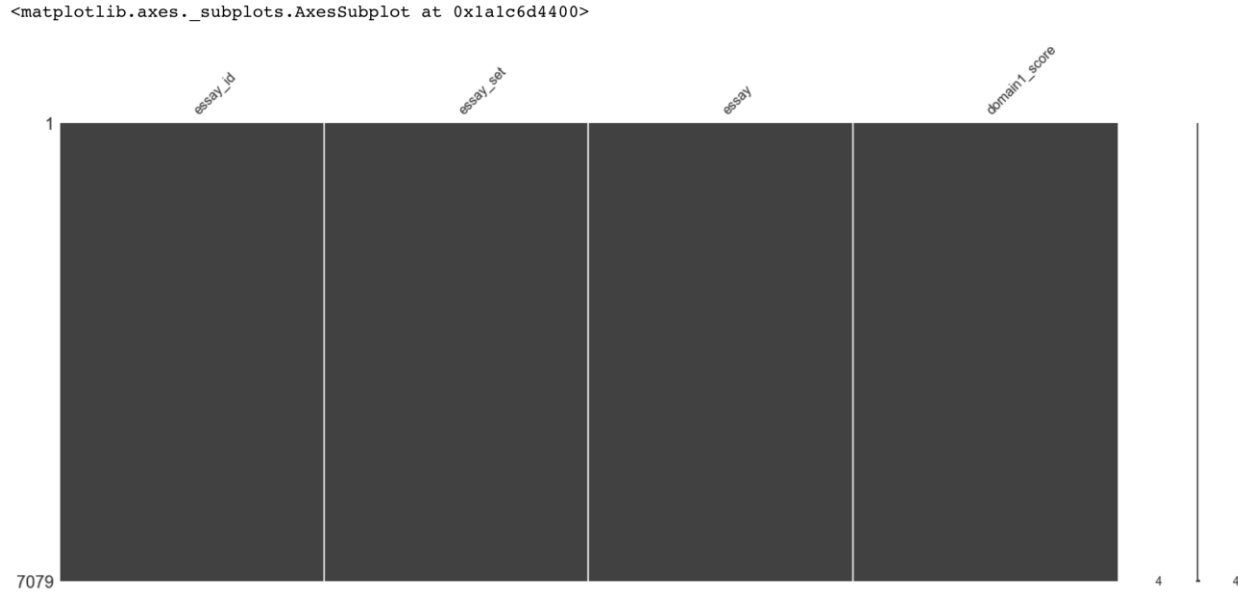
1. ETS Rater: ETS Rater, also called as e-rater is used by competitive exams' organizations like GRE and TOEFL to check language proficiency of students and reduce the evaluation time of essays. These exams have a two level scoring system, where the final grade is calculated by taking an average of the human and machine graded score. The parameters that e-rater focuses on are grammatical errors, vocabulary usage, style of writing and organization of essay. Therefore, they are generally well-positioned to score essays that are intended to measure text-production skills. Many current systems also evaluate the semantic content of essays, their relevance to the prompt, and aspects of organization and flow. [1]Assessment of creativity, poetry, irony, or other more artistic uses of writing is beyond such systems. Another drawback of this system is that they cannot assess rhetorical points, logical ideas or arguments, the relevance of the concept or the entire context of the essay.

2. **Pearson's Intelligent Essay Assessor:** The Pearson's Essay Grader also known as Intelligent Essay Assessor (IES) is another widely used system for automated essay grading based on LSA which focuses on parameters that are in some way drawbacks of the e-rater system like identifying ideas, the logical organization of essay, identifies short hands and slangs that especially essential in domain specific fields, fluency of statements and tone conveyed in essay. The only major drawback of this system is that the features are not correlated to each other and therefore, despite applying best of artificial intelligence algorithms it fails to achieve the an efficient accuracy. Its performance is at par with Linear Regression models.
3. **Automated Essay Scoring:** This system can be considered as generic implementation of E-Rater and IEA, this model summarizes the several implementation techniques that are generally followed while designing an automated essay scoring system. This system focuses on lexicons that include words and phrases associated with different subjectivity dimensions – sentiments, factive verbs, entailments, intensifiers and hedges. The feature sets for these models mostly focuses on domain features that mostly includes parts of speech, grammar and style, organization and development, lexical complexity, word and sentence count, prompt-specific vocabulary usage, domain level semantics, sentiment lexicon and set dependent tasks to rate an essay.

### 3. TEXT PREPROCESSING

The following techniques were implemented for preprocessing the text data:

- a. **De-Anonymization:** The dataset consisted of a few anonymized characters that contained confidential information like name of person, location, name of company, telephone number, etc. We translated these anonymized string using regex to replace relevant part of speech words in place of those characters.
- b. **Remove Stopwords:** A customized list of stopwords was used to remove common prepositions and pronouns from the essays. Since text data is usually really time consuming to preprocess, removing redundant words from data helps a lot with the execution time and efficiency.



**Fig. 4 Feature Preprocessing**

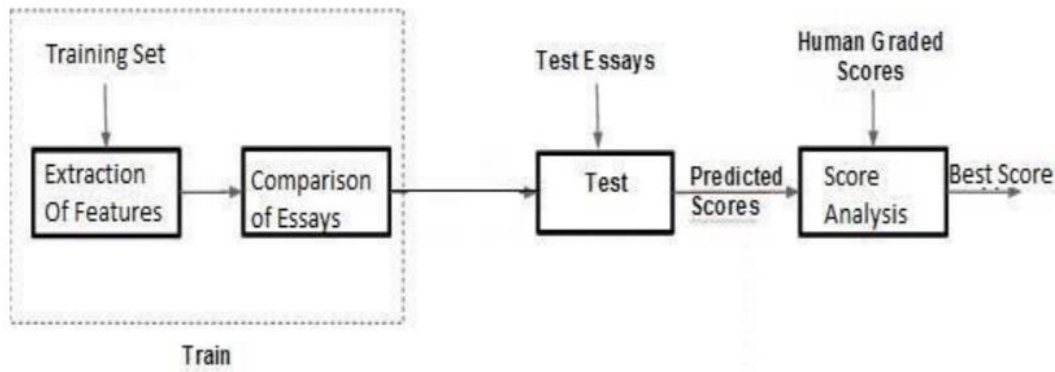
- c. **Remove Punctuations:** To improve the quality of text before tokenization, punctuations were removed from essays and the text was converted to lower case so that the features to be created using this column would have optimized values.
- d. **Tokenization:** Using the nltk sentence and word tokenizer, the essay which was in the form of paragraphs was divided into sentence and word tokens which are integral in deriving other features to train the model.
- e. **Lemmatization:** Using nltk WordNet Lemmatizer, we reduced the tokenized words a morphological format, by lemmatizing.
- f. **POS Tagging:** Using nltk pos tagger, part of speech was identified over the lemmatized set of words. This was essentially done to create features for the machine learning algorithms.

## 4. Feature Engineering

Feature Extraction is the essence of training any supervised learning algorithm, having the right set of features is the building block of the model. The focus of this implementation is on features related to language fluency, grammatical and syntactic correctness, vocabulary and types of words used, length of essay and words, etc. The following were the features created for training Essay Grade Predictor Model:



- a. Word count: Every academic and assessment essay has a word count criteria which defines a limited number of words to be present or a range of number of words in the essay that impacts the score. This feature is created by counting the words from lemmatized words column.
- b. Average words: After removing stopwords, punctuations and redundant words from sentences, it is an implicit expectation that the sentence should not be left with 1 or 2 words. This feature counts the average number of words present in the tokenized sentences, and is a highly correlated feature, in almost every model.
- c. Distinct words: To distinguish good and average essays, it becomes an area of interest that how many distinct words are used in writing the essay. This feature is created by retrieving count of unique words from lemmatized words of essays.
- d. Grammar check: Spelling mistakes and grammatically incorrect sentences are the reasons for most deductions in scores of essay. By using the language\_check tool, a grammatical error score for each essay was calculated.



**Fig 5: Feature Engineering**

- e. Adjective word: After analyzing the part of speech words dictionary obtained by preprocessing, it was found that adjectives are most correlated to the target variable. There a count of adjectives used in essays was made a feature.
- f. Vectorized words: These features are the string features present in the text that need to be translated into vectorized format for the supervised learning algorithms to train the model. The values from the list of adjective words was vectorized for all the algorithms since adjectives were identified as a highly correlated feature.

As specified earlier, the feature columns that were extracted from the original dataset were essay id, essay set id, essay, domain score and the newly created features are word count,

average word count, distinct words, grammar check, adjective word and to be determined number of vectorized feature words columns.

## 5. APPROACH AND EVALUATION

The proposed implementation considers features as all the parameters that could result or have an impact in deciding the score. The target variable to be predicted that is domain score is a column with discrete values and therefore supervised machine learning models like Linear Regression, Support Vector Machine (SVM), Random Forest Regressor, Multinomial Naïve Bayes would work exceedingly well with the type of target variable. We have implemented these four models with different combinations of features following is the detailed analysis of the experiment,

Modeling Techniques:

Predictor Variable: domain\_score

Independent Variable: essay, word count, average word, distinct word, grammar check, adjective word count

1. Multinomial Naive Bayes (MNB): MNB is a specific instance of a Naïve Bayes Classifier which uses a multinomial distribution for each of the features. It is a probabilistic classifier, therefore it will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be the output. The techniques used to implement this model are Vectorizer and Lasso. Vectorizer is used to convert string values to numeric values whereas Lasso is used for feature selection. Lasso Regression was used to find features with high correlation with the predictor variable.

The accuracy of 61.22% is achieved with MNB.

2. Support Vector Machine: [4] In machine learning, SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new example to one category or the other, making it a non-probabilistic binary linear classifier. The techniques used are Vectorizer and Gradient search. Gradient descent is used to minimize function by iteratively moving in direction of steepest function. The optimal values of gamma and c were obtained from gradient descent and it worked much better than applying brute force. The model was trained for  $c=0.001$  and  $\text{gamma} = 0.0001$

Accuracy: 61%

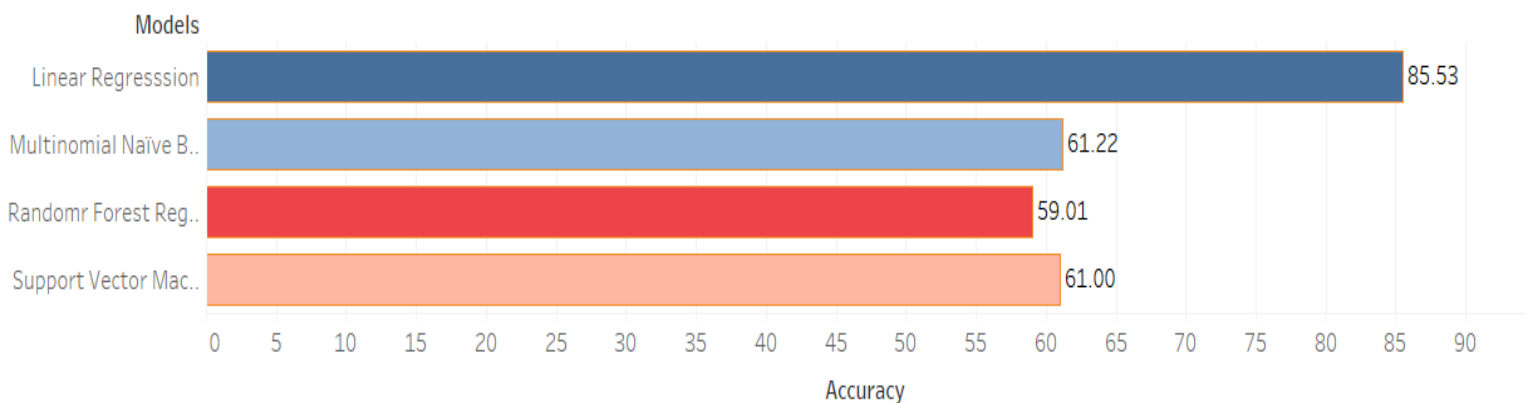
3. Random Forest Regressor: It is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Techniques- Chi-Square, Vectorizer. [2]Chi-square is used to verify the fitting of test data and to avoid overfitting. The first step was to find highly correlated feature

sets by using Kselect by setting the value of K to 3 and then comparing the features obtained from that with the predictor variable.

Accuracy: 59.01

4. Linear regression: Is a linear approach to modeling the relationship between a scalar response and one or more explanatory variables. Since the target variable is a list of discrete values Linear Regression performed really well. The features had high covariance among themselves which made Linear Regression the best choice of algorithm, PCA analysis was performed in in order to avoid multicollinearity between independent variables and to select appropriate features.

Accuracy: 85.53%



**Fig 6. Model Comparison**

## 6. Future Scope and Conclusion

Enhance models by adding task similarity features to detect plagiarism, features to detect and analyze essay domain. Going through a research paper we came across that there are two types “copy paste” and “paraphrasing” of plagiarism techniques. Cosine metric factor can be used to illustrate the relevance among the documents as in most of the cases the user modulates the content in different style and word which allows the detection tool to report negatively. Along with that in future we can work on detecting if the user is entering the content for instance, the original content is in text form and represents it in tabular form. The approach should also detect if the reference provided by the user is valid or not.

Implement Deep Learning and neural networks for better accuracy and essay grading. Implementation of neural network might remove the dependency on feature engineering and might automatically learn the representations required for the task to be graded. Neural network is being successfully implemented by researchers to effectively utilize the content of essay to extract the required information for grading it.

Use semantic parsers by identifying good semantic and syntactical features to implement polynomial basis functions like neural networks. Combining these two techniques will enhance the performance and might make the comparisons more direct and accurate. Introducing a neural network architecture can provide computationally efficient semantic role tagging.

Automated Essay Grading model is highly efficient and effective in overcoming human errors and saving time. With the new trends in Natural Language Processing, this model can be perfected to score like a human user, there are some systems currently working on integrating emotions and adaptiveness to these systems, so that the grading becomes more aligned to human grading. The model implemented as a part of this project tries to bridge in the gap between the existing models and the new research that is currently going on. The results obtained from these models are from first round of preprocessing and feature selection and therefore, are better than expected since the performance of Linear Regression Model matches the industry standards and the existing systems in use.

## REFERENCES

- [1] ResearchGate | Share and discover research. (PDF) Task-Independent Features for Automated Essay Grading. Retrieved from [http://www.researchgate.net/publication/278383803\\_Task-Independent\\_Features\\_for\\_Automated\\_Essay\\_Grading](http://www.researchgate.net/publication/278383803_Task-Independent_Features_for_Automated_Essay_Grading)
- [2] Jill Burstein, Claudia Leacock, and Richard Swartz. 2001. Automated evaluation of essays and short answers. Loughborough University Press.
- [3] Yigal Attali and Jill Burstein. 2006. Automated essay scoring with e-rater v. 2. The Journal of Technology, Learning and Assessment, 4(3).
- [4] Robert Östling. 2013. Automated essay scoring for swedish. In Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications, pages 42–47.
- [5] Mark D Shermis and Jill C Burstein. 2002. Automated essay scoring: A cross-disciplinary perspective. Routledge.
- [6] Manvi Mahana, Mishel Johns, and Ashwin Apte. 2012. Automated essay grading using machine learning.
- [7] Salvatore Valenti, Francesca Neri, and Alessandro Cucchiarelli. 2003. An overview of current research on automated essay grading. Journal of Information Technology Education: Research, 2(1):319–330.
- [8] IOPscience. Retrieved from <http://iopscience.iop.org/article/10.1088/17426596/1000/1/012030/pdf>  
Retrieved from <http://www.aclweb.org/anthology/D15-1049>
- [9] Ellis Batten Page. 1994. Computer grading of student prose, using modern concepts and software. The Journal of Experimental Education.
- [10] Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python. O'Reilly Media.
- [11] Mark D. Shermis and Jill Burstein, editors. 2013. Handbook of Automated Essay Evaluation: Current Applications and New Directions. Routledge.  
Technology Education: Research, 2(1):319–330.

## APPENDIX

## TEXT PREPROCESSING:

```

In [6]: data = dataTrain[['essay_id', 'essay_set', 'essay', 'domain1_score']]

In [7]: data.head()
Out[7]:
  essay_id  essay_set  essay  domain1_score
0         1         1  Dear local newspaper, I think effects computer...      8
1         2         1  Dear @CAPS1 @CAPS2, I believe that using compu...      9
2         3         1  Dear, @CAPS1 @CAPS2 @CAPS3 More and more peopl...      7
3         4         1  Dear Local Newspaper, @CAPS1 I have found that...     10
4         5         1  Dear @LOCATION1, I know having computers has a...      8

In [8]: len(data)
Out[8]: 7079

In [9]: data=data.dropna()

In [10]: len(data)
Out[10]: 7079

In [11]: data=data.replace({'@CAPS\d': 'Bhatia', '@LOCATION\d': 'Victoria', '@PERCENT\d': '80', '@ORGANIZATION\d': 'D-GANG', '@PERSON\d': 'Harsh', '@
Out[11]:
In [12]: data[:10]
Out[12]:

```

```

In [13]: fstop = open('Smart.English.stop', 'r')
stoptext = fstop.read()
fstop.close()
stopwords = nltk.word_tokenize(stoptext)
len(stopwords)
Out[13]: 577

In [14]: import string

In [15]: data["remove_nonalpha"] = data.essay.apply(lambda x: x.lower())
data.remove_nonalpha = data.remove_nonalpha.apply(lambda x: x.translate(string.punctuation))
data.remove_nonalpha = data.remove_nonalpha.apply(lambda x: x.translate(string.digits))

In [16]: #Sentence Tokenization
data['tokenized_sents'] = data.apply(lambda row: nltk.sent_tokenize(row['remove_nonalpha']), axis=1)

In [17]: #Word Tokenization
data['tokenized_words'] = data.apply(lambda row: nltk.word_tokenize(row['remove_nonalpha']), axis =1)
#data['tokenized_words'] = data['tokenized_words'].apply(lambda x: [item.lower() for item in x])

In [18]: data['removed_stop'] = data['tokenized_words'].apply(lambda x: [item for item in x if item not in stopwords])

In [19]: lemmatizer = nltk.stem.WordNetLemmatizer()

def lemmatize_text(text):
    return [lemmatizer.lemmatize(w) for w in text]

data['text_lemmatized'] = data.removed_stop.apply(lemmatize_text)

In [20]: #Tagged sentences - CHECK THIS AND DECIDE IF WE SHOULD CONSIDER ADJECTIVES USED IN ESSAYS
data['pos_tagged'] = data.apply(lambda row: nltk.pos_tag(row['text_lemmatized']), axis =1)

```

## FEATURE ENGINEERING

```

In [21]: #Word Count
data['word_count'] = data['text_lemmatized'].apply(lambda x: len(str(x).split(" ")))

In [22]: #Average Word Length in sentence
def avg_word(sentence):
    words = sentence.split()
    return (sum(len(word) for word in words)/len(words))

In [23]: data['avg_word'] = data['essay'].apply(lambda x: avg_word(x))

In [24]: #unique words in essay
data['distinct_words']=data['essay'].apply(lambda x: pd.value_counts(x.split(" ")).sum(axis = 1))

In [25]: #grammar and language checker
import language_check

In [26]: tool = language_check.LanguageTool('en-US')

In [27]: match = []
for item in data['essay']:
    val = tool.check(item)
    match.append(len(val))

In [28]: data['grammar_check'] = match

```

## LINEAR REGRESSION

```

In [60]: # model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("-----")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("-----")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

The model performance for training set
-----
RMSE is 1.1972336857309591
R2 score is 0.8436807778744083

The model performance for testing set
-----
RMSE is 1.137874059310308
R2 score is 0.8553416652017354

```

## MULTINOMIAL NAÏVE BAYES

```

In [200]: predicted = model.predict(xtrain_NB)

In [202]: expected = ytrain_NB

In [203]: from sklearn.metrics import confusion_matrix
          mat = confusion_matrix(expected, predicted)

In [180]: from sklearn.metrics import classification_report

In [207]: print(classification_report(expected, predicted))

```

	precision	recall	f1-score	support
0	0.12	0.18	0.14	350
1	0.36	0.71	0.48	1267
2	0.35	0.10	0.16	1390
3	0.53	0.17	0.26	1440
4	0.50	0.73	0.60	795
5	0.25	0.45	0.32	92
6	0.00	0.00	0.00	117
7	0.16	0.17	0.16	135
8	0.44	0.62	0.51	687
9	0.30	0.13	0.18	334
10	0.32	0.34	0.33	316
11	0.20	0.27	0.23	109
12	0.17	0.28	0.21	47
avg / total	0.39	0.37	0.33	7079

## RANDOM FOREST

```

In [124]: # Use the forest's predict method on the test data
          predictions = rf.predict(test_features)
          # Calculate the absolute errors
          errors = abs(predictions - test_labels)
          # Print out the mean absolute error (mae)
          print('Mean Absolute Error:', round(np.mean(errors), 2))

          Mean Absolute Error: 1.21

In [128]: # Limit depth of tree to 3 levels
          rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
          rf_small.fit(train_features, train_labels)
          # Extract the small tree
          tree_small = rf_small.estimators_[5]

In [129]: # Get numerical feature importances
          importances = list(rf.feature_importances_)
          # List of tuples with variable and importance
          feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature_list, importances)]
          # Sort the feature importances by most important first
          feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
          # Print out the feature and importances
          [print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];

          Variable: avg_word                Importance: 0.61
          Variable: word_count              Importance: 0.23
          Variable: grammar_check           Importance: 0.1
          Variable: distinct_words          Importance: 0.07

In [143]: from sklearn.metrics import r2_score
          print(r2_score(test_labels, predictions))

          0.5901776711764035

```



## SUPPORT VECTOR MACHINE

```

In [51]: #Import Library
from sklearn import svm
from sklearn.svm import SVC
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = svm.SVC(kernel='linear', C=0.001, gamma=0.0001)
# there is various option associated with it, Like changing kernel, gamma and C value. Will discuss more # about it in next section
model.fit(train, Y)
model.score(train, Y)
#Predict Output
predicted= model.predict(train[:])

```

```

In [50]: X = train
Y = data.iloc[:,3]

```

```

In [52]: from sklearn.metrics import classification_report
y_true = Y
y_pred = predicted
print(classification_report(y_true, y_pred))

```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	350
1	0.60	0.80	0.69	1267
2	0.59	0.63	0.61	1390
3	0.51	0.62	0.56	1440
4	0.51	0.63	0.56	795
5	0.00	0.00	0.00	92
6	0.00	0.00	0.00	117
7	0.00	0.00	0.00	135
8	0.47	0.46	0.46	687
9	0.28	0.09	0.14	334
10	0.35	0.46	0.40	316
11	0.00	0.00	0.00	109
12	0.00	0.00	0.00	47