Contents

Abstract

In this project, our main goal was to make an automated essay grader. We used a dataset of essays written by high school students on various types of exam questions. All of them were graded by the instructors. In order to find a relationship in between scores and texts, we extracted features like lemma-noun-sentence count, average word length and etc. We used those features along with the scores to form both a regression and a classification model. When we evaluated them, we saw that we can predict the scores with within an acceptable range. (Word count: 94)

Introduction

In order to determine proficiency in verbal & quantitative reasoning and qualitative analysis, writing is a tool that has been used since primary school and it is the first thing to be learned in schools to transfer our ideas. In school format, students write essays for evaluation. However, grading of these essays is a time consuming operation to satisfy non-biased, full care reading that will satisfy fairness. Also, the time in which students receive their feedbacks is another trade-off of this costly operation. Since 30-40 years, with the improvements in computer technologies and specifically natural language processing, automated essay scoring systems have been developed.

In automated essay grading systems, quality of the writing is determined mostly by the essay grades given by human graders to the trained datasets. From these datasets, statistical analysis, feature extractions and dimensionality reductions are conducted to understand what creates an impression of good essay in the eyes of human graders. Later on, using these attributes obtained, scoring of humans is simulated. If the grades given by the system shows statistical significance with those of humans and determines accurately what features contribute to that grading, it is considered successful. Diversity between these systems can be due to the different techniques applied for each step like feature extraction. Alikaniotis et al. states the quality of text is influenced by many criteria and automated essay grading systems usually focus on the broad range of textual attributes like vocabulary, style, syntax etc. (Alikaniotis et al., 2016). On top of these, features like complexity of sentence and grammatical relations are used to refine the grading.

Current approaches of grading mostly focus on regression for predictions along with the features that best fit to regression. However, a feature that fits to one type of content might be irrelevant for another. Assessing these different domain essays altogether causes decrease in accuracy or sampling too much features that further complicates the grading algorithm which results with overfitting. For simplistic models like ours to work on low computational power and yield accurate results, separation of essays based on their context, school-grade must be accomplished and any exponential complications like multiple-order Markov chains must be avoided.

In our project, we have focused on simple structural features of the text like sentence, word, character, adjective counts and more advanced structural features like lemma and misspell count. For the modeling of these features, we used linear regression and adaptive boosting (AdaBoost) regression. For classification, lower and upper quantiles are used as FAIL and SUCCESS criteria and middle %50 is considered as PASS. Also

since linear kernel couldn't separate non-linear data well, SVM with RBF kernel is used in classification.

The sections in the rest of the paper are organized as follows: Related Work, Dataset, Feature Extraction, Machine Learning Methods; Regression & Classification and Conclusion and Discussion.

Related work

History of automated essay grading systems dates back to 1960s and there are many different models. In 1966, Page developed Project Essay Grader in which grading is done based on fluency and grammar independent of content. In 1972, Jones developed vector-space model that uses term-document matrix. Thanks to that paper, TF-IDFs made its first introduction into automated essay grading field. Evaluation of overall essays was worse compared to older work due to usage of cosine similarity. In 1999, IEA is developed by Foltz et al. in which structural features are processed together with contextual features in latent semantic analysis word vector representations. In 1990, e-rater a product of ETS is introduced and further developed in 2003 and 2006 by Burstein et al. which extended the system into combination of multiple systems like Bayesian Essay Test Scoring System of Rudner. In e-rater, features like grammar and discourse components are also combined (Islam & Hoque 2010) (Farra et al., 2015).

In recent years, state of art methods like discourse coherence quality assessment, persuasiveness assessment, lexical chaining, word association technique are implemented. Klebanov, Flow and Somasundaran obtained significantly high essay scores in separate papers. For argumentation, Stab and Gurevych used a corpus and scheme for annotation in which the persuasiveness of essays is evaluated with the stances of supportiveness and non-supportiveness. Also, Klebanov detected sentiments in sentences with the help of multi-word expressions. Bin L. et al. integrated the K Nearest Neighbour algorithm on automated essay grading and turned the essays into VSMs. Using TF-IDFs and IG with a small corpus of text, precision more than %75 is reached (Islam & Hoque 2010) (Farra et al., 2015) (Alikaniotis et al., 2016). Most of these researches have been conducted for English language where extensive corpuses, tools and studies were present before to be used. But there were studies for other languages like Arabic, too. Nahar K. M. et al. implemented a system for online exams in Arabic and obtained 60% precision. In our work, we followed a minimalistic approach in which the features used must operate on non-exponential complexity but still achieve a relatively consistent accuracy with

other studies. Thanks to that implementation, its results can be interpreted even by an average teacher without confusion.

Dataset

In this project, we used a dataset from an old Kaggle Competition from 2012: The Hewlett Foundation, Automated Essay Scoring. There are eight different essay sets which have texts ranging from 100 to 600 words. They each consist of around 1500 samples. These sets are divided according to the level of the student and the differences in their tasks. Some of them are dependent on a certain topic whereas others are general writing. They are written by students from 7^{th} grade to 10^{th} grade. More detailed information can be seen from the table below:

Essay set no	Type of the Essay	Grade Level	Number of Samples	Min domain1 score	Max domain1 score
1	persuasive / narrative / expository	8	1783	2	12
2	persuasive / narrative / expository	10	1800	1	6
3	source dependent responses	10	1726	0	3
4	source dependent responses	10	1772	0	3
5	source dependent responses	8	1805	0	4
6	source dependent responses	10	1800	0	4
7	persuasive / narrative / expository	7	1569	0	30
8	persuasive / narrative / expository	10	723	0	60

Table 1: Dataset description

Essays are hand graded. There are three raters and each assign a score as "rater1_domain1", "rater2_domain1" ... Not all essays are graded by three raters but there are at least two scores given per each. There is also metric called "domain1_score" which is the resolved grade between all raters. All essays have this

and that's why we used that score in our experiments. There are other metrics as "domain2_score" and etc. but it is only available in a fraction of the dataset.

The grading criteria are not same for the sets. For some of them, scores range in between 0-3 whereas for others 0-60. This would create a problem when we combine them to form a general model as error ranges will not be same. That's why we squeezed all scores in the range of 0-3 before we make our experiments.

Essays are not in raw format. They were pre-processed by the organizers of the competition. They anonymize the texts by trying to remove personally identifying information. They believe that names, locations and etc. would create an unnecessary noise in the data. They used the Named Entity Recognizer (NER) from the Stanford Natural Language Processing group and a variety of other approaches. Some of these entities are PERSON, ORGANIZATION, LOCATION, DATE, TIME, MONEY, PERCENT, MONTH, CAPS, CITY, and STATE. When a relevant entity is recognized, it is replaced with a text "@LOCATION1". The sentence "once my family took my on a trip to Springfield" is edited to "once my family took me on a trip to @LOCATION1". Similarly, "my phone number is 555-2106" to "...my phone number is @NUM1". The integer values next to the entities are increased when a new entity in that category is detected. Even though, these operations are made, misspellings are not corrected to alter the originality of the data.

Feature extraction

Series of features are extracted from the texts using Stanford's Natural Language Processing Toolkit and pyspellcheker.

- Average word length
- Sentence count
- Word count
- Character count
- Noun count
- Adjective count
- Verb count
- Adverb count
- Lemma count
- Misspell count

There are in total ten features and only they are used for regression and classification tasks. Even though, they are mostly structural, they give huge insights on the quality of the essays.

Machine Learning Methods

We trained both regression and classification models on the features that we extract on the previous step. For each set, an individual model is trained and evaluated for both tasks. We randomly split them to train (70%), test (30%) and then evaluate the models. In the end, a more general analysis with all sets combined are made with similar processes.

For regression task, we used the squeezed scores (0-3) from the previous step. Methods: linear regression and AdaBoost regression are used for predictions. For classification, each set is relabelled according to its score distribution. If the score is below 25^{th} percentile, it is labelled as 0 (fail) and if it is above 75^{th} percentile, 2 (good). The remaining ones in the middle are labelled as 1 (pass). SVM with RBF kernel is used to make classification afterwards.

Regression

We used Linear Regression and an ensemble method called AdaBoost Regression for making predictions. AdaBoost is a meta-estimator that begins by fitting the model on the original dataset. Afterwards, it adds additional copies from the same dataset but with weight assigned according to the errors on the previous step. It performed slightly better than linear regression for some sets. We were expecting it to be much better in all cases as it can learn more compared to other method but the results show no causality.

For this task, we used two evaluation metrics: mean squared error (MSE) and R2 score. MSE is the average of the squared errors. It is the simplest metric showing the squared deviation from the ideal value on both sides. R2 score represents the goodness of the fit. It is in between 0 and 100. 0 indicates that the model shows no variability to the data around its mean whereas 100 indicate full. On average, R2 scores for our models are around 60, showing that our results are on the good side.

$$MSE = \frac{\sum_{i=1}^{n}(Ideal - Prediction)^2}{n} (n = \#\ of\ samples)$$

$$R2 = \frac{Explained\ variation}{Total\ variation}$$

The results can be seen below:

Essay set no	Linear Regression		AdaBoost Regression	
	MSE in percentage	R2 Score	MSE in percentage	R2 Score
1	1.63	0.71	1.47	0.74
2	2.34	0.52	2.43	0.50
3	10.9	0.50	12.8	0.42
4	11.6	0.60	14.0	0.52
5	5.5	0.66	5.4	0.67
6	6.85	0.54	8.42	0.44
7	5.35	0.54	5.27	0.55
8	1.16	0.58	1.34	0.51
9 (All sets combined)	10.9	0.28	8.55	0.44

Table 2: Results for the regression models under different scenarios

Classification

We used SVM with RBF kernel for classification. Linear SVM was not performing good as the data cannot be separated linearly in a proper way. That's why we used an RBF kernel to capture the nonlinear points.

$$\begin{aligned} K\left(x^{(i)}, x^{(j)}\right) &= \phi(x^{(i)})^T \phi(x^{(j)}) \\ &= \exp\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right), \qquad \gamma > 0 \end{aligned}$$

It is almost same as the Gaussian kernel with gamma replacing the $1/2\sigma^2$. For most of the sets, it performed quite well when gamma was equal to the 10^{-5}. We found this

number by trial and error. There are some heuristics to find an optimal value but we simply tried values from 1 to 10^{-7} and chose the best as our parameter.

We used precision, recall, f1-score and kappa score as metrics for our evaluation. First three of them are very well known by everyone. Kappa score is equally used a lot but it is less known compared to others. According to Cohen's article, kappa scores lower than 0 indicates no agreement. Scores: 0–0.2 as slight, 0.2–0.4 as fair, 0.4–0.6 as moderate, 0.6–0.8 as substantial, and 0.81–1.00 as almost perfect agreement in between predictions and the labels. In most of the texts, scores larger than 0.8 are considered as an excellent result. On average, our kappa scores are around 0.55, showing that our model is not perfect. However, the other scores are in the range of 0.70. The results are shown below:

Essay set no	Precision	Recall	F1-Score	Kappa score
1	0.79	0.79	0.79	0.62
2	0.74	0.74	0.74	0.53
3	0.76	0.78	0.77	0.55
4	0.72	0.72	0.70	0.52
5	0.74	0.73	0.73	0.56
6	0.74	0.76	0.74	0.45
7	0.65	0.64	0.63	0.39
8	0.62	0.63	0.62	0.41
9 (all sets combined)	0.63	0.63	0.61	0.37

Table 3: Results for the classification model

Conclusion and Discussion

We further investigated our models by making a simple exploratory data analysis. When we take a look to the variable importance (weightings of the features), we have seen that for the most sets, lemma count is the highest. Lemma count represents the total distinct words in an essay. Essays with higher lemma count scored more. This

makes sense as for the most of the time essays with richer content are better. On the other hand, average word length is on the lower side. This is also quite logical as it is quite random and almost same for the most essays.

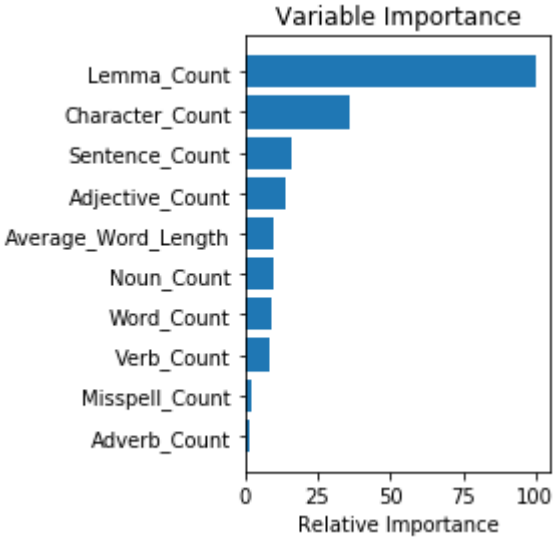

Figure 2: Variable importance in regression model on overall

When we fit a distribution to certain features and labels, we can see that some of them are more distinct and have pattern. It can be seen from the figures below:

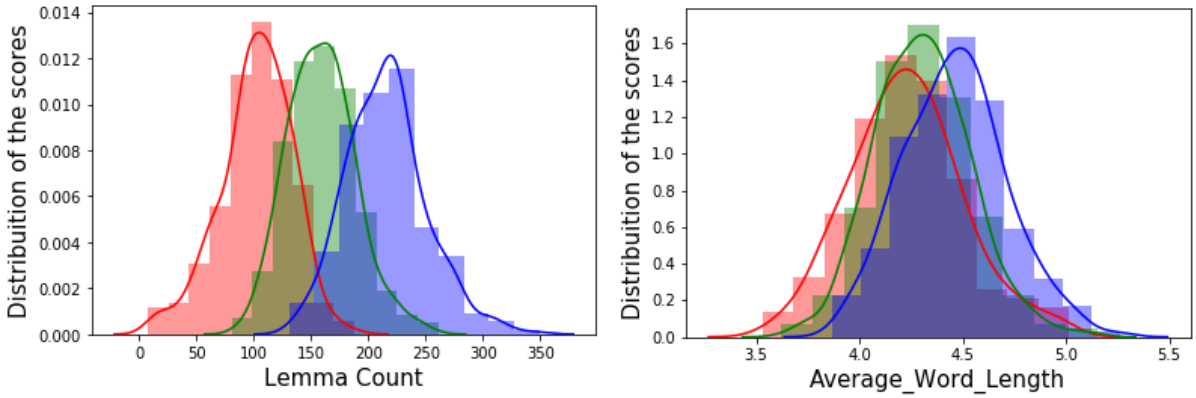

Figure 2: Distribution fitted for features: lemma count and average word length (red: fail, green: pass and blue: good) for set 1

Our results in classification models are not consistent. This might be caused by our biased scoring scheme. While we were converting essay scores to labels, we divided scores to three with respect to the score distribution. If a student is on the lower side (below 25^{th} percentile), his/her essay is labelled as 0 (fail). Not all exams are in this format. For some tests, everyone can get a good grade and pass.

The results for regression models are quite good. For most of the training scenarios, the MSE is below 10% meaning that we can predict the scores with minor error. Our models performed much better for persuasive, narrative and expository essays whereas it's worse for source dependent ones.

Apart from misspell count, the time required to extract features are not much. Misspell count is the slowest as it checks every word and compares it with a huge English corpus. There isn't any good alternative to this approach to make an improvement. Furthermore, it lacks the new words that technology brought to us like Snapchat, Instagram and etc. Despite these disadvantages, there is no good alternative to replace it with.

Future Work

Even though, our model investigates the essays on the structural properties only, it performs quite well in terms of our scoring metrics. It might perform better if the content of the essay could be analysed as well. Essays can be vectorised using Bag of Words or TF-IDF approach and they can be used along with the structural features. Even now, models are quite slow as feature extraction process is costly. It might unnecessarily increase the time required to complete the task. N-grams can be used to capture the flow as well.

Word Count and Other Properties

Word count:	2554
Sentence count:	172
Character count:	13047
Noun count:	694
Adjective count:	202
Verb count:	383
Adverb count:	84
Lemma count:	724
Average word length:	4.83895

References

Alikaniotis, D., 2016, "Automatic Text Scoring Using Neural Networks," pp. 715-725 in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, Germany: Association for Computational Linguistics.

Islam, M., 2010, "Automated Essay Scoring Using Generalized Latent Semantic Analysis," pp.616-626 in: 13th International Conf. on Computer and Information Technology , Dhaka, Bangladesh.

Farra, N., 2015, "Scoring Persuasive Essays Using Opinions and their Targets," pp. 64-74 in Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications, Denver, CO: Association for Computational Linguistics.

This is work done in partial fulfilment of the requirements for the course CS 578 - Natural Language Processing. The software developed is original to a large extent. It was written by the members of the project group and is not simultaneously being submitted to another course.

Appendix

Python code:

```
import nltk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
import re, collections
from collections import defaultdict
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import mean_squared_error, r2_score, cohen_kappa_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from spellchecker import SpellChecker
from nltk.tokenize import word_tokenize
import string
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams

## Loading data

#dataframe = pd.read_csv('all_essaysets.csv', encoding = 'latin-1')
dataframe = pd.read_csv('training.tsv', encoding = 'latin-1', sep='\t')
dataframe.describe()

dataframe.head()

## Methods

# selecting which set to be used 1-8
# in order to combine them all assign set number to 9
def select_set(dataframe,setNumber):
    if setNumber == 9:
        dataframe2 = dataframe[dataframe.essay_set ==1]
        texts = dataframe2['essay']
        scores = dataframe2['domain1_score']
        scores = scores.apply(lambda x: (x*3)/scores.max())
        for i in range(1,9):
            dataframe2 = dataframe[dataframe.essay_set == i]
            texts = texts.append(dataframe2['essay'])
            s = dataframe2['domain1_score']
            s = s.apply(lambda x: (x*3)/s.max())
            scores = scores.append(s)
    else:
        dataframe2 = dataframe[dataframe.essay_set ==setNumber]
        texts = dataframe2['essay']
        scores = dataframe2['domain1_score']
        scores = scores.apply(lambda x: (x*3)/scores.max())
    return texts, scores

# get histogram plot of scores and average score
def get_hist_avg(scores,bin_count):
    print(sum(scores)/len(scores))
    scores.hist(bins=bin_count)

#average word length for a text
def avg_word_len(text):
```

```
    clean_essay = re.sub(r'\W', ' ', text)
    words = nltk.word_tokenize(clean_essay)
    total = 0
    for word in words:
        total = total + len(word)
    average = total / len(words)

    return average

# word count in a given text
def word_count(text):
    clean_essay = re.sub(r'\W', ' ', text)
    return len(nltk.word_tokenize(clean_essay))

# char count in a given text
def char_count(text):
    return len(re.sub(r'\s', '', str(text).lower()))

# sentence count in a given text
def sent_count(text):
    return len(nltk.sent_tokenize(text))

#tokenization of texts to sentences
def sent_tokenize(text):
    stripped_essay = text.strip()

    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    raw_sentences = tokenizer.tokenize(stripped_essay)

    tokenized_sentences = []
    for raw_sentence in raw_sentences:
        if len(raw_sentence) > 0:
            clean_sentence = re.sub("[^a-zA-Z0-9]"," ", raw_sentence)
            tokens = nltk.word_tokenize(clean_sentence)
            tokenized_sentences.append(tokens)
    return tokenized_sentences


# lemma, noun, adjective, verb, adverb count for a given text

def count_lemmas(text):

    noun_count = 0
    adj_count = 0
```

```
    verb_count = 0
    adv_count = 0
    lemmas = []
    lemmatizer = WordNetLemmatizer()
    tokenized_sentences = sent_tokenize(text)

    for sentence in tokenized_sentences:
        tagged_tokens = nltk.pos_tag(sentence)

        for token_tuple in tagged_tokens:
            pos_tag = token_tuple[1]

            if pos_tag.startswith('N'):
                noun_count += 1
                pos = wordnet.NOUN
                lemmas.append(lemmatizer.lemmatize(token_tuple[0], pos))
            elif pos_tag.startswith('J'):
                adj_count += 1
                pos = wordnet.ADJ
                lemmas.append(lemmatizer.lemmatize(token_tuple[0], pos))
            elif pos_tag.startswith('V'):
                verb_count += 1
                pos = wordnet.VERB
                lemmas.append(lemmatizer.lemmatize(token_tuple[0], pos))
            elif pos_tag.startswith('R'):
                adv_count += 1
                pos = wordnet.ADV
                lemmas.append(lemmatizer.lemmatize(token_tuple[0], pos))
            else:
                pos = wordnet.NOUN
                lemmas.append(lemmatizer.lemmatize(token_tuple[0], pos))

    lemma_count = len(set(lemmas))

    return noun_count, adj_count, verb_count, adv_count, lemma_count

def token_word(text):
    text = "".join([ch.lower() for ch in text if ch not in string.punctuation])
    tokens = nltk.word_tokenize(text)
    return tokens

def misspell_count(text):
    spell = SpellChecker()
    # find those words that may be misspelled
```

```
    misspelled = spell.unknown(token_word(text))
    #print(misspelled)
    return len(misspelled)
```

```
def create_features(texts):
    data =
pd.DataFrame(columns=('Average_Word_Length','Sentence_Count','Word_Count'
,
                          'Character_Count', 'Noun_Count','Adjective_Count',
                          'Verb_Count', 'Adverb_Count', 'Lemma_Count' ,
'Misspell_Count'
                           ))

    data['Average_Word_Length'] = texts.apply(avg_word_len)
    data['Sentence_Count'] = texts.apply(sent_count)
    data['Word_Count'] = texts.apply(word_count)
    data['Character_Count'] = texts.apply(char_count)
    temp=texts.apply(count_lemmas)
    noun_count,adj_count,verb_count,adverb_count,lemma_count = zip(*temp)
    data['Noun_Count'] = noun_count
    data['Adjective_Count'] = adj_count
    data['Verb_Count'] = verb_count
    data['Adverb_Count'] = adverb_count
    data['Lemma_Count'] = lemma_count
    data['Misspell_Count'] = texts.apply(misspell_count)
    return data

def data_prepare(texts,scores):
    #create features from the texts and clean non graded essays
    data = create_features(texts)
    data.describe()
    t1=np.where(np.asanyarray(np.isnan(scores)))
    scores=scores.drop(scores.index[t1])
    data=data.drop(scores.index[t1])

    #scaler = MinMaxScaler()
    #data = scaler.fit_transform(data)

    #train test split
    X_train, X_test, y_train, y_test = train_test_split(data, scores, test_size =
0.3)

    #checking is there any nan cells
```

```
    print(np.any(np.isnan(scores)))
    print(np.all(np.isfinite(scores)))
    return X_train, X_test, y_train, y_test, data

def lin_regression(X_train,y_train,X_test,y_test):
    regr = LinearRegression()
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)

    # The mean squared error
    mse=mean_squared_error(y_test, y_pred)
    mse_per= 100*mse/3
    print("Mean squared error: {}".format(mse))
    print("Mean squared error in percentage: {}".format(mse_per))
    #explained variance score
    print('Variance score: {}'.format(regr.score(X_test, y_test)))

def adaBoost_reg(X_train,y_train,X_test,y_test):
    #regr = RandomForestRegressor(max_depth=2, n_estimators=300)
    #regr = SVR(gamma='scale', C=1, kernel='linear')
    regr = AdaBoostRegressor()
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    # The mean squared error
    mse=mean_squared_error(y_test, y_pred)
    mse_per= 100*mse/3
    print("Mean squared error: {}".format(mse))
    print("Mean squared error in percentage: {}".format(mse_per))
    #explained variance score
    print('Variance score: {}'.format(regr.score(X_test, y_test)))

    feature_importance = regr.feature_importances_

    # make importances relative to max importance
    feature_importance = 100.0 * (feature_importance / feature_importance.max())
    feature_names =
list(('Average_Word_Length','Sentence_Count','Word_Count',
                          'Character_Count', 'Noun_Count','Adjective_Count',
                          'Verb_Count', 'Adverb_Count', 'Lemma_Count'
,'Misspell_Count'
                           ))
    feature_names = np.asarray(feature_names)
    sorted_idx = np.argsort(feature_importance)
    pos = np.arange(sorted_idx.shape[0]) + .5
```

```
    plt.subplot(1, 2, 2)
    plt.barh(pos, feature_importance[sorted_idx], align='center')
    plt.yticks(pos, feature_names[sorted_idx])
    plt.xlabel('Relative Importance')
    plt.title('Variable Importance')
    plt.show()

# convert numerical scores to labels
# (0-1.5) bad (1.5-2.3) average (2.3-3) good
# bad:    '0'
# average '1'
# good    '2'
def convert_scores(scores):
    def mapping(x):
        if x < np.percentile(scores,25):
            return 0
        elif x < np.percentile(scores,75):
            return 1
        else:
            return 2
    return scores.apply(mapping)



# selecting which set to be used 1-8
# in order to combine them all assign set number to 9
def select_set_classification(dataframe,setNumber):
    if setNumber == 9:
        dataframe2 = dataframe[dataframe.essay_set ==1]
        texts = dataframe2['essay']
        scores = dataframe2['domain1_score']
        scores = scores.apply(lambda x: (x*3)/scores.max())
        scores = convert_scores(scores)
        for i in range(1,9):
            dataframe2 = dataframe[dataframe.essay_set == i]
            texts = texts.append(dataframe2['essay'])
            s = dataframe2['domain1_score']
            s = s.apply(lambda x: (x*3)/s.max())
            s = convert_scores(s)
            scores = scores.append(s)
    else:
        dataframe2 = dataframe[dataframe.essay_set ==setNumber]
        texts = dataframe2['essay']
        scores = dataframe2['domain1_score']
        scores = scores.apply(lambda x: (x*3)/scores.max())
```

```
        scores = convert_scores(scores)
    return texts, scores

## Dataset selection

texts, scores = select_set(dataframe,1)
get_hist_avg(scores,5)
X_train, X_test, y_train, y_test, data = data_prepare(texts,scores)

## Regression Analysis

print('Testing for Linear Regression \n')
lin_regression(X_train,y_train,X_test,y_test)
print('Testing for Adaboost Regression \n')
adaBoost_reg(X_train,y_train,X_test,y_test)


## Dataset selection 2


texts, scores = select_set_classification(dataframe,1)
X_train, X_test, y_train, y_test, data = data_prepare(texts,scores)
```

```
## Classification analysis

a=[0.1,1,10,100,500,1000]
for b in a:
    clf = svm.SVC(C=b, gamma=0.00001)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print (b)
    print (clf.score(X_test,y_test))
    print (np.mean(cross_val_score(clf, X_train, y_train, cv=3)))

clf = svm.SVC(C=100, gamma=0.00001)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Cohen's kappa score: {}'.format(cohen_kappa_score(y_test,y_pred)))

print(classification_report(y_test, y_pred))

## Data Analysis
sns.countplot(scores)
```

```
zero = data[(data["Character_Count"] > 0) & (scores == 0)]
one = data[(data["Character_Count"] > 0) & (scores == 1)]
two = data[(data["Character_Count"] > 0) & (scores == 2)]
sns.distplot(zero["Character_Count"], bins=10, color='r')
sns.distplot(one["Character_Count"], bins=10, color='g')
sns.distplot(two["Character_Count"], bins=10, color='b')
plt.title("Score Distribution with respect to Character_Count",fontsize=20)
plt.xlabel("Character_Count",fontsize=15)
plt.ylabel("Distribuition of the scores",fontsize=15)
plt.show()

zero = data[(data["Lemma_Count"] > 0) & (scores == 0)]
one = data[(data["Lemma_Count"] > 0) & (scores == 1)]
two = data[(data["Lemma_Count"] > 0) & (scores == 2)]
sns.distplot(zero["Lemma_Count"], bins=10, color='r')
sns.distplot(one["Lemma_Count"], bins=10, color='g')
sns.distplot(two["Lemma_Count"], bins=10, color='b')
plt.title("Score Distribution with respect to lemma count",fontsize=20)
plt.xlabel("Lemma Count",fontsize=15)
plt.ylabel("Distribuition of the scores",fontsize=15)
plt.show()

zero = data[(data["Sentence_Count"] > 0) & (scores == 0)]
one = data[(data["Sentence_Count"] > 0) & (scores == 1)]
two = data[(data["Sentence_Count"] > 0) & (scores == 2)]
sns.distplot(zero["Sentence_Count"], bins=10, color='r')
sns.distplot(one["Sentence_Count"], bins=10, color='g')
sns.distplot(two["Sentence_Count"], bins=10, color='b')
plt.title("Score Distribution with respect to sentence count",fontsize=20)
plt.xlabel("Sentence Count",fontsize=15)
plt.ylabel("Distribuition of the scores",fontsize=15)
plt.show()

zero = data[(data["Word_Count"] > 0) & (scores == 0)]
one = data[(data["Word_Count"] > 0) & (scores == 1)]
two = data[(data["Word_Count"] > 0) & (scores == 2)]
sns.distplot(zero["Word_Count"], bins=10, color='r')
sns.distplot(one["Word_Count"], bins=10, color='g')
sns.distplot(two["Word_Count"], bins=10, color='b')
plt.title("Score Distribution with respect to word count",fontsize=20)
plt.xlabel("Word_Count",fontsize=15)
plt.ylabel("Distribuition of the scores",fontsize=15)
plt.show()
```

```
zero = data[(data["Average_Word_Length"] > 0) & (scores == 0)]
one = data[(data["Average_Word_Length"] > 0) & (scores == 1)]
two = data[(data["Average_Word_Length"] > 0) & (scores == 2)]
sns.distplot(zero["Average_Word_Length"], bins=10, color='r')
sns.distplot(one["Average_Word_Length"], bins=10, color='g')
sns.distplot(two["Average_Word_Length"], bins=10, color='b')
plt.title("Score Distribution with respect to Average_Word_Length",fontsize=20)
plt.xlabel("Average_Word_Length",fontsize=15)
plt.ylabel("Distribuition of the scores",fontsize=15)
plt.show()
```