Ruiqi Li

Professor Amir Jafari

DATS 6312

12 Dec 2022

NLP Project Individual Report

My group's project focuses on evaluating middle school students' writings. The original dataset is published on Kaggle by Georgia State University. The aim is to use the NLP techniques we have learned to present the EDA and predict the type of discourse elements for each text. The discourse element is the label, and there are 7 discourse elements: lead, position, claim, counterclaim, rebuttal, evidence, and concluding statement. For a general outline of shared work, we handle data preprocessing and EDA, then make predictions with Logistic Regression and Naïve Bayes Model. After that, we use transformers with heads of MLP, CNN, and LSTM to predict the labels, and generate the summary of each discourse text and then use that to predict the labels with the transformers using different heads.

To begin with, here is some context information about the algorithms I have used. Naïve Bayes is a generative classifier while logistic regression is a discriminative classifier. The Naïve Bayes model calculates the probability of classes, which is based on Bayes Rule. The Bayes Rule calculates the posterior probability of an event given the probability of another event that has happened. A and B are events and P(B) must not be undefined. P(A|B) is posterior probability of A given B. P(A) is prior probability of A. P(B|A) is the likelihood probability of B given A. P(B) is the prior probability of B. The formula is shown in Fig 1. For the basic logistic regression, there is the input vector of of [x1,x2…xn] and corresponding weight of [w1,w2…wn]. The

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

Fig 1

output is binary {0,1} (Fig 2 & Fig 3). Then calculate the sum of weighted features and bias and input to function of z that ranges from 0 to 1, which is in sigmoid form (Fig 4). After that, calculate the probability and then classify it (Fig 5). In addition, I have applied the LSA to the

$$x = [x_1, x_1, \ldots, x_n] \qquad Z = (\sum_{i=1}^{n} w_i x_i) + b$$

$$W = [w_1, w_2, \ldots, w_n] \qquad Z = w \cdot x + b$$

$$\hat{y} \in \{0, 1\}$$

Fig 2 & Fig 3

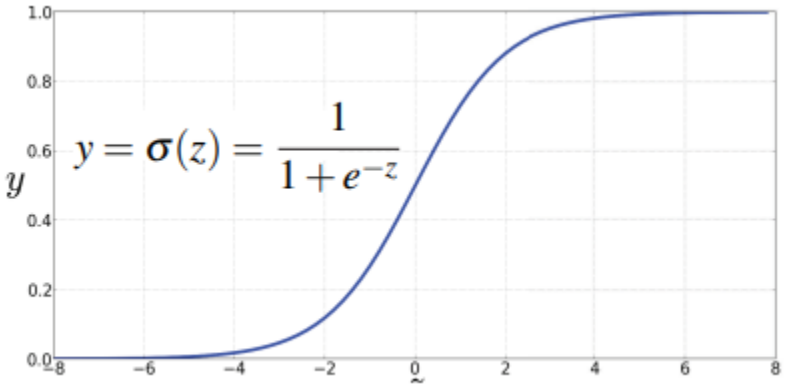

Fig 4

$$P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + exp(-(w \cdot x + b))}$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b) = 1 - \frac{1}{1 + exp(-(w \cdot x + b))}$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b) = \frac{exp(-(w \cdot x + b)}{1 + exp(-(w \cdot x + b))}$$

$$\hat{y} = 1 if P(y = 1|x) > 0.5 otherwise 0$$

Fig 5

data before logistic regression. In the lecture's example of Fig 6, each word can be shown as a score of importance in each row, which is an array. These arrays of numbers are singular values. The data in this form will be transformed to a lower-dimensional matrix table of data which is a truncated SVD (Fig 7). The idea is to improve computational efficiency when training the model.

	algorithms	computers	data	energy	family	food	fun	games	health	home	java	kids	learning	love	machine	money	programming	science	structures
0	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.54	0.00	0.00	0.54	0.00
1	0.00	0.00	0.00	0.00	0.46	0.00	0.37	0.00	0.00	0.46	0.00	0.46	0.00	0.00	0.00	0.46	0.00	0.00	0.00
2	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.54
3	0.00	0.00	0.00	0.42	0.00	0.42	0.34	0.42	0.42	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00	0.00	0.00
4	0.64	0.64	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Fig 6

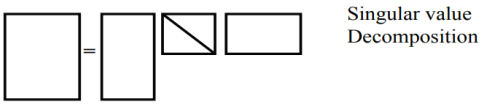

Fig 7

My contribution to work is mostly about the establishment of rule-based models. Other members build transformers to predict the labels, and I use the rule-based models to do the prediction. After getting the data that had been preprocessed by the team, I begin doing some modifications, which are decapitalization, punctuation removal, stemming, tokenization, stopwords removal, lemmatization, and TF-IDF transformation (Fig 8 – Fig 11). The formatted

```
### Lower
body_text_list = df_train['text'].tolist()
body_text_list_t = df_test['text'].tolist()
text_lower = [str(i).lower() for i in body_text_list]
text_lower_t = [str(i).lower() for i in body_text_list_t]

### Punctuations
text_no_punc = [re.sub(r'[^\w\s]', '', i) for i in text_lower]
text_no_punc_t = [re.sub(r'[^\w\s]', '', i) for i in text_lower_t]

### Stem
def stem(phrase):
    return ' '.join([re.findall('^(.*ss|.*?)(s)?$', word)
                     [0][0].strip("'") for word in phrase.lower().split()])
body_text_list_stemmed = [stem(i) for i in text_no_punc]
body_text_list_stemmed_t = [stem(i) for i in text_no_punc_t]
```

Fig 8

```
### Token
token_words_list = []
for i in body_text_list_stemmed:
    w = word_tokenize(i)
    token_words_list.append(w)
token_words_list_t = []
for i in body_text_list_stemmed_t:
    w = word_tokenize(i)
    token_words_list_t.append(w)

### Stopwords
stopword = stopwords.words('english')
list_no_stop = []
list_no_stop_t = []
for i in token_words_list:
    list_no_stop.append([j for j in i if j not in stopword])
for i in token_words_list_t:
    list_no_stop_t.append([j for j in i if j not in stopword])
```

Fig 9

```
### Lemma
lemmatizer = WordNetLemmatizer()
lemmatized_list = []
for i in list_no_stop:
    s = []
    for ii in i:
        s.append(lemmatizer.lemmatize(ii))
    lemmatized_list.append([lemmatizer.lemmatize(j) for j in i])
lemmatized_list_t = []
for i in list_no_stop_t:
    s = []
    for ii in i:
        s.append(lemmatizer.lemmatize(ii))
    lemmatized_list_t.append([lemmatizer.lemmatize(j) for j in i])
```

Fig 10

```
### Clean text list and save
list_text = [' '.join(i) for i in lemmatized_list]
list_text_t = [' '.join(i) for i in lemmatized_list_t]
df_train_new = pd.DataFrame({'text': list_text,
                             'label': df_train['label']})
df_test_new = pd.DataFrame({'text': list_text_t,
                            'label': df_test['label']})

### Vectorizer TFIDF
tfidf_vector = TfidfVectorizer()
tfidf_vector.fit(df_train_new['text'])
X_train_tfidf = tfidf_vector.transform(df_train_new['text'])
X_test_tfidf  = tfidf_vector.transform(df_test_new['text'])
```

Fig 11

training data and test data will be in TF-IDF format, so that they can be input to the models (Fig 12). In addition, I have also trained the logistic model with LSA-processed input. I use different

n_components (output data's dimensionality) and n_iter (number of iterations) when transforming the input.

```
### Logistic no LSA
clf_lo = LogisticRegression().fit(X_train_tfidf, df_train_new['label'])
predict_lo = clf_lo.predict(X_test_tfidf)
report_lo = classification_report(df_test_new['label'], predict_lo,
                                  target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                  output_dict=True)
### Logistic + LSA
lsa = TruncatedSVD(n_components = 500, n_iter = 100)
lsa.fit(X_train_tfidf)
X_train_lsa = lsa.transform(X_train_tfidf)
X_test_lsa = lsa.transform(X_test_tfidf)
clf_l = LogisticRegression().fit(X_train_lsa, df_train_new['label'])
predict_l = clf_l.predict(X_test_lsa)
report_l_lsa = classification_report(df_test_new['label'], predict_l,
                                     target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                     output_dict=True)
### Naive Bayes
clf_n = MultinomialNB().fit(X_train_tfidf, df_train_new['label'])
predict_n = clf_n.predict(X_test_tfidf)
report_nb = classification_report(df_test_new['label'], predict_n,
                                  target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                  output_dict=True)
```

Fig 12

For the experimental results, I have evaluated the models with the classification report. The main metrics are F1 and accuracy. Fig 13 is the report of logistic model and Fig 14 is the report of bayes naïve model. The comparison is straightforward, which shows that logistic regression has better accuracy and credibility than the naïve bayes because the average f1 score is more than 0.5 and accuracy is 0.65. For the LSA-logistic model, I set n_component to 500 and n_iter to 1000 (Fig 14). The evaluation results are hardly improved compared to pure logistic model's.

	precision	recall	f1-score	support
0	0.652985	0.559105	0.602410	313.000000
1	0.604478	0.532895	0.566434	304.000000
2	0.644444	0.530488	0.581940	328.000000
3	0.512953	0.372180	0.431373	266.000000
4	0.710162	0.790488	0.748175	778.000000
5	0.638104	0.840336	0.725389	833.000000
6	0.715054	0.407975	0.519531	326.000000
accuracy	0.653748	0.653748	0.653748	0.653748
macro avg	0.639740	0.576210	0.596464	3148.000000
weighted avg	0.652199	0.653748	0.642334	3148.000000

Fig 13

	precision	recall	f1-score	support
0	0.605166	0.523962	0.561644	313.000000
1	0.610169	0.473684	0.533333	304.000000
2	0.631579	0.512195	0.565657	328.000000
3	0.479452	0.394737	0.432990	266.000000
4	0.679267	0.762211	0.718353	778.000000
5	0.626606	0.819928	0.710348	833.000000
6	0.663212	0.392638	0.493256	326.000000
accuracy	0.630559	0.630559	0.630559	0.630559
macro avg	0.613636	0.554194	0.573654	3148.000000
weighted avg	0.627776	0.630559	0.619453	3148.000000

Fig 14

In short, both naïve bayes and logistic regression are efficient rule-based models even though logistic regression tends to have better evaluation scores (F1 and accuracy) than naïve bayes. For the naïve bayes, it is important that the feature probabilities are all independent given the event happened. LSA does not quite enhance the logistic model. The input data's dimensionality may not need a necessary reduction. Also, the balanced dataset is a fundamental setting for such model training to prevent biases. If improvements can be made, I would like to refine the text handling for the model. Even though I have used popular methods like tokenization, stemming, lemmatization, etc. I can still see some words that are not in the best shapes; therefore, trying other NLP packages could be useful. For example, I could try other tokenization methods like RegexpTokenizer and TreebankWordTokenizer. PorterStemmer could also be helpful during the stemming.

For my percentage of the outside code, it should be around 2.17%. My calculation is (2-0) / (2+90). This is the closest I can calculate since there is empty space and print statements. Lecture codes, import lines, empty lines, and print lines are not included, but the general percentage is close to this number.

References:

1. “Feedback Prize - Evaluating Student Writing.” *Kaggle*, https://www.kaggle.com/competitions/feedback-prize-2021/data.

2. Ioana. “Latent Semantic Analysis: Intuition, Math, Implementation.” *Medium*, Towards Data Science, 22 Nov. 2020, https://towardsdatascience.com/latent-semantic-analysis-intuition-math-implementation-a194aff870f8.

3. “Sklearn.decomposition.truncatedsvd.” *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html.

4. Natural Language Processing Lecture 4 Slides: Regular Expression & Bag of Words

5. Natural Language Processing Lecture 5 Slides: Naive Bayes & Logistc Regression