GUID: 2413917
Name: Ching Han, Cheng

Text-as-Data Coursework

Part A

Q1

In the beginning, the dataset has been separated into a training dataset and testing dataset, the data distribution has been displayed. Since there are some features needed which are combined in the 'posts' variable, the process of collecting new features such as 'author ID', 'body' into datasets are approached.

(i) Before building the model, due to the construction of the dataset is in texts, the NLTK (Natural Language Toolkit) process is needed for transforming the data structure into quantities to be analyzed. There are mainly two processes for data preprocessing, namely tokenization and normalization. In the tokenization process, each word in texts is split by punctuations such as _-.?!() or whitespace characters such as " "TAB \n \r. In the normalization process, the uppercase characters will be transformed into lowercase type and the tokens will be stemmed which only leave the core meaning part of the word. For the three features applied in this step namely title, body, author ID, both tokenizing and normalizing into lower case and stems are applied to the previous two, but the author ID is applied different process without stemming since each author name should be unique, if it is stemmed then the new word may represent to different author. For instance, if there are two authors, Amy and Ami, after the stemming process they can be turned into ami and ami.

(ii) After tokenizing and normalization process, there are two different vectorizers processes provided which are one- hot encoding and TF- IDF turning them into vectors. Five classifiers namely Logistic regression, SVC, Bernoulli NB classifiers and Dummy Classifier with two types of strategy, 'stratified' and 'most_frequent' are then approached. The performance of each classifier is shown in the table below:

Classifier	Logistic Regression		SVC classifier		Bernoulli NB classifier		Dummy Classifier (strategy='stratified')		Dummy Classifier (strategy='most_frequent')	
Overall performance	One- hot encoding	TF- IDF	One- hot encoding	TF- IDF	One- hot encoding	TF- IDF	One- hot encoding	TF- IDF	One- hot encoding	TF- IDF
F1	**0.645**	0.514	0.05	0.019	0.116	0.116	0.049	0.052	0.019	0.019
Accuracy	0.707	0.619	0.277	0.23	0.356	0.356	0.103	0.102	0.23	0.23
Recall	0.743	0.735	0.055	0.012	0.465	0.465	0.049	0.053	0.012	0.012
Precision	0.612	0.459	0.074	0.05	0.119	0.119	0.049	0.052	0.05	0.05

Table 1. Macro classifier performances

The classifier with best F1 is Logistic Regression with one- hot encoding vectorizer, so the confusion matrix of it is provided with labels in the following:

L1	askreddit	L11	pcmasterrace
L2	atheism	L12	personalfinance
L3	buildapc	L13	reddit.com
L4	electronic_cigarette	L14	relationships
L5	explainlikeimfive	L15	starcraft
L6	explainlikeimfive2	L16	summonerschool
L7	hearthstone	L17	techsupport
L8	jailbreak	L18	tipofmytongue
L9	leagueoflegends	L19	trees
L10	movies	L20	whowouldwin

Table 2. Labels in confusion matrix

	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
L1	74	0	0	0	1	4	0	0	3	0	0	1	0	0	0	0	0	0	1	0
L2	4	7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
L3	3	0	30	0	0	0	0	0	1	0	1	0	0	0	0	0	2	0	0	0
L4	1	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
L5	2	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L6	1	0	0	0	0	8	0	0	3	0	4	0	0	0	0	0	0	1	0	0
L7	3	0	0	0	0	2	9	0	0	0	0	0	0	0	0	0	0	1	0	0
L8	0	0	0	0	0	0	0	9	1	0	0	0	0	0	0	0	0	1	0	0
L9	5	0	1	0	0	1	0	0	40	0	0	0	0	0	0	1	0	0	0	0
L10	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0
L11	2	0	5	0	0	3	0	0	1	0	10	0	0	0	0	0	2	0	0	0
L12	1	0	0	0	1	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0
L13	4	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
L14	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
L15	2	0	0	0	0	2	0	0	1	0	0	0	1	0	3	1	0	0	0	0
L16	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	2	0	0	0	0
L17	1	0	2	1	0	1	0	0	0	0	1	0	0	0	0	0	7	0	0	0
L18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0
L19	7	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	11	0
L20	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Table 3. Confusion matrix of Logistic regression classifier base on One- hot encoding data vectorizer

(iii) The plot below shows the F1- score of Logistic regression classifier base on One- hot encoding vectorizer:

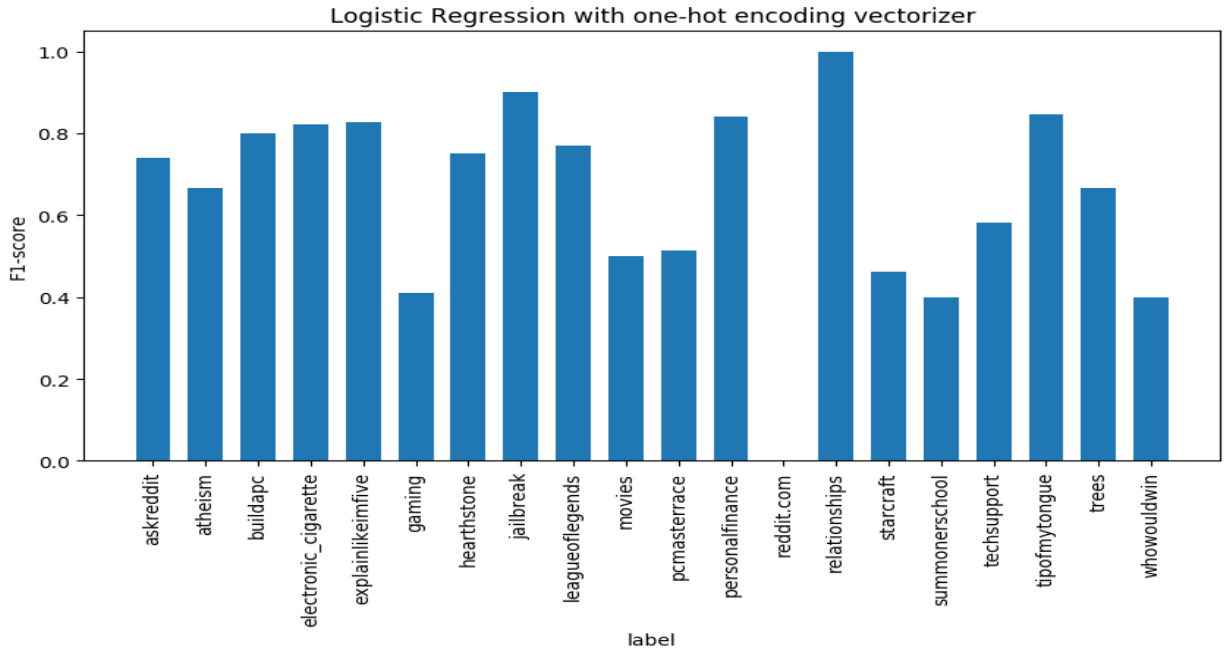

Figure 1. F1- score of Logistic regression classifier base on One- hot encoding vectorize in top 20 subreddit

(iv) The best performance model combines the process of One- hot encoding and Logistic regression classifier. The reason might be the Logistic regression classifier is more suitable to be applied for processing binary data, and one- hot encoding is exactly the way to transform data into binary type. So the combination can build a more effective model.

Q2

(i) Parameters tuning

The aim of this part is to tune the model of Logistic Regression with TF-IDF vectorization, there are two types of parameters need to be tuned, which are parameters of Logistic Regression and TF-IDF vectorizer parameters.

The parameters have been tuned in Logistic Regression classifier are 'C', 'solver' and 'multi_class' with possible values provided below:

parameters:

{'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'], 'multi_class': ['ovr', 'multinomial']}

The parameters have been tuned in TF-IDF vectorizer are 'max_features', 'ngram_range' and sublinear_tf with possible values provided below:

parameters:

{'vect__max_features': (None, 5000, 10000, 50000), 'vect__ngram_range': ((1, 1), (1, 2), (1, 3)), 'vect__sublinear_tf': (True, False)}

Since the goal is to improve the performance on F1 score so the parameters of our grid search is set as

grid_search = GridSearchCV(pipeline, parameters, cv=5, n_jobs=-1, verbose=1,scoring='f1_macro')

The tuning result of Logistic Regression is shown in the following:

Best score: 0.656 Best parameters set: C: 10000 multi_class: 'ovr' solver: 'saga'

The tuning result of TF-IDF is shown in the following table:

	Body	Title	Author Id
Best score	0.646	0.497	0.378
Best parameters set			
	Body	Title	Author Id

vect__max_features	10000	10000	50000
vect__ngram_range	(1,1)	(1,1)	(1,1)
vect__sublinear_tf	TRUE	FALSE	TRUE

Table 4. Best parameters after vectorizer tuning process

After tuning processes, the new parameters are provided into the new training model and the performance is improved which the F1 is increased from 0.514 to 0.742.

The performance of TF-IDF Logistic regression classier after tuning process: F1=0.742, Acc=0.784, P=0.716, R=0.806

(ii) Error analysis

For all three features error analysis, the performance score is 1, 0.78 on training data and testing data. Although the testing data get a lower fitting score, the difference is not too significant to be concerned.

Nevertheless, the error analysis on author Id and title present a significant difference in the training and testing score which are 0.997, 0.41 and 0.999, 0.55 in respectively which can be considered as overfitting situation. There is a pattern shows a higher possibility to predict a post as ‘askreddit’ subreddit label.

In author Id perspective, the reason for this situation might be the same author tends to post contents or leave a comment on different subreddits and if ‘askreddit’ is a more popular channel to activate then most authors would have posted on ‘askreddit’ before.

In title perspective, there are some titles with ‘Ask Reddit’in sentence, but it is a post on ‘reddit.com’ rather than ‘askreddit’. Moreover, it is easy to settle any title with ‘How’, ’What’… as ‘askreddit’ as well, but some posts obviously belong to another channel. For example, ‘What are some movies that are sure to make you cry?’ should be classified into movies rather than ‘askraddit’.

At last, with single feature ‘body’ gets a much higher accuracy score on testing data compared to the other two features, with training score 1 and testing score 0.73. However, the lengths of the body are various in posts, and some contents are repeated for several times in a text. These may lead to an unfair situation on counting the numbers of specific words shown in a thread.

(iii) Adding new features:

Based on the previous opinions in error analysis, there are two new features added in this process, length of posts/ threads and word embedding feature. The length of posts/ threads is provided to count the numbers of post belong to the same thread and then divided by the total numbers of thread in the dataset. The word embedding feature is mainly applied to estimate the similarity of meaning of words, the 'Word2vec' and 'Glove' model are provided to generate the dense vector as a new feature. The performances on adding features are shown in the following table:

Features list/ Performance	F1	Accuracy	Precision	Recall
Original	0.716	0.778	0.747	0.739
length of posts/ threads	0.729	0.778	0.747	0.739
word embedding	0.749	0.795	0.724	0.824
length of posts/ threads + word embedding	0.754	0.797	0.762	0.771

Table 5. Performance on TF-IDF Logistic regression model with added features

The table shows the performances of models by adding any of the new features or both are improved. The length of posts/thread can help to solve the problem of unfair comparison and the word embedding may ease the problem about easily locate unsure post into ‘askreddit’ subreddit. For further improvement, I think the way to remove the duplicate information can be provided since the contents of 'body' are often quite long but might lack information.

Part B

In part B, the three features and parameters generated in Q2 are provided, however, the performance of the model is not better than the original one so the original one is applied for the following tuning process.

Q3:

(i) Model tuning

Before tuning the model, I have tried to add or remove some other features such as 'url', 'id' and 'majority_link' etc. to check if the F1 score would improve. As a result, there are four features 'title', 'author', 'body' and 'majority_link' are added in the tuning process.

The result of tuning process is:

Regression parameters {C: 10000, solver: 'lbfgs', multi_class: 'ovr'}

TF-IDF parameters for body { vect__max_features: 50000, vect__ngram_range: (1, 2), vect__sublinear_tf: True}

The performance of model before tuning process: **F1=0.223, Acc=0.381, R=0.255, P=0.210**

The performance of model after tuning process: **F1=0.315, Acc=0.479, R=0.359, P=0.294**

C1	agreement	C6	elaboration
C2	announcement	C7	humor
C3	answer	C8	negativereaction
C4	appreciation	C9	other
C5	disagreement	C10	question

Table 6. Class labels of confusion matrix

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
C1	231	0	433	53	16	161	6	9	6	36
C2	0	130	0	0	0	0	1	0	15	219
C3	185	0	5466	182	91	1517	65	29	24	381
C4	47	0	346	1065	10	167	20	9	22	34
C5	30	0	376	8	38	137	8	6	4	34
C6	148	0	2187	123	51	873	22	17	30	179
C7	15	4	206	34	3	108	27	9	9	42
C8	8	0	125	29	3	80	3	26	6	27
C9	8	17	100	20	0	63	11	4	49	104
C10	58	123	1070	82	32	401	27	18	32	1582

Table 7. Confusion matrix

(ii) Error analysis

In error analysis, to consider the confusion matrix in table 7 shown in the previous answer, the model is more likely to predict a post as an 'answer' disclosure since most contents below the question can be identified as an answer.

There are two patterns of normal error:

At first, it is more likely to have a wrong prediction when a comment is left by the same author who asked the question since the author might just want to reply those who have solved the problem for him/her.

For example:

The real disclosure: appreciation, The wrong prediction: answer, Body: Great info! It shows that stopping on a …

The other one is when the depth of post equal to 0, it is normally an 'announcement' and while it is getting deeper around more than 5, it would become an elaboration.

374 question elaboration post_depth 7

377 elaboration answer post_depth 10

Q4:

1. **Content + Punctuation: Words unigrams, bigrams, and trigrams - Multi-label & Multi-class type**

   This feature is implemented by setting the parameter ngram_range=(1, 3) in TF-IDF process which means in indexing process it would consider to take one, two and three units of tokenizing.

   For instance, while there is a sentence: 'Hard Disk Failing. Please Help a Brother out :'('

   The result of 1-gram indexing would be hard | disk | failing | please |…

   The result of 2-gram indexing would be hard disk | disk failing | failing please | please help | …

   The result of 3-gram indexing would be hard disk failing | disk failing please | failing please help | …

   The reason for applying this feature is because we do not often get the core meaning of the sentence by a single word, most key words are built with two or three words, take hard disk as an example, if the hard and disk are separated then it can be different to get the meaning of words.

2. **Structure: The depth of the comment, raw or normalized by the length of the thread - Single-label & Multi-class type**

   In this feature, the depth of the comment is normalized by the length of the thread through the calculation way shown below:

$$\textbf{depth of each comment after normalization} = \frac{\textbf{post depth of each post}}{\textbf{length of the thread}}$$

   For example, while the depths of the comment are 2, 3, 3, 1 … in training dataset then they would be turned into 2/79267, 3/79267, 3/79267, 1/79267 …

   On the other hand, while the depths of the comment are 1,2,3,2 … in testing dataset then they would be turned into 1/19812, 2/19812, 3/19812, 2/19812 …

   The reason for applying this feature is because if the depth of the comment are deeper means people willing to discuss the issue more seriously, which also means the post with more valuable information or it may get more controversial reactions from other users. And since the depth of posts base on each thread, the length of the training and testing thread should be provided to normalize them into scales.

3. **Author: A binary feature for whether the current author is also the author of the initial post - Single-label & Binary-class type**

   This feature considers if the current post user is the same as the initial author by applying the initial author ID to do the comparison with each current post author ID.

$$\text{And if } \begin{cases} \text{the current author ID} = \text{the initial author ID}, & 1 \\ \text{the current author ID} \neq \text{the initial author ID}, & 0 \end{cases}$$

   For example, if the initial author ID for a post is 'greymeta', then in the same post, if there is a current author ID also called 'greymeta', the value 1 will be given in the matrix otherwise 0 will be located.

   The reason for adding this feature is since the initial author can be identified as a manager of the post, so they tend to reply their own post more often, and the information they generate can be considered as summarize or new direction of the discussion so there could be more information inside.

4. **Thread features: The total number of comments in the discussion - Multi-label & Single-class type**

   This feature is to calculate the numbers of comment are left in a post, the method to implement this is to generate a

dictionary with post titles and while there are same titles shown in the dataset, the dictionary would group them and generate the counts. For example, after feature generating it would become { **'Help me decide my new PvP main' :** 7, '**Anime so bad it's good?**' : 38}

The reason for putting this feature is because while there are more replies to the post, the post can be considered as more important since the issue can get more attention to be discussed.

5. **Community The subreddit the post came from - Multi-label & Multi-class type**

   This feature is to locate the subreddit of the post came from, since each subreddit discuss about different topic so this may help to recognize some specific channel with similar disclosure type.

6. **Word2Vec / NLP : Average/ Max of pre-trained Glove embeddings for the post - Multi-label & Multi-class type**

   This feature has been provided and discussed in Q2, it is to identify the similarity between meanings of words and with this feature the problem about typo and ambiguous words can be identified much easier.

Table 8. performances of TF-IDF Logistic regression classifier by adding features

Features list/ Performance	F1	Accuracy	Precision	Recall
Original	**0.315**	**0.479**	**0.359**	**0.294**
Feature 1	**0.239**	**0.52**	**0.459**	**0.226**
Feature 2	**0.314**	**0.479**	**0.357**	**0.293**
Feature 3	**0.315**	**0.481**	**0.356**	**0.295**
Feature 4	**0.299**	**0.474**	**0.433**	**0.275**
Feature 5	**0.294**	**0.524**	**0.434**	**0.267**
Feature 6	**0.307**	**0.478**	**0.414**	**0.278**
ALL 6 features	**0.21**	**0.506**	**0.368**	**0.205**

The plot is generated by the final six features model, and it shows the top 20 features importance to each discourse. The 'answer' discourse gets the highest top 20 importance which means it has more influence on the performance of the model. In contrast, 'negativereaction' gets the lowest importance and has less influence on the model.

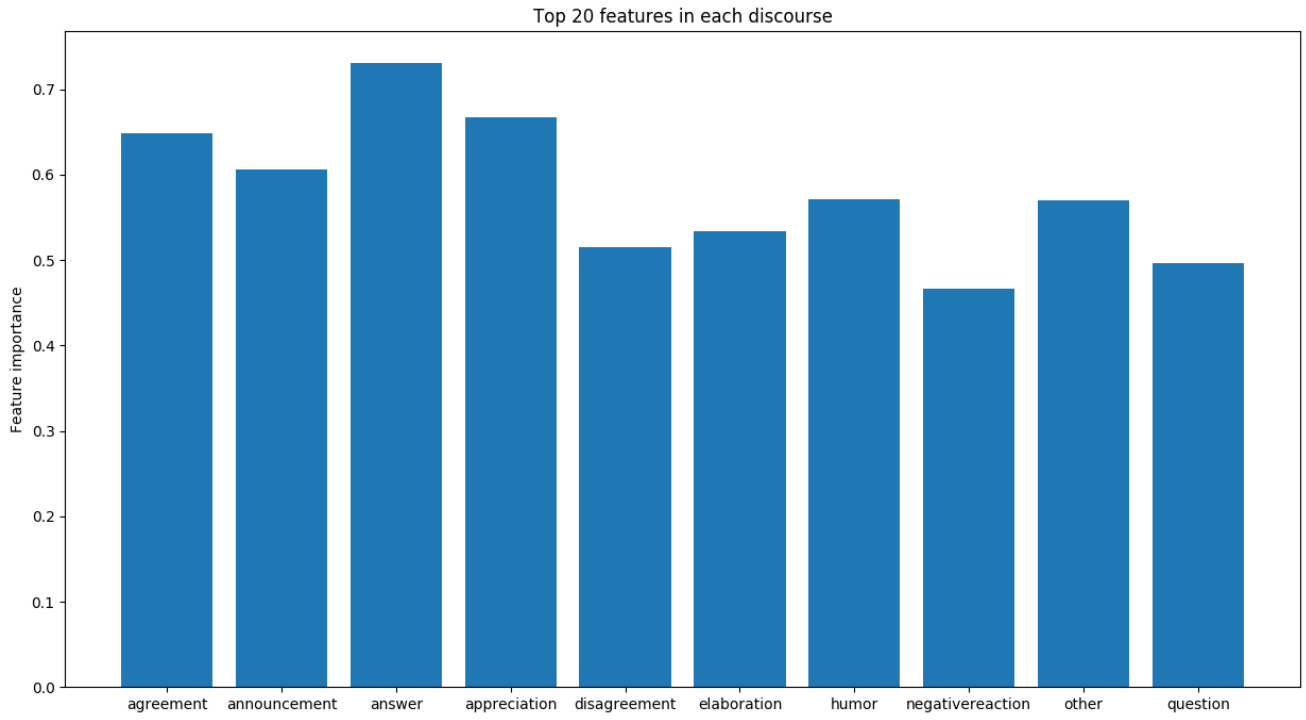

Figure 2. Top 20 feature importance in 10 disclosure types