Word Complexity Estimation

Bianca-Oana Guiță
411

I. Introduction

Word complexity estimation is a task supposed to give each target word a probability based on how complex it is. We are given a training set containing 14,002 labeled examples and a test set that consists of another 1,764 examples. The complexity of one target word is calculated as it follows:

$$\text{complexity} = \frac{\text{the number of annotators who marked the word as difficult}}{\text{the total number of annotators}}$$

II. The Dataset

The dataset that we were provided consists in three txt files:

- train_full.txt: contains the training data and their associated labels
- test.txt: contains the test data
- sample_submission.txt: gives the correct format of a submission

The training data is represented by sentences (one per line) having a complex word in a certain context. The columns of the training table show, in this order, the Id of the training sample, the actual sentence, the start offset of the target word, the end offset of the target word, the actual target word, the number of native and non-native annotators that have seen the sentence, the number of native and non-native annotators that find the target word complex, and the probability.

	ID	Sentence	Start_Offset	End_Offset	Target Word	NatAnnot	NonNatAnnot	NrNat	NrNonNat	Prob
0	1	The barren islands, reefs and coral outcrops a...	4	10	barren	10	10	6	2	0.40
1	2	The barren islands, reefs and coral outcrops a...	4	18	barren islands	10	10	0	1	0.05
2	3	The barren islands, reefs and coral outcrops a...	20	25	reefs	10	10	1	2	0.15
3	4	The barren islands, reefs and coral outcrops a...	11	18	islands	10	10	0	0	0.00
4	5	The barren islands, reefs and coral outcrops a...	30	35	coral	10	10	0	0	0.00
...	...	...	...	...	...	...	...	...	...	...
13997	13998	Stephen Biddle, a defense analyst at the Counc...	185	201	security partner	10	10	2	0	0.10
13998	13999	Stephen Biddle, a defense analyst at the Counc...	185	193	security	10	10	0	0	0.00
13999	14000	Stephen Biddle, a defense analyst at the Counc...	194	201	partner	10	10	0	0	0.00
14000	14001	Stephen Biddle, a defense analyst at the Counc...	217	223	troops	10	10	0	0	0.00
14001	14002	Stephen Biddle, a defense analyst at the Counc...	234	238	home	10	10	0	0	0.00

14002 rows × 10 columns

Test data format is similar to train data's one except that there are present only the first seven columns. The number of native and non-native annotators that found the target word difficult and the probability are not given.

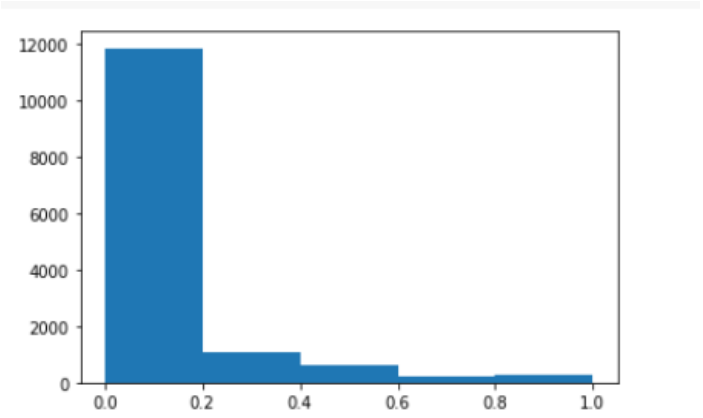
	ID	Sentence	Start_Offset	End_Offset	Target Word	NatAnnot	NonNatAnnot
0	14003	Syrian troops shelled a rebel-held town on Mon...	7	13	troops	10	10
1	14004	Syrian troops shelled a rebel-held town on Mon...	0	6	Syrian	10	10
2	14005	Syrian troops shelled a rebel-held town on Mon...	14	21	shelled	10	10
3	14006	Syrian troops shelled a rebel-held town on Mon...	24	34	rebel-held	10	10
4	14007	Syrian troops shelled a rebel-held town on Mon...	51	59	sparking	10	10
...	...	...	...	...	...	...	...
1759	15762	The state will put less than 15 billion euros ...	128	132	past	10	10
1760	15763	The state will put less than 15 billion euros ...	139	144	years	10	10
1761	15764	The state will put less than 15 billion euros ...	157	166	estimated	10	10
1762	15765	The state will put less than 15 billion euros ...	146	148	de	10	10
1763	15766	The state will put less than 15 billion euros ...	149	156	Guindos	10	10

1764 rows × 7 columns

III. Preprocessing data, features and visualizations

First of all, I was curious to discover what type of words had the highest probability regarding their complexity. I noticed that there were pretty long words having the number of consons greater than the vowels so I thought the length and number cons/vowels would be two suitable features to use for my model.

By plotting the distribution of words based on their complex probability, it came to my attention that most of the words are being understood by both native and non-native annotators. There is a percent though that consider the target word as being difficult (probabilities between 0.6 and 1.0).



Along with the number of vowels, cons and length of the word, I also chose as features the number of synonyms that share a common meaning with the target word, the length of the definition for the target word, whether it starts with a capital letter or not, and the type of it (noun/verb/adj etc). Checking the correlation between all the data, we can remark that there actually is a connection between them, which proves that my extracted features would be helpful.

	ID	Start_Offset	End_Offset	NatAnnot	NonNatAnnot	Prob	Vowels	Cons	LenWord	NumSyns	LenDef	Capital	Type
ID	1.000000	0.031585	0.031432	NaN	NaN	-0.005773	-0.004872	0.001727	-0.000652	-0.005618	-0.009577	0.011450	0.000336
Start_Offset	0.031585	1.000000	0.997099	NaN	NaN	0.002798	0.006531	0.003119	0.004519	0.001998	0.010313	0.079042	-0.012278
End_Offset	0.031432	0.997099	1.000000	NaN	NaN	0.014813	0.077011	0.077551	0.080627	-0.025454	-0.020904	0.084753	-0.038516
NatAnnot	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NonNatAnnot	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Prob	-0.005773	0.002798	0.014813	NaN	NaN	1.000000	0.184798	0.133128	0.157951	-0.167423	-0.086333	0.222707	0.134775
Vowels	-0.004872	0.006531	0.077011	NaN	NaN	0.184798	1.000000	0.826901	0.926152	-0.344009	-0.352975	0.103280	-0.286131
Cons	0.001727	0.003119	0.077551	NaN	NaN	0.133128	0.826901	1.000000	0.977926	-0.346995	-0.415237	0.059669	-0.356190
LenWord	-0.000652	0.004519	0.080627	NaN	NaN	0.157951	0.926152	0.977926	1.000000	-0.360544	-0.409643	0.078394	-0.345205
NumSyns	-0.005618	0.001998	-0.025454	NaN	NaN	-0.167423	-0.344009	-0.346995	-0.360544	1.000000	0.176591	0.238475	0.181216
LenDef	-0.009577	0.010313	-0.020904	NaN	NaN	-0.086333	-0.352975	-0.415237	-0.409643	0.176591	1.000000	-0.093826	0.268054
Capital	0.011450	0.079042	0.084753	NaN	NaN	0.222707	0.103280	0.059669	0.078394	0.238475	-0.093826	1.000000	0.181813
Type	0.000336	-0.012278	-0.038516	NaN	NaN	0.134775	-0.286131	-0.356190	-0.345205	0.181216	0.268054	0.181813	1.000000

As for the sentences in training data, I preprocessed the text by excluding the punctuation and tokenizing the words. Also treated some exceptions and cleaned the text by separating numbers from words and eliminating special characters so it can be ready for the models.

IV. Models

My first approach after splitting the training data with train_test_split, was to try some basic models such as LinearRegression, SGD Regressor with Standard Scaler, MLP Regressor, and Random Forest Regressor. The results were not that appealing, even after parameter-tuning. For these, as features I only used the length of the word and number of vowels and cons.

```
15] from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_absolute_error
    reg = LinearRegression().fit(X_train, y_train)
    mean_absolute_error(reg.predict(X_test),y_test)
```

```
0.11272926679867074
```

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
regr = make_pipeline(StandardScaler(),
                     SGDRegressor(loss= 'huber',penalty = 'l2',max_iter=1000,
                     tol=1e-3, random_state=1,epsilon=0.01,
                     learning_rate='invscaling',
                     average=15))
regr.fit(X_train, y_train)
mean_absolute_error(regr.predict(X_test),y_test)
```

```
0.08643405295676906
```

```
from sklearn.neural_network import MLPRegressor
regr = MLPRegressor(random_state=1,activation='logistic', max_iter=500,
                    solver='adam',batch_size=10, learning_rate='adaptive',
                    learning_rate_init=0.001,shuffle=False,tol=1e-5,
                    epsilon=1e-9).fit(X_train, y_train)
mean_absolute_error(regr.predict(X_test),y_test)
```

```
0.09306952444068817
```

```
[19] params = {'bootstrap': [True, False],
      'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
      'max_features': ['auto', 'sqrt'],
      'min_samples_leaf': [1, 2, 4],
      'min_samples_split': [2, 5, 10],
      'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
regr = RandomForestRegressor()
grid_search = GridSearchCV(estimator = regr, param_grid = params,
                           cv = 3, n_jobs = -1, verbose = 2, scoring = "neg_mean_absolute_error")
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 3960 candidates, totalling 11880 fits
```

The next approach was to create and train my own Word2Vec model only using the sentences I was provided in my dataset. I tried a various number of possible parameters and in the end, the best results were provided by having a vector size of 200, two windows, hs value of 1, cbow_mean value of 0 and negative value of 5.

```
my_model=gensim.models.Word2Vec(data, min_count = 1, size = 200, window = 2, sg =1,seed=1,hs=1,cbow_mean=0, negative=5)
```

Firstly, just to see the difference, I also trained my model on MLP Regressor and noticed an improvement. Here, I used only word2vec word embeddings.

```
from sklearn.neural_network import MLPRegressor
regr = MLPRegressor(random_state=1,activation='logistic',
                    max_iter=500,solver='adam',batch_size=48,
                    learning_rate='adaptive',learning_rate_init=0.001,
                    shuffle=False,tol=1e-5,epsilon=1e-9).fit(X_train, y_train)
mean_absolute_error(regr.predict(X_test),y_test)
```

```
0.06794055644214425
```

For Random Forest Regressor I added as features the number of synonyms that share a common meaning with the target word, the length of the definition for the target word, whether it starts with a capital letter or not, the type of it and word2vec embeddings.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
regr = RandomForestRegressor(bootstrap=False, max_features= 'log2',
                             n_estimators=1600).fit(X_train, y_train)
mean_absolute_error(regr.predict(X_test),y_test)
```

```
0.05950165819065008
```

For testing purposes, I used the cross validation score to ensure that with this combination I achieve good validation scores and generalization.

Cross Validation Scores					
Model	Validation 1	Validation 2	Validation 3	Validation 4	Validation 5
Linear Regression	-0,111366	-0,109851	-0,099416	-0,100987	-0,104030
Random Forest	-0,060020	-0,055327	-0,059742	-0,059427	-0,061678

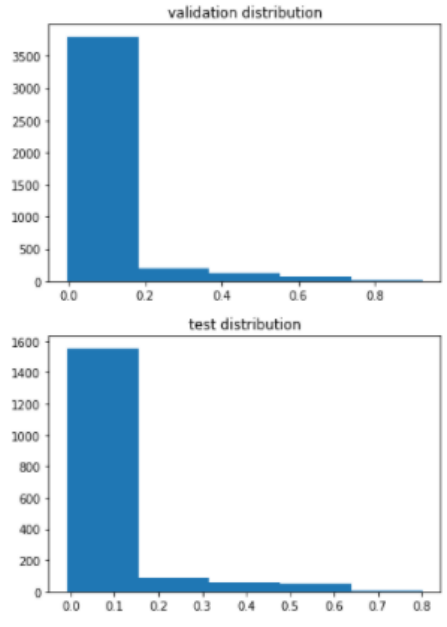
Because I wanted even better results, I tried using my word2vec model with neural networks. I utilized four layers of Linear, ReLU as an activation function and two layers of Dropout to prevent overfitting.

For the loss function, my best choice was L1 Loss, Adam as the optimizer with a learning rate of 1e-4. I also went for a learning rate scheduler that drops the learning rate with 0.6 at epoch 4 and 11.

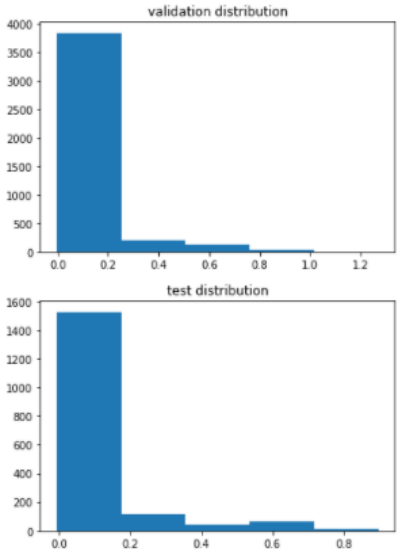
```
[ ]  # Initialize the MLP
     mlp = MLP().cuda()

     # Define the loss function and optimizer
     loss_function = nn.L1Loss()
     optimizer = torch.optim.Adam(mlp.parameters(), lr=1e-4)
     lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer,[4,11],0.6,verbose=True)

     Adjusting learning rate of group 0 to 1.0000e-04.
```

To see my model “in action”, the way it is learning, I plotted the validation and test distribution per each epoch (15 epochs in total). It came out as my best results so far.

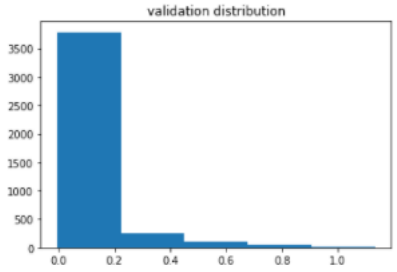
```
Adjusting learning rate of group 0 to 6.0000e-05.
Loss-ul la finalul epocii 10 are valoarea 0.3622859716415405
valid err:  0.05910135825686822
```

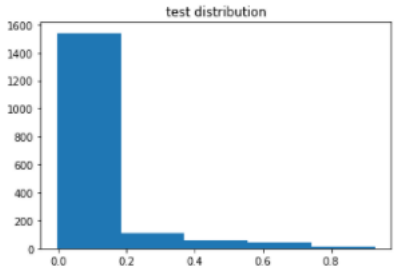



```
Adjusting learning rate of group 0 to 3.6000e-05.
Loss-ul la finalul epocii 11 are valoarea 3.11913640871353e-06
valid err:  0.05873732560981225
```

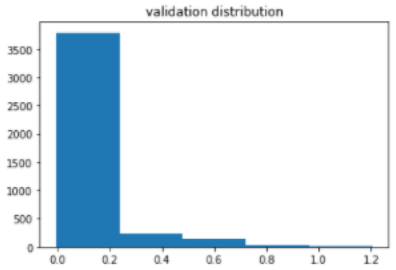


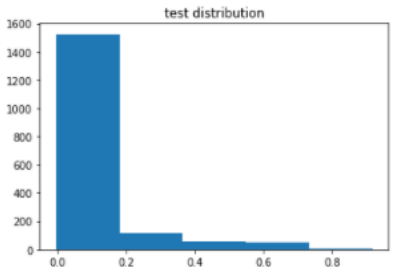

```
Adjusting learning rate of group 0 to 3.6000e-05.
Loss-ul la finalul epocii 12 are valoarea 3.25437358696945e-05
valid err:  0.05840184609149201 6
```






```
Adjusting learning rate of group 0 to 3.6000e-05.
Loss-ul la finalul epocii 13 are valoarea 5.866142487320758e-07
valid err:  0.0577096815430599
```






```
Adjusting learning rate of group 0 to 3.6000e-05.
Loss-ul la finalul epocii 14 are valoarea 0.03953289985656738
valid err:  0.05724886349522587
```

V. Conclusion and further work

In conclusion, I have noticed that the results provided by the neural network layers are better on my word2vec model, and the distribution between test and validation data is looking pretty similar. Regarding the fact that I have only trained my model solely on the dataset given in the competition, for further work I would try training it on larger datasets such as WikiDump or Web Corpuses.