Bad Labelling

January 3, 2022

Bad Labelling Experiment

In this notebook, we will be conducting an experiment which aims at capturing the keywords which were wrongly labelled in our input dataset. This process is crucial as it helps us in gaining more useful insights of the data points that we will be using to train our classifier but also find any similar trend that may occur in our input keywords.

The github repository of the author was used to follow this experiment which can be found on the link here.

This project/notebook consists of several Tasks.

- **Task 1**: Installing all dependencies for our DoubtLab library.
- **Task 2**: Importing the required libraries in the environment.
- **Task 3**: Importing the dataset which was manually labelled or the final results of the classification task.
- **Task 4**: Data Analysis and Pre-processing of keywords by one-hot-encoding.
- **Task 5**: Assign the label to investigate and pass it through a Logistic Regression and BytePair Embeddings pipeline.
- **Task 6**: Assign the doubts from the github page of the author.
- **Task 7**: Investigate each reasons individually and extract keywords which do not match with their assigned label names.
- **Task 8**: Evalute the keywords and store the subset dataset for a label in a pickel for future use.

0.0.1 Task 1: Installing all the Dependencies for our DoubtLab Library

Firstly, installing all of the dependent libraries for using the DoubtLab. You will need to run the following cells when using the notebook for the first time to have these libraries in this notebook environment.

[ ]:
```
#pip install doubtlab
```

[ ]:
```
#pip install --upgrade pip
```

[ ]:
```
#!pip install whatlies
```

0.0.2 Task 2: Importing the required libraries in the environment.

[37]:
```
# Data Analysia and Pre-processing
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Training the keywords
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer

#Assigning the doubt reasons
from doubtlab.ensemble import DoubtEnsemble
from doubtlab.reason import ProbaReason, DisagreeReason, ShortConfidenceReason

# Visualizing data values
import plotly as py
import plotly.graph_objs as go
import ipywidgets as widgets
from scipy import special
import plotly.express as px

py.offline.init_notebook_mode(connected = True)
```

0.0.3 Task 3: Importing the dataset

In this notebook, the dataset that we will be using is of df.xlsx a name given to replicate the dataset used.

[ ]:
```
df = pd.read_excel("df.xlsx", sheet_name = 'df1')
df = df.rename({'Segment 1': 'Topic'}, axis = 1)
df = df[['Keyword','Topic']]
df.head(20)
```

0.0.4 Task 4: Data Analysis and Pre-processing of keywords by one-hot-encoding.

[ ]:
```
df.info()
```

[ ]:
```
fig = px.histogram(df,x='Topic')
fig.show()
```

From the above histogram generated, we can see that the most populated labels are the `a` and `b`. While the labels with the least keywords are the `topic name` and `topic name 1`. For this we will be investigating the `topic name` labels as these sets of labels seem quite close enough and may have same type of keywords present in them.

[ ]:
```
rated_dummies = pd.get_dummies(df['Topic'])
df = pd.concat([df, rated_dummies], axis=1)
df.pop('Topic')
df
```

0.0.5 Task 5: Assign the label to investigate and pass it through a Logistic Regression and BytePair Embeddings pipeline.

Firstly, we will be passing all of the keywords from our input dataset and only extracting the ones which are labelled as `topic name` and marking them to 1 to differentiate them from other label names. And this dataset will be fit to the Logistic Regression by using Count Vectorizer technique and to compare it off, we will be passing it through Logistic Regression again but this time by using the BytePair embeddings since we are dealing with textual data values.

For this experiment, we will be using the values of hyperparameters which are used by the author to investigate bad labelling on google-emotions dataset.

[ ]:
```
#Topic to investigate labelling for
label_of_interest = 'topic name'
(df[['Keyword', label_of_interest]]
  .loc[lambda d: d[label_of_interest] == 1]
  .sample(4))
```

[ ]:
```
X, y = list(df['Keyword']) , df[label_of_interest]
print(f'Number of keywords: {len(X)}, Number of Labels: {len(y)}')

pipe = make_pipeline(
    CountVectorizer(),
    LogisticRegression(class_weight = 'balanced', max_iter = 1000))
```

[ ]:
```
from sklearn.pipeline import make_union
from whatlies.language import BytePairLanguage

pipe_emb = make_pipeline(
    make_union(
        BytePairLanguage("en", vs=1_000),
        BytePairLanguage("en", vs=100_000)
    ),
    LogisticRegression(class_weight='balanced', max_iter=1000)
)
```

[ ]:
```
#Training both pipelines
pipe.fit(X,y)
pipe_emb.fit(X,y)
```

0.0.6 Task 6: Assign the doubts from the github page of the author.

Doubts are specified reasons the technique performes internally to check wheather the labels match or not.

Following are the specified reasons the technique performs: 1. `proba`: Assigns doubt when the `pipe` pipeline doesn't predict any label with a high confidence. 2. `disagree`: Assigns doubt when the `pipe` pipeline doesn't agree with their `pipe_emb` pipeline. So, when they do not match. 3. `short_pipe`: Assigns doubt when `pipe` pipeline predicts correct labels with a low confidence. 4. `short_pipe_emb`: Assigns doubt when the `pipe_emb` predicts the correct label with a low confidence.

[ ]:
```
reasons = {
    'proba': ProbaReason(pipe),
    'disagree': DisagreeReason(pipe, pipe_emb),
    'short_pipe': ShortConfidenceReason(pipe),
    'short_pipe_emb': ShortConfidenceReason(pipe_emb),
}

doubt = DoubtEnsemble(**reasons)
```

[ ]:
```
# Return a dataframe with reasoning behind sorting
predicates = doubt.get_predicates(X, y)

# Use predicates to sort original dataframe
df_sorted = df.iloc[predicates.index][['Keyword',label_of_interest]]

# Create a dataframe containing predicates and original data
df_label = pd.concat([df_sorted, predicates], axis=1)
```

[ ]:
```
(df_label[['Keyword', label_of_interest]]
  .head(20))
```

Below, we can see the keywords which were labelled as `topic name` in the input dataset.

[ ]:
```
labeledas_topicname = (df_label[['Keyword', label_of_interest]]
  .loc[lambda d: d['topic name'] == 1])
labeledas_topicname.sample(20)
```

From the above results, we can convey that there are all of the keywords are related to some type of topic name activity. This shows us that the topic name keywords are somewhat correctly labelled.

[ ]:
```
labeledas_topicname[labeledas_topicname['Keyword'].str.contains("substring1" or␣
↪"substring2")]
```

[ ]:
```
(df_label
 .sort_values("predicate_disagree", ascending=False)
 .head(20)[['Keyword', label_of_interest]]
 .drop_duplicates())
```

[ ]:
```
df[df['Keyword'] == 'keyword']
```

0.0.7 Task 7: Investigate each reasons individually and extract keywords which do not match with their assigned label names.

0.0.8 CountVectorizer shot on Confidence

The following dataset contains all the keywords which should have been labeled as topic `topic name` with a high confidence but were not. This is taken out from the countvectorizer technique which transfers strings into vectors.

[ ]:
```
(df_label
 .sort_values("predicate_short_pipe", ascending=False)
 .head(20)[['Keyword', label_of_interest]]
 .drop_duplicates())
```

By looking at the resultant keywords from the count vectorizer technique, we can find keywords which contain the sub-string `substring1 or substring2` but aren't labelled as topic name label. These set of keywords are interesting as it shows explicitly these keywords needs some extra attention as to understand what label they were assigned to. For this, we will take some keywords and find what labels they were assigned to. df[df['Keyword'].isin(['keyword1','keyword2','keyword3'])]

1. From the above subset, we can capture a trend that states if there are two possible label names in a keyword then, that keywords can be assigned to any one of the label. Therefore, if these keywords as assigned to either one of the label then we can lay trust of these label assignment.
2. But it's vital to notice that there are some keywords which do not follow the above trend, for example, the keywords that explicitly contain a different label name in it. These type of keywords could be the one that would be transferred after also consulting with the SEO specialists.

0.0.9 CountVectorizer with Low Proba

For this reason, we get the list of keywords for which the technique was not confident in capturing the assignment. This could be due to multiple labels being present in the keyword or a totally new type of keyword which doesn't include any label name.

[ ]:
```
(df_label
 .sort_values("predicate_proba", ascending=False)
 .head(10)[['Keyword', label_of_interest]]
 .drop_duplicates())
```

By looking at the top 10 keywords for this list, we can find keywords which contain double label names mostly relating to topic name and topic name. These results can be combined with the above results to look at them together.

0.0.10 BytePair Embeddings short on Confidence

This reasoning is based on word embeddings. These embeddings are pooled together before passed to the classifier. BytePair Encodings are word embeddings that are precomputed on the sub-word

level. This means they are able to embed any word by splitting words into subwords and looking up their embeddings. For example: unfortunately into un and fortunately. Whereby into where and by and so on...

[ ]:
```
topicname_byte_re = (df_label
 .sort_values("predicate_short_pipe_emb", ascending=False)
 .sample(10)[['Keyword', label_of_interest]]
 .drop_duplicates())
topicname_byte_re
```

[ ]:
```
df[df['Keyword'].isin(['Keyword1','Keyword2','Keyword3'])]
```

1. From the above results from the embeddings, we can conclude that the keywords in this list seem to have a different trend. By using the embeddings, we get some objects that can be used to topic name something such as the 'keyword1' is a topic name which is correctly labelled but it quite possibly can be used for topic name.
2. Secondly, keywords contain sub string `substring` have also been captured in here, which is also interesting to look as as how find a correlation between our topic name and topic name labels.

0.0.11 Task 8: Evalute the keywords and store the subset dataset for a label in a pickel for future use.

[ ]:
```
topicname_byte_re.to_pickle('topicname_bad_labelling')
```

0.0.12 Conclusions:

From the above experiment we found out some interesting trends and insights for our label `topic name`. While there were some keywords which contained keywords aiming towards performing a specific task with a much better way, there were some substrings and objects that could also be referred to topic name. Both of the results cpatured by the count vectorization and the embeddings were useful and should be clubbed and discussed together with the SEO specialist for planning the next steps for these.

A similar group of steps were performed for comparing it to the `topic name` label and for other labels in our datasets as well.

[ ]:
```

```