Predicting Effective Arguments in an Essay

By: Srihari Inukurthi and Jason Lee

Project Description

- Aim is to create a model to predict the effectiveness in an argumentative essay
- Essays will be evaluated and assessed into three categories:
  - Effective
  - Adequate
  - Ineffective
- Motivation is to create an automated feedback for students to instantly get reports of their writing

Dataset Reading and Exploratory Data Analysis

```python
train = pd.read_csv("Feedback_Predicting_Effective_arguments/feedback-prize-effectiveness/train.csv")
test = pd.read_csv("Feedback_Predicting_Effective_arguments/feedback-prize-effectiveness/test.csv")
submission = pd.read_csv("Feedback_Predicting_Effective_arguments/feedback-prize-effectiveness/sample_submission.c
```

Python

+ Code + Markdown

```python
# print(train)
print(train.shape)
# print(train.head)
print(train.columns)
```

Python

```
(36765, 5)
Index(['discourse_id', 'essay_id', 'discourse_text', 'discourse_type',
       'discourse_effectiveness'],
      dtype='object')
```

```python
print(test.head)
print(test.shape)
print(test.columns)
```

```python
def missing_values(dataframe):
    tuple_list = []
    for col in dataframe.columns:
        # print(col)
        missing_count = dataframe[col].isna().sum()
        if missing_count > 0:
            tuple_list.append(tuple([col, missing_count]))
    tuple_list = sorted(tuple_list, key=lambda x: x[1], reverse=True)
    return tuple_list
```

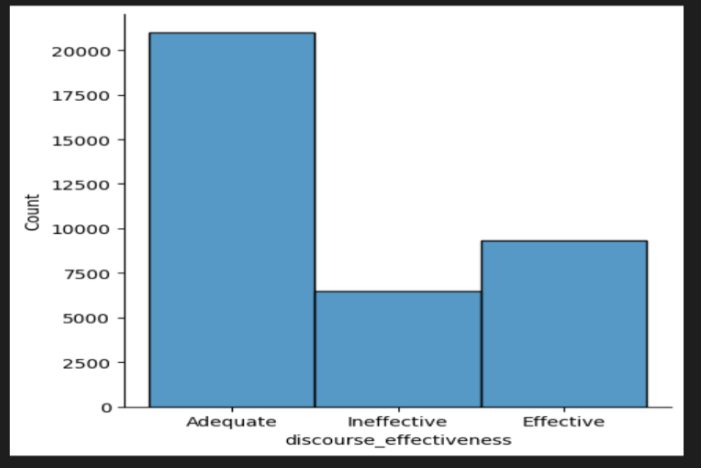
```python
print(missing_values(train))
```

```
[]
```

Dataset Reading and Exploratory Data Analysis

```
sb.displot(train['discourse_effectiveness'])
```

```
<seaborn.axisgrid.FacetGrid at 0x19382503e10>
```




```
train['discourse_effectiveness'] = train['discourse_effectiveness'].replace({'Adequate':2,'Ineffe
```

```
print(train['discourse_effectiveness'].value_counts())
print(train['discourse_effectiveness'].value_counts() / len(train['discourse_effectiveness']))
```

```
discourse_effectiveness
2    20977
1     9326
3     6462
Name: count, dtype: int64
discourse_effectiveness
2    0.570570
1    0.253665
3    0.175765
Name: count, dtype: float64
```

Data cleaning and Pre processing

```python
def read_discourse_paper_train():
    return train['essay_id'].apply(lambda x: open(f'Feedback_Predicting_Effective_arguments/feedback-pri
def read_discourse_paper_test():
    return test['essay_id'].apply(lambda x: open(f'Feedback_Predicting_Effective_arguments/feedback-priz
```

✓ 0.0s

```python
train['discourse_paper'] = read_discourse_paper_train()
test['discourse_paper'] = read_discourse_paper_test()
print(train.shape)
print(train.columns)
print(test.columns)
print(test.shape)
```

✓ 3.3s

```
(36765, 6)
Index(['discourse_id', 'essay_id', 'discourse_text', 'discourse_type',
       'discourse_effectiveness', 'discourse_paper'],
      dtype='object')
Index(['discourse_id', 'essay_id', 'discourse_text', 'discourse_type',
       'discourse_paper'],
      dtype='object')
(10, 5)
```

```python
target_class = train['discourse_effectiveness']
new_train = train.drop(['discourse_effectiveness'], axis=1)
print (new_train['discourse_type'].value_counts())
print(new_train['discourse_type'].value_counts()/len(new_train['discourse_type']))
print(len(new_train['discourse_paper']))
```

✓ 0.0s

```
discourse_type
Evidence                12105
Claim                   11977
Position                 4024
Concluding Statement     3351
Lead                     2291
Counterclaim             1773
Rebuttal                 1244
Name: count, dtype: int64
discourse_type
Evidence                0.329253
Claim                   0.325772
Position                0.109452
Concluding Statement    0.091146
Lead                    0.062315
Counterclaim            0.048225
Rebuttal                0.033837
Name: count, dtype: float64
36765
```

Data cleaning and Pre processing

```python
import nltk
# from ntlk import stopwords
import re as re
text = new_train['discourse_paper']
print(len(text))
def clean_text(text):
    text = text.replace('#', ' ')
    return text
text = clean_text(text)
print(text)
```

```
36765
0        Hi, i'm Isaac, i'm going to be writing about h...
1        Hi, i'm Isaac, i'm going to be writing about h...
2        Hi, i'm Isaac, i'm going to be writing about h...
3        Hi, i'm Isaac, i'm going to be writing about h...
4        Hi, i'm Isaac, i'm going to be writing about h...
                                   ...
36760    Some people may ask multiple people for advice...
36761    Some people may ask multiple people for advice...
36762    Some people may ask multiple people for advice...
36763    Some people may ask multiple people for advice...
36764    Some people may ask multiple people for advice...
Name: discourse_paper, Length: 36765, dtype: object
```

```python
def convert_shorts(text, contractions):
    for word in text.split():
        if word in contractions:
            text = text.replace(word, contractions[word])
    return text
print(text.head())
text = text.apply(lambda x: convert_shorts(x, contractions))
print(text.head())
```

[19] ✓ 1.3s

```
0    Hi, i'm Isaac, i'm going to be writing about h...
1    Hi, i'm Isaac, i'm going to be writing about h...
2    Hi, i'm Isaac, i'm going to be writing about h...
3    Hi, i'm Isaac, i'm going to be writing about h...
4    Hi, i'm Isaac, i'm going to be writing about h...
Name: discourse_paper, dtype: object
0    Hi, I am Isaac, I am going to be writing about...
1    Hi, I am Isaac, I am going to be writing about...
2    Hi, I am Isaac, I am going to be writing about...
3    Hi, I am Isaac, I am going to be writing about...
4    Hi, I am Isaac, I am going to be writing about...
Name: discourse_paper, dtype: object
```

Data cleaning and Pre processing

```python
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
target_class = pd.DataFrame(target_class)
column_trans = make_column_transformer((OneHotEncoder(), ['discourse_effectiveness']), remainder='passthrough')
y = pd.DataFrame(column_trans.fit_transform(target_class), columns=column_trans.get_feature_names_out())
print(y.columns)
print(y.shape)
```

✓ 0.0s

```
Index(['onehotencoder__discourse_effectiveness_1',
       'onehotencoder__discourse_effectiveness_2',
       'onehotencoder__discourse_effectiveness_3'],
      dtype='object')
(36765, 3)
```

Building Model

1. Before building a model, we need to convert all the text sentences into a vectors.
2. This is considered as a partial classification and partial regression problem.
3. We plan on using 2 models one on top of the other.
4. We use a classification algorithm and on top of that we use a regression algorithm that can produce multiple predictions of a target.

Future plans

1. After creating the model, we plan to evaluate the model based on the given test samples.
2. If the model performance is not good enough, we plan to improve the model's performance by fine tuning the model.
3. If the fine tuning is not very much helpful, we plan on using a different model as a backup that is helpful in predicting the effective arguments in an essay.